석 사 학 위 논 문
Master's Thesis

메모리 효율적인 NBNN 이미지 분류

Memory-Efficient NBNN Image Classification

2016

이 윤 석 (李 侖 錫 Lee, YoonSeok)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

메모리 효율적인 NBNN 이미지 분류

2016

이 윤 석

한 국 과 학 기 술 원

전산학부

메모리 효율적인 NBNN 이미지 분류

이 윤 석

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2016년 6월 9일

심사위원장 윤 성 의

심 사 위 원 최 성 희

심 사 위 원 Martin Ziegler

Memory-Efficient NBNN Image Classification

YoonSeok Lee

Advisor: Sung-eui Yoon

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in Engineering in Computing

Daejeon, Korea
June 9, 2016

Approved by

Sung-eui Yoon
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

MCS
20144503

이윤석. 메모리 효율적인 NBNN 이미지 분류. 전산학부 . 2016년. 15+iii 쪽. 지도교수: 윤성의. (영문 논문)
YoonSeok Lee. Memory-Efficient NBNN Image Classification. School of Computing . 2016. 15+iii pages. Advisor: Sung-eui Yoon. (Text in English)

초 록

NBNN은 이론적으로 간결하며 구현이 용이한 최근접 이웃 기반의 이미지 분류기로 표현자 양자화를 거치지 않아 표현자의 분별력을 잘 유지하며 클래스의 특성을 일반화하는 능력이 뛰어나지만, 데이터 양이 많아지면 질의 시간이 오래 걸리는 문제를 안고 있다. 본 논문에서는 NBNN과 그 후속 연구에 Spherical hashing을 적용하고, 이진 코드의 최근접 이웃 연산에 적합한 계층적 인덱싱 방법을 제안하여 확장성 문제를 해결하고자 한다. 제안된 해싱 기법을 적용하면 이전의 연구와 비교하여 분류 정확도와 질의 시간은 비슷한 수준으로 유지하면서 메모리 사용량을 수십 배 가량 감소시킬 수 있다. 또한, 확장성 문제가 더욱 심해지는 고차원 표현자를 이용한 이미지 분류 작업에서 이러한 이점이 더욱 극대화될 수 있다.

핵 심 낱 말 이미지 분류, NBNN, 해싱, 메모리 최적화, 인덱싱

Abstract

NBNN is a simple image classifier based on identifying nearest neighbors. NBNN uses original image descriptors (e.g., SIFTs) with vector quantization for preserving the discriminative power of descriptors and has a powerful generalization characteristic. However, it has a distinct disadvantage; its memory requirement can be prohibitively high as we have a large amount of data. We identify this problem of NBNN techniques and we apply a binary code embedding technique, i.e., spherical hashing, to encode data compactly without a significant loss of classification accuracy. We also propose to use an inverted index to identify nearest neighbors among those binarized image descriptors. To demonstrate benefits of our method, we apply our method to two of existing NBNN techniques with a image dataset. By using 64 bit lengths, we are able to observe 16 times memory reduction with a higher performance without a significant loss of classification accuracy. This result is thanks to our compact encoding of image descriptors without losing much information of original image descriptors.

Keywords Image classification, NBNN, hashing, memory efficiency, indexing

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Image classification has been researched as an important task of the computer vision field for a long time. Image classification is a task of assigning an appropriate class label to a query image.

Lately, deep CNNs(Convolutional Neural Networks)[15] are receiving attention on the strength of their high classification accuracy and fast query speed, and its follow-up studies are being actively conducted. [6] [28] Likewise, recently proposed methods for image classification mostly utilize benefits of CNNs, however, it doesn't make image classifiers useless which were studied before. There is a study[27] in the direction of combining an existing classifier with CNNs, and previous image classifiers may be used in the situation where using deep CNNs is not appropriate because of its long training time.

Among many available classification techniques, Naive Bayes Nearest Neighbor (NBNN) [2] is one of the popular image classifiers that does not require an explicit learning process. NBNN is designed based on the naive Bayes assumption and using nearest neighbor search. It usually uses local descriptors (e.g., SIFTs), which is densely extracted from a query image for the classification. Unlike many other conventional image classifiers, NBNN does not perform descriptor quantization like bags-of-words for compact representation. Instead, NBNN utilizes original image descriptors as they maintain discriminative power as original descriptors. NBNN measures "Image-to-class" distances for all the classes by identifying the nearest neighbor for each local descriptor and assigns a class which has the minimum sum of distances as the class type of a query image.

Thanks to characteristics of NBNN, the NBNN approach has advantages over other learning based techniques for image classification. NBNN is theoretically simple and easy to implement. As a result, it is also easy to modify NBNN for a particular purpose. For example, NBNN is adjusted for solving domain adaptation problems [24]. Furthermore, NBNN shows high generalization power [16] **YOON: cite: When Naive Bayes Nearest Neighbors Meet Convolutional Neural Networks YS: Added** , since it works mainly in a data-driven way without tuning parameters for a particular dataset.

Nonetheless, NBNN has certain drawbacks such as low accuracy compared to recent convolutional neural net based approaches, slow runtime performance, and high memory requirement. Accuracy and slow performance have been addressed by many prior approaches [25, 18, 23, 16], but the memory issue is not addressed well.

In this paper, we propose a memory-efficient NBNN technique. To compact represent image descriptors, we apply a binary code embedding technique to map original local image descriptors into short binary codes. We then perform fast approximate nearest neighbor search by using an inverted index structure built from those binary codes. To verify benefits of our method, we test our method against a standard image dataset, and compare our method against two well-known NBNN approaches, the original NBNN and the local NBNN that improves the performance of the original NBNN. By using our method, we are able to observe faster running performance and lower memory requirement without a significant loss of classification accuracy. Especially, when we use 64 bit lengths for binary codes, we are able to achieve 16 times memory reduction over those two NBNN approaches, while 12 times and 1.125 times faster running performance over the original and local NBNNs, respectively **YOON: Fix numbers.. YS: Fixed** . This result is mainly thanks to accurately embedding original descriptors into binary codes.

Chapter 2. Related Work

In this section, we review prior approaches that are directly related to our method.

2.1 NBNN

NBNN[2] uses original image descriptors to preserve the discriminative power of the features instead of using descriptor quantization method like bag-of-words, which is used in many other image classifiers. In addition, NBNN utilizes image-to-class distance metric in order to generalize the characteristics of each class in contrast to other methods using classical image-to-image distance. Therefore, it can classify images successfully by searching similar local descriptors to the query descriptors among all the descriptors in the certain class in the labeled dataset even if there is no similar single image to the query image in the dataset.

To address drawbacks of the original NBNN and extend it to other related problems, many studies have been proposed. Optimal NBNN [1] studied parameters to consider the assumptions that were made for designing NBNN, and dependencies among the local features are also studied [23]. Recently, NBNN was utilized for data adaptation problem[24], and image retrieval[27]. In order to address this issue of a high runtime overhead in querying, McCann et al. [18] proposed local NBNN, which only calculates the distance from the query descriptors to others in a single time, instead of all the descriptors in every class. However, the memory scalability problem arisen from using unquantized original descriptors is not considered yet.

Nearest neighbor search. Exact or approximate nearest neighbor search has been widely studied. One of most common acceleration data structures is kd-trees [5]. kd-trees were also widely adopted in many computer vision techniques and various optimization techniques with kd-trees have been proposed [14]. Some of well-known optimization techniques in the computer vision field include randomized kd-trees [22] and relaxed orthogonality of partition axes [12]. Muja and Lowe [19] have proposed an automatic parameter selection algorithm of some of aforementioned techniques (e.g., [22]). Nonetheless, many hierarchical techniques including ones based on kd-trees have been known to work ineffectively for high dimensional problems.

2.2 Hashing

As an approximate, yet scalable nearest neighbor search approach, hashing techniques have been extensively studied recently. These techniques can be broken into two categories: data-independent and data-dependent techniques. Data-dependent techniques [26, 7] can produce more high accuracy for the search problem, by computing hashing functions considering input data. Unfortunately, most these techniques tend to rely upon learning techniques or to require high computation time and thus we focus on data-independent techniques, which are more suitable for NBNN approaches.

The most well-known technique under the data-independent category is locality sensitive hashing [10]. This technique draws hyperplanes randomly from a certain distribution function, and use them

for hashing functions. This technique has been generalized into many different directions including ones to support different distance metrics [3] and GPU acceleration [21].

These hashing functions can be used for encoding input data into binary codes. Recently, hypersphere based hashing function and binary code embedding techniques is proposed [8]. This technique can generates more closed regions in high dimensional spaces, resulting in a high accuracy for approximate neighbor search. This property can preserve the distances between the original data well with their corresponding binary codes. Thanks to this high accuracy, we adopt to use it for encoding image descriptors and using their binary codes for NBNN techniques.

Chapter 3. Memory-Efficient NBNN

In this section, we first explain the original NBNN technique. We then explain two main components of our method: binarization and inverted indexing.

3.1 NBNN based classification

Let us represent an image I as a set of local descriptors, i.e., $I = \{d_1, d_2, \cdots, d_n\}$. In order to classify the image with NBNN, we define and measure the image-to-class distance, D_{ItC}, which uses a descriptor-to-class distance, D_{DtC}. We also define $NN_c(d)$ to be the nearest neighbor descriptor to the given descriptor d among descriptors assigned to the class c. The descriptor-to-class and image-to-class distances can be then defined as follows:

$$D_{DtC}(d, c) = ||d - NN_c(d)||, \tag{3.1}$$

$$D_{ItC}(I, c) = \sum_{i=1}^{n} D_{DtC}(d_i, c), \tag{3.2}$$

where n is the number of the local descriptors extracted from the image I.

NBNN identified a class of an image I according to the following equation, which is derived by simplifying the maximum likelihood classifier based on the naive Bayes probabilistic model [2]:

$$\hat{c} = \underset{c}{argmin} D_{ItC}(I, c). \tag{3.3}$$

NBNN technique relies on computing the nearest neighbor given a descriptor. This nearest neighbor search is efficiently supported by Approximate Nearest Neighbor (ANN) search methods using kd-trees [5]. By utilizing kd-trees, we can achieve fast search performance. Nonetheless, we found that this nearest neighbor search is still the main bottleneck of NBNN and can take XXX% **YOON: fix** of the total computation of the NBNN method in our experiment. Furthermore, the memory requirement of storing local descriptors and such tree-based indexing structure is high.

3.2 Binarization of descriptors

Our main goal is to perform nearest neighbor search in a memory-efficient manner, which is the main computational component of NBNN techniques. Fortunately, nearest neighbor search has been studied well even for high-dimensional data such as our image descriptors. Especially, for such high-dimensional problems, hashing techniques have been demonstrated to work well and well-known examples include locality sensitive hashing [**?**]. These hashing techniques can work as binary code embedding that compactly represents data points based on hashing functions.

In order to present image descriptors as a binary code for our problem, we utilize spherical hashing [8]. Spherical hashing is one of the state-of-the-art methods to represent high dimensional points into compact binary codes. Most prior works used hyperplanes to partition data into two set and to encode those partitioned data with one bit (0 for one set or 1 for the other set).

On other other hand, spherical hashing computes binary codes based on hyperspheres, each of which tightly bounds input data. While $D+1$ hyperplanes are required to define a closed region in a D dimensional space, one hypersphere is enough to define such a closed region. In other words, the average of the maximum distance among points with the same binary code can be bounded, and thus errors caused by representing original data into such binary codes can be bounded too, resulting in higher approximate nearest search while compactly representing data. Thanks to this property, spherical hashing has been demonstrated to show higher accuracy over other hyperplane based techniques given the same number of bit lengths. Nonetheless, any binary code embedding techniques can be used instead of spherical hashing, our chosen method for this work.

Suppose that we represent an image descriptor, d, to a binary code b by using an binary code embedding or hashing method, $h(\cdot)$; i.e. $b = h(d)$. The image I is then represented as a set of binary codes, $I_b = \{b_1, b_2, \cdots, b_n\}$, which are computed by applying the hashing function to the original image descriptors.

Once we represent descriptors into binary codes, we cannot use distance functions defined with those image descriptors. Instead, we define a distance function between a binary code and a class, D_{BtC}, as the following:

$$D_{BtC}(b, c) = HD(b, NN_c(b)), \tag{3.4}$$

where $HD(\cdot, \cdot)$ is the Hamming distance between two binary codes. By replacing D_{DtC} by D_{BtC} in Eq. 3.2 and 3.3, we have the classification function for our method using binary codes:

$$D_{I_btC}(I, c) = \sum_{i=1}^{n} D_{BtC}(b_i, c), \tag{3.5}$$

$$\hat{c} = \underset{c}{argmin} D_{I_btC}(I, c). \tag{3.6}$$

While we can represent image descriptors with binary codes, we lose information of original image descriptor during binary code embedding. As a result, the accuracy of our approximate nearest neighbor search goes down, as a smaller bit is used for encoding binary codes. We discuss behaviors of accuracy and memory requirement of bit lengths in Sec. **?? YOON: this is critical info.**

YOON: show this in the result/discussion sec.
Show the trade-off graph between the accuracy and the bit length. Test it with different benchmarks. **YOON: Show the trade-off graph between the runtime perf. and the bit length. Test it with different benchmarks.** **YS: Need more experiments**

3.3 Indexing

We can save memory usage by applying binary hashing to the image descriptors. However, we still have the issue of query time scalability. As the nearest neighbor operation on raw image descriptors causes time scalability problem taking most of the query time in original NBNN, the nearest neighbor operation on binary code can also cause a similar problem if we take linear search algorithm to find the closest code. So we need proper indexing method to perform accurate and fast nearest neighbor search, however, ANN using kd-trees can't be applied to the nearest neighbor search on binary codes, because kd-tree assumes that the data points are located in the high dimensional spaces, and it loses most of advantages if we treat each bit of binary code as one dimension for kd-tree.

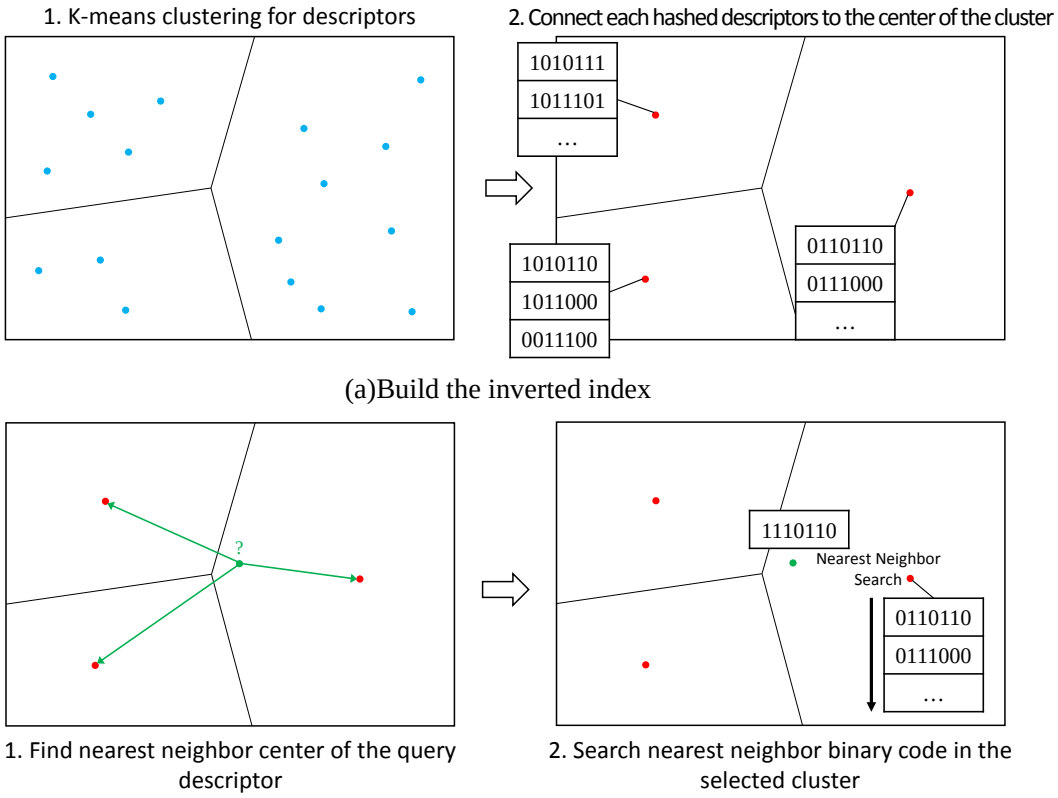

Figure 3.1: **YOON: Refine the fig.** **YS: Fixed** The top row(a) shows the inverted indexing structure for our method, while the bottom row(b) shows how to access the structure to identify nearest neighbors. Blue, red, green dots represent training image descriptors, cluster centers, and a query image descriptor, respectively.

To support an efficient search of identifying nearest neighbors to the given query, we adopt an inverted indexing structure as illustrated in Fig. 3.1. To build the inverted index, we perform the following steps:

1. **Computing clusters.** We perform k-means clustering on the original descriptors to build clusters. Any clustering methods can be used instead of the simple k-means clustering. Especially, product quantization has been demonstrated to work well with binary codes and high-dimensional descriptors [11].

2. **Assigning to the closet cluster.** For each original descriptor, we identify its closest cluster by computing the distance between the descriptor and centers of clusters. Instead of storing the original descriptor, we compute a binary code of the descriptor and associate the binary code with the cluster. We can then efficiently organize our inverted index with our binary codes. When we want to access their original descriptors and images, we also store these data associated with each cluster in a secondary memory space (e.g., disk).

For simplicity, we explained the simple, inverted index. Recently, multi-index has been proposed [9], and can be more complex, yet more efficient for large-scale problems.

At a query time, we use the computed inverted index as the following:

1. **Finding the nearest cluster.** Given a query, we identify the nearest cluster among the cluster centers.

2. **Identifying k nearest neighbors.** Given the nearest cluster, we access binary codes of image descriptors associated with the cluster. We first convert the image descriptor of the query into a binary code. We then measure the Hamming distances between binary codes of the query and others associated with the cluster. By performing sorting according to the computed Hamming distance, we can identify k nearest neighbors.

By using the inverted index, we can efficiently identify potential candidates of k nearest neighbors from the query data. The aforementioned inverted index requires the number clusters for computing center clusters. Depending on the number of clusters, we can control the number of descriptors per each cluster. In Sec. 4.2, we discuss effects of varying number of clusters.

Chapter 4. Experiments

In this section, we performed experiments to compare the performance of hashing applied NBNN to original NBNN and local NBNN. We have focused on the query time, classification accuracy and memory usage of NBNN image classification system.

4.1 Implementation and Datasets

We used 101 classes of Caltech-101 image dataset [4], excluding the background class. We utilized SIFT [17] as the local descriptor densely, which means we divide an image into grids instead of using an ordinary keypoint extracting algorithm, and extracted them in multi-scale.

We followed the experiment protocol laid out by the prior work [2] to set the experiment environment for our paper. We randomly choose 15 training images and 15 test images for each class. 64 bit code length is used, unless mentioned otherwise, when hashing is applied to descriptors.

We implemented NBNN [2] and local NBNN [18] based on guidelines mentioned in their corresponding papers. These methods utilize a fast approximate nearest neighbor search method, FLANN [20], to efficiently identify nearest neighbors based on kd-trees. We use L1 and L2 distance to calculate the distance between our image descriptors and use the Hamming distance to measure the distance between binary codes for our method. The number of clusters for inverted index structure and k value of k-nearest neighbor search in local NBNN are manually set.

YOON: this is too specific and is moved here for now. The ratio of the saved memory usage can be controlled by changing the number of bits in hashing function. For example, if we use 64 bits code length for our hashing function, which is long enough to maintain the classification accuracy in most cases, a single SIFT descriptor whose size is 128byte can be reduced by 16 times. Even though the order of the space complexity is remain unchanged, reducing space usage by being divided by a big constant is highly effective especially when the raw data size which is bigger than hardware memory capacity can be shrinked into the size that fits to machine's memory size, due to the difference of accessing speed between main memory and auxiliary memory like hard disk.

4.2 Result

We performed experiments to compare the performance of our method, applied NBNN classifier with spherical hashing (NBNN+SH), with the original NBNN. We also apply spherical hashing to Local NBNN (Local NBNN+SH) with existing Local NBNN [18]. The classification accuracy and query time of tested classification methods are shown in Fig 4.1. We measure classification accuracy as the ratio of the correctly classified query images over all the test images. The average query time per image is calculated by measuring the total time taken for classifying all test images and dividing it by the number of test images.

We set the number of clusters as 30 in NBNN+SH and 2000 in Local NBNN+SH **YOON: Explain why we use different numbers for these methods** . **YS: Explained** We set these parameters differently because when the number of descriptors in an indexing scheme becomes bigger, the number of clusters should be bigger for better performance. For NBNN, the number of descriptors in a single

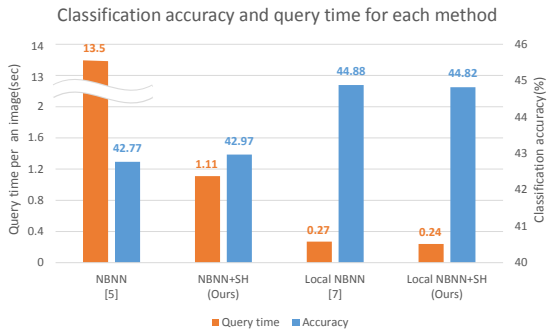

Figure 4.1: Classification accuracy and query time on average of different methods. **YOON: show bar for the accuracy** **YS: Fixed**

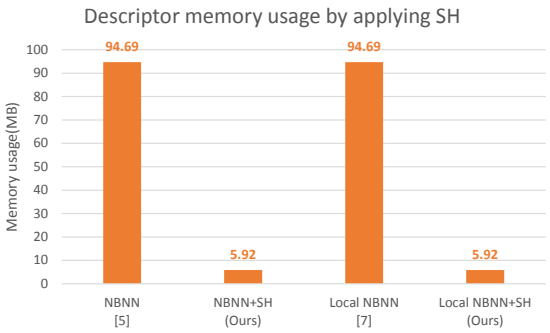

Figure 4.2: Memory requirement of different methods. **YOON: Show results of Local methods.** **YS: Fixed**

indexing scheme is much smaller than the case of Local NBNN, because NBNN builds an indexing structure for each class, while Local NBNN manages all the descriptors in a single indexing structure.

First of all, we observe faster running time and higher accuracy by using the local NBNN over the original NBNN, as demonstrated by the paper of local NBNN [18]. Furthermore, by using our method applying the spherical hashing to those prior NBNN techniques, we are able to observe higher accuracy without a significant loss of accuracy. For the case of NBNN, the query performance of NBNN+SH is more than 10 times faster than that of NBNN with a slight accuracy improvement. This drastic performance improvement is achieved mainly because computing the Hamming distance between binary codes is much faster than the Euclidean computation between the original SIFT descriptors. We think that hashing, a type of dimension reduction techniques, cancels variance of image descriptors of the same object, resulting in a slightly higher accuracy in this case.

Comparing Local NBNN+SH to Local NBNN, their query speeds don't show big difference while the classification accuracies are also similar. It is because more distances **YOON: Measure this and support this claim** need to be calculated even if the distance computation between a single pair of binary codes is still faster than computation between a pair of SIFT descriptors. **YS: Added here** In each cluster in the inverted index, the linear search has to be performed rather than searching in the k-d trees based indexing whose time complexity is in log scale.

We also measure the memory requirement of different methods. However, in memory usage, the case which Spherical hashing applied show a significant advantage over the original one because only 8 bytes are necessary to represent a binary code while 128 bytes are needed to represent one SIFT descriptor. This difference results in 16 times less memory usage excluding the overhead for constructing indexing scheme, while preserving classification accuracy and query time. Fig 4.2 shows the required memory size to represent descriptors in our experiment environment. Considering that the SIFT descriptor is relatively lower dimensional data among image descriptors, more advantage will be given in the higher dimensional space.

We also investigate effects of different number of clusters of our indexing structure used with the binary codes (Table 4.1). In all the tested cases, the classification accuracies are similar to each other, ranging range between 43% and 46%. 4.1, The query time is getting smaller when the number of clusters gets larger, but gets longer when the number of clusters becomes too large for the tested dataset, i.e., 3 k clusters. When we have a small number of clusters, finding the nearest cluster is fast, but the cluster

Table 4.1: Effects with varying numbers of clusters for Local NBNN+SH. **YOON: Show accuracy and num. of images**

	The number of clusters							
	50	100	500	1000	1500	2000	2500	3000
Query time (sec)	3.77	1.90	0.48	0.29	0.25	0.24	0.43	0.43
Finding clusters (sec)								
Avg. num. of descriptors per cluster	7756	3878	775	387	258	193	155	129
Accuracy	44.03	43.17	44.03	44.95	45.54	44.82	44.62	44.75

has many images and thus require a long computation time to find the nearest image from those images associated with the cluster. On the other hand, when the number of clusters is too high, finding the nearest cluster is high, resulting in longer computation time. Given this trade-off, the best performance is achieved when we have 2 k clusters for the tested benchmark.

YOON: Why we achieve such results? show the 2d graph of H-dist and E. dist.

YOON: Optionally, why we achieve even higher accuracy? - while we quantize descriptors, we cancel out noise or variance of original descriptors. This requires further study.

YOON: Show results w/ big benchmarks. **YS: Need more experiments**

Chapter 5. Conclusion & Future Works

YOON: Mention that we can try recent CNN or global features. YOON: mention that we can use recent multi-dimensional index, cite JP's paper, and MY's dependency features, cite MY's paper.

In this paper, we applied binary hashing method to NBNN based image classifiers and tested indexing structure for nearest neighbor search on binary codes. Using Caltech-101 image dataset, we studied the proposed method shows similar or better classification accuracy and query speed while saving memory usage about 16 times. We expect that this memory saving advantage will be more noticeable in the higher dimensional space because binary hashing techniques are usually getting more efficient in the higher dimensional space. However, in that case, we have to carefully consider the overhead of the hashing method because it is expected to become larger when the dimension of the space gets higher.

For the further research, we can utilize global image features which recently studied like CNN [15] or VLAD [13]. Because the NBNN classifiers assume image descriptors as local features, so how to use global feature in NBNN can be an interesting research problem. We can also apply recent techniques about shortlist selection and distance estimation [9] ,which works well in inverted index and multi-index scheme, for faster k-nearest neighbor operation. We can also try to consider dependencies among the image descriptors referring prior work [23] to improve the quality of classification. We believe that this study will help to solve scalability of NBNN based image classifiers.

Bibliography

[1] R. Behmo, P. Marcombes, A. Dalalyan, and V. Prinet. Towards optimal naive bayes nearest neighbor. In *ECCV*, 2010.

[2] O. Boiman, E. Schechtman, and M. Irani. In defense of nearest neighbor based image classification. In *CVPR*, 2008.

[3] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, 2004.

[4] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPRW*, 2004.

[5] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[7] Y. Gong and S. Lazebnik. Iterative quantization: a procrustean approach to learning binary codes. In *CVPR*, 2011.

[8] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *CVPR*, 2012.

[9] J.-P. Heo, Z. Lin, X. Shen, J. Brandt, and S. eui Yoon. Shortlist selection with residual-aware distance estimator for k-nearest neighbor search. In *CVPR*, 2016.

[10] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

[11] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128, 2011.

[12] Y. Jia, J. Wang, G. Zeng, H. Zha, and X.-S. Hua. Optimizing kd-trees for scalable visual descriptor indexing. In *CVPR*, 2010.

[13] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010.

[14] K. Kim, M. K. Hasan, J.-P. Heo, Y.-W. Tai, and S.-E. Yoon. Probabilistic cost model for nearest neighbor search in image retrieval. *Computer Vision and Image Understanding (CVIU)*, 2012.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] I. Kuzborskij, F. M. Carlucci, and B. Caputo. When naïve bayes nearest neighbors meet convolutional neural networks. In *CVPR*, 2016.

[17] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110.

[18] S. McCann and D. G. Lowe. Local naive bayes nearest neighbor for image classification. In *CVPR*, 2012.

[19] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Int'l Conf. Computer Vision Theory and Applications*, 2009.

[20] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP*, 2009.

[21] J. Pan and D. Manocha. Fast gpu-based locality sensitive hashing for k-nearest neighbor computation. In *ACM SIGSPATIAL GIS*, 2011.

[22] C. Silpa-Anan, R. Hartley, S. Machines, and A. Canberra. Optimised kd-trees for fast image descriptor matching. In *CVPR*, 2008.

[23] M. Sun, Y. Lee, and S.-E. Yoon. Relation based bayesian network for nbnn. *JCSE*, 9(4):204–213, 2015.

[24] T. Tommasi and B. Caputo. Frustratingly easy nbnn domain adaptation. In *ICCV*, 2013.

[25] T. Tuytelaars, M. Fritz, K. Saenko, and T. Darrell. The nbnn kernel. In *ICCV*, 2011.

[26] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, 2010.

[27] L. Xie, R. Hong, B. Zhang, and Q. Tian. Image classification and retrieval are one. In *ICMR*, 2015.

[28] N. Zhang, J. Donahue, R. Girshick, and T. Darrell. Part-based R-CNNs for fine-grained category detection. In *ECCV*, 2014.

Acknowledgments in Korean

Curriculum Vitae in Korean

이 름: 이 윤 석

생 년 월 일: 1988년 5월 14일

주 소: 대전 유성구 대학로 291 한국과학기술원 전산학부 3440호

전 자 주 소: kstylee@gmail.com

학 력

2004. 3. – 2007. 2. 한국과학영재학교

2007. 2. – 2014. 2. 한국과학기술원 전산학과 (학사)

경 력

2013. 1. – 2013. 11. 맘모스 개발팀장(창업자)