

# Fast and Accurate Handwritten Character Recognition using Approximate Nearest Neighbours Search on Large Databases

Juan C. Perez-Cortes, Rafael Llobet and Joaquim Arlandis

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Camino de Vera, s/n 46071 Valencia (SPAIN) , jcperez@disca.upv.es

**Abstract.** In this work, fast approximate nearest neighbours search algorithms are shown to provide high accuracies, similar to those of exact nearest neighbour search, at a fraction of the computational cost in an OCR task. Recent studies [26], [15] have shown the power of  $k$ -nearest neighbour classifiers ( $k$ -nn) using large databases, for character recognition. In those works, the error rate is found to decrease consistently as the size of the database increases. Unfortunately, a large database implies large search times if an exhaustive search algorithm is used. This is often cited as a major problem that limits the practical value of the  $k$ -nearest neighbours classification method. The error rates and search times presented in this paper prove, however, that  $k$ -nn can be a practical technique for a character recognition task.

**Keywords:** Handwriting Recognition, OCR, Fast Nearest Neighbour Search, Approximate Search,  $k$ -NN.

## 1 Introduction

Statistical non-parametric methods, such as  $k$ -nearest neighbours classifiers, are receiving renewed attention in the latter years, since very good results are being reported on many pattern recognition tasks (e.g.[28],[3]). Their theoretical properties, already known for at least three decades, are also being revisited and restated under milder assumptions [21], [9].

One of the basic requirements for these methods to obtain good performances, however, is the access to a very large database of labeled prototypes. In some tasks, like handwritten character recognition, collecting a large number of examples is not as hard as in other applications, but searching through the whole database to find the nearest objects to a test image is time-consuming, and has to be done for every character in a document. This has been a recurring argument against the use of  $k$ -nn for this task, since a character recognizer is supposed to carry out many classifications per second to on a moderately powerful machine to be useful and competitive. Additionally, the whole database has to be stored in main memory to perform the search efficiently, since access to secondary storage would penalize even further the search time.

The fast increase in the memory capacity of recent computers alleviates the space occupation problem, making possible the storage of up to tens of millions of prototypes in a typical personal computer. The speed problems, on the other hand, can be approached from two (potentially complementary) points of view: trying to reduce the number of prototypes without degrading the classification power, or using a fast nearest neighbours search algorithm.

The first approach has been widely studied in the literature, with techniques like editing [29], [8], condensing [16], LVQ [19], DSM [12], and their multiple variations being the better known representatives. These methods, have shown good results, equaling or sometimes improving the classification rates of the  $k$ -nn rule. Their power resides in the smoother discrimination surfaces they yield, by eliminating not only the redundant prototypes, but also the dubious ones that appear to “get into other classes’ regions” and give rise to highly intricate separation surfaces. Smooth discrimination surfaces mean less risk of over-learning (good performance on the training set, or on a given validation set being used to guide the process or set the parameters, but poorer results on an unseen test set). In a pure  $k$ -nn classifier with a large database, this feature is provided by an adequate choice of  $k$ , which usually has to be made larger as the size of the database grows.

The second approach, adopted in this work, has also been extensively studied in the literature. A number of methods exist to reduce the cost of an exhaustive search of the prototypes set to find the  $k$  nearest neighbours to a test point. A brief review of these techniques is presented in the next section.

## 2 Fast nearest neighbour search methods

The nearest neighbour search problem can be formulated in several distinct domains: from Euclidean vector spaces to (pseudo)metric spaces. Most algorithms intended for vector spaces are directly based on the construction of a data structure known as  $kd$ -tree [10], [4], [18]. A  $kd$ -tree is a binary tree where each node represents a region in a  $k$ -dimensional space. Each internal node also contains a hyperplane (a linear subspace of dimension  $k-1$ ) dividing the region into two disjoint sub-regions, each inherited by one of its sons. Most of the trees used in the context of our problem divide the regions according to the points that lay in them. This way, the hierarchical partition of the space can either be carried out to the last consequences to obtain, in the leaves, regions with a single point in them, or can be halted in a previous level so as each leaf node holds  $b$  points in its region. In [27], a very illustrative general exposition of the methods based on  $kd$ -trees is presented, along with a general scheme that allows the reader to clearly identify the different existing variants and choose one (or a combination of them) according to the peculiarities of the problem. In [31] can be found a thorough study of the cost of many  $kd$ -tree algorithms for different sample sizes, dimensionalities and values of  $b$ .

Other algorithms, as the one proposed by Fukunaga and Narendra [11], perform a hierarchical partition of the space resorting to concepts different to those

used in the *kd*-trees, for example, by means of a hierarchical clustering of the data. The search in the trees obtained by these methods is performed applying similar concepts to the ones discussed for the classical *kd*-trees. Other data structures intended to improve on the *kd*-trees are for example the VP-trees [30] and the Geometric Near-neighbour Access Trees (GNATs) [7]. The methods cited have been developed by researchers working in the fields of Algorithmics, Data Structures, Computational Geometry, Pattern Recognition, etc., and are oriented towards main memory storage of the data structure, but a large number of disk-oriented data structures have also been devised in the field of Spatial Databases, Geometric Queries in Databases, Image Search and Retrieval, etc., among them the K-D-B-Trees [25], the R-trees [14], R\*-trees [2], X-trees [5], etc. Unfortunately, methods coming from these not-so-distant fields have not been compared or combined often.

A problem closely related to the search in a set of prototypes of the nearest neighbour of a point is the search of a subset with the  $k$  nearest neighbours, for a given constant  $k$ . In classification applications, the  $k$  Nearest Neighbours Rule is a classical method which offers consistently good results, ease of use and certain theoretical properties related to the expected error. The extension of most of the referenced algorithms to this variation of the problem is simple, with a cost equivalent or inferior to  $k$  times the cost of the original algorithm.

### 3 Approximate nearest neighbour search

In many cases, an absolute guarantee of finding the real nearest neighbour of the test point is not necessary. In this sense, a number of algorithms of approximate nearest neighbour search have been proposed. These methods can also be regarded as sub-optimal algorithms for the original problem of exact nearest neighbour search [6], [22], [1], [23], [17], [20].

But, why seek a suboptimal solution if so many sub-linear exact nearest neighbour search algorithms exist? The answer to this question comes from practical issues. For instance, the average costs of the most popular exact algorithms based on search trees are analysed in [27]. According to this author, it is not correct to assume that these algorithms achieve logarithmic or lower average costs in many practical cases. That assumption is valid for low dimensionalities, but when the number of components of the points gets larger, the number of points necessary to keep the average cost in the same terms is often extremely big.

In this work, two different approximate nearest neighbour algorithms have been tested. The first one is based on the classical *kd*-tree method. In a *kd*-tree, the search of the nearest neighbour of a test point is performed starting from the root, which represents the whole space, and choosing at each node the sub-tree that represents the region of the space containing the test point. When a leaf is reached, an exhaustive search of the  $b$  prototypes residing in the associated region is performed. Unfortunately, the process is not complete at this point. In that case, the cost involved in the search would be logarithmic with the number

of points and the technique would be definitive and extremely useful. However, as noted before, it is perfectly possible that among the regions defined by the initial partition, the one containing the test point be not the one containing the nearest prototype. It is easy to determine if this can happen in a given configuration, in which case the algorithm backtracks as many times as necessary until it is sure to have checked all the regions that can hold a prototype nearer to the test point than the nearest one in the original region. The resulting procedure can be seen as a Branch-and-Bound algorithm.

If a guaranteed exact solution is not needed, the backtracking process can be aborted as soon as a certain criterion is met by the current best solution. In [1], the concept of  $(1 + \epsilon)$ -approximate nearest neighbour query is introduced, along with a new data structure, the BBD-tree. A point  $p$  is a  $(1 + \epsilon)$ -approximate nearest neighbour of  $q$  if the distance from  $p$  to  $q$  is less than  $1 + \epsilon$  times the distance from  $p$  to its nearest neighbour. One of the splitting rules proposed for the BBD-tree, and the algorithm used to perform the  $(1 + \epsilon)$ -approximate nearest neighbour queries, based on a priority search scheme, have been used on conventional  $kd$ -trees in the experiments, ran using the implementation provided by D.M. Mount and S. Arya, available from <http://www.cs.umd.edu/~mount/ANN>. The parameters used were the ones by default, i.e. sliding midpoint splitting and bucket size of 1 point.

The second approximate nearest neighbour search algorithm tested is based on The Extended General Spacefilling Curves Heuristic [24]. The method works by mapping several times each prototype (an  $n$ -dimensional point) into the one dimensional Real Line through the application of a Spacefilling Mapping. Each mapping is preceded by a different set of rotations, normalization and transformations. The unidimensional values that correspond to each mapping (sub-model) are sorted and stored into a vector, inserted into a  $b$ -tree or inserted into an indexed table. When a test point  $p$  is presented to the system, it is mapped again into the Real Line with a different transformation for each of the  $r$  submodels. The  $b$  nearest neighbors of the unidimensional value in the Real Line, for each sub-model, can be readily found using a conventional search in the  $b$ -tree, in  $O(\log N + b)$ . The union of the  $r$  sets of  $b$  neighbours obtained from the  $r$  submodels produces a set (of size  $\leq rb$ , typically much lower than that upper bound) which will be exhaustively searched to find the nearest neighbour of  $p$  in the original multidimensional space. The constant  $b$  can be considered loosely related (and intentionally homonym) to the number of prototypes that is assigned to each leaf node in the models where a  $kd$ -tree does not partition the space completely.

## 4 Databases used

The well-known NIST Special Databases 3 and 7 have been used in all the experiments. These databases of isolated handwritten characters, composed of lower-case, upper-case letters, and digits, were used as training and test sets in the First Census Optical Character Recognition Systems Conference, sponsored

in 1992 by the American National Institute of Standards and Technology (NIST). The purpose of that event was to determine the state of the art in off-line handwritten character recognition, and twenty nine participants from North America and Europe took part in it. A training database (SD3) consisting of 223,125 handwritten digits, 45,313 lower case letters and 44,951 upper-case letters, with 128x128 binary pixels images, segmented and labelled by hand, was delivered. Twenty days later, a test database (TD1, known today as SD7) composed of 59,000 digits, 24,000 lower case letters and 24,000 upper case letters was sent to the participants, who had to return the results in the next 15 days.

The conference participants and many researchers thereafter have used SD3 and SD7 to test character recognition methods. An important conclusion obtained from that experience is that SD7 is significantly harder than SD3 for digits and at least very different, if not harder, for upper and lower case letters. In fact, SD3 is often split into a training and a test set, and SD7 is taken as a second test set. In those cases, SD7 is sometimes referred to as “hard test” since its error rates are considerably larger. The reasons given for that behaviour are related to the different ways in which SD3 and SD7 were obtained.

Although both databases were acquired by segmenting the characters filled out in boxes on forms, the forms for SD3 were completed by 2100 permanent Census field workers, who were probably very motivated and conscious of the importance of legible writing in the processing of large amounts of forms. SD7, on the other hand, was acquired from 500 high school students who were forced to fill out the forms in class. Additionally, SD7 forms were segmented by a different person than SD3 forms.

Some of the participants in the conference used exclusively SD3 to train, but others used proprietary databases. Among the first, the best recognition results at zero-rejection-rate were 96.84% for digits, 96.26% for upper-case letters, and 87.26% for lower-case letters. The methods used in the best system for digits and in most of the best placed systems were based in the  $k$ -nearest neighbours rule.

In [13], SD3 was used to compare several classifiers on handwritten digits, namely Multilayer Perceptrons, Radial Basis Function Networks, Gaussian Parametric Classifiers, and  $k$ -Nearest Neighbours methods. 7480 digits randomly selected among the first 250 writers were used as training and 23140 digits extracted from the second 250 writers constituted the test set. The best results were again from the  $k$ -nearest neighbours methods. In [15], a perturbation method using neural networks is proposed, achieving an excellent 97.1% result on the SD7 digits hard test. The idea of perturbing (distorting) the characters has been also used in the experiments of this work.

## 5 Experiments

The preprocessing and feature extraction methods employed were very simple. In the first place, the character images were sub-sampled from their original 128x128 binary pixels into 14x14 gray value representations by first computing

the minimum inclusion box of each character, keeping the original aspect ratio, and then accumulating into each of the 14x14 divisions the area occupied by black pixels to obtain a continuous value between 0 and 1. Principal Components Analysis (or Karhunen-Loève Transform) was then performed on the image representations to reduce its dimensionality to 45. The reduced resolution and final dimensionality values were chosen as a good compromise for all the types of characters tested after extensive experimentation.

Usual slant normalization techniques were not found to improve the results, but the insertion of artificially slanted images to the training set produced significant improvements. The slant was applied to the original binary images and consisted on right or left-shifting each row an integer number of positions. The central row was never shifted, and the amount of shift increased linearly from there to the top and bottom rows, respectively. The new pixels entering the area after a shift were set to white. Morphological erosions and dilations on the original images were also introduced in the database as a preliminary test to find out if adding distorted versions of the training data can be useful. The results of these tests are presented later in this section.

A first set of experiments were performed using the first 200,193 digits from SD3 as a training set and the remaining 22,903 digits as test set (no writer was split by this setting). The results for the Spacefilling Curves model (SPFC) with one of the best combinations of  $r$  (number of submodels) and  $b$  (number of neighbours on the Real Line), namely  $r=15$  and  $b=80$ , are shown in Table 1. With the same training and test sets, the  $kd$ -tree model performed better for a range of values of  $\epsilon$ , as can also be seen in Table 1. These results are at zero rejection rate, for a number of neighbours  $k=4$ , which gave the best results in all the tests. In our experience, the SPFC method outperforms  $kd$ -trees only when the data points follow uniform distributions, which is not the case for most real pattern recognition tasks.

**Table 1.** Results of handwritten digit recognition with a partition of the NIST SD3 database for training and test. CPU times in a Pentium II - 450Mhz machine are shown.

Method and Setting	Recog. Rate (%)	Search Time (ms/char)
SPFC	99.09	6.44
$kd$ -tree $\epsilon = 0.5$	99.21	12.66
$kd$ -tree $\epsilon = 1.5$	99.21	2.40
$kd$ -tree $\epsilon = 3.0$	99.10	0.65

According to these first results, the rest of the experiments were focused on the  $kd$ -tree models. Similar tests with the first 38,678 upper-case letters from SD3 as training and the remaining 6,273 as test were performed, along with tests using the first 38976 lower-case letters for training and the remaining 6,337

for test. The results of these experiments are summarized in Table 2. In this case, the best numbers of neighbours were  $k=4$  and  $k=6$  respectively.

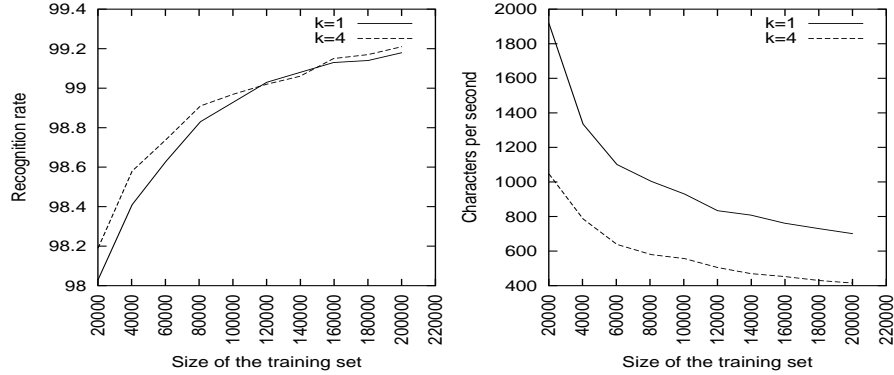
**Table 2.** Results of upper and lower-case letters recognition with partitions of the NIST SD3 database for training and test. CPU times are in ms/character in a Pentium II - 450Mhz machine.

Upper-case letters ( $k=4$ )			Lower-case letters ( $k=6$ )		
$\epsilon$	Recog. Rate (%)	Search Time	$\epsilon$	Recog. Rate (%)	Search Time
0.0	94.64	20.99	0.0	89.81	20.64
0.5	94.64	8.05	0.5	89.81	8.12
1.5	94.64	1.87	1.5	89.74	1.86
3.0	94.28	0.57	3.0	89.49	0.54

To obtain a quantitative indication of the improvements that can be expected from training sets of increasing size, a series of experiments was conducted with training sets from 20,089 to 200,193 digits. The results and recognition speeds are shown in Figure 1. The throughputs are measured on a Pentium II - 450 Mhz machine running UNIX (Linux 2.2.9) and do not include the preprocessing time of the test character.

Given the slow increase of the search times incurred when the database grows, an interesting approach to improve the accuracy, keeping at the same time high recognition speeds, is to insert new prototypes into the training set. Of course, making larger the original database is an evident way to do it, but the manual or semi-automated segmentation and labeling procedures needed to build a good, large database are very time-consuming. Therefore, a possible approach to exploit the information of a given database as much as possible is to perform controlled deformations on the characters to insert them into a new larger training set. Similar approaches based on deformations of the data have been proposed in [15] with excellent results. Here, we propose as a faster and cleaner option to include the distorted characters in the training set instead of distorting the test character in several ways and carrying out the classification of each deformed pattern. We have tested this scheme using slanted versions of the original characters, as explained at the beginning of this section. The recognition rate using 4 slant angles to obtain a training set of 1,000,965 characters (including the 200,193 original ones) improved to 99.43%, from 99.21%, thus cutting the error rate by more than one fourth, in the test on SD3 digits, with  $k=4$  and  $\epsilon = 1.5$ . The search time increased from 2.4 ms/char. to 4.5 ms/char.

All the results presented have been obtained using only SD3. The same experiments have been also performed on SD7, taking the whole SD3 as training set. In Table 3, the results for both databases are summarized. A last experiment was conducted to test if other kinds of deformations could increase the recognition rates. Two additional sets of images obtained by eroding and dilating by



**Fig. 1.** Recognition rate at zero percent rejection (left), and number of searches per second (right) for two different values of  $k$ , and increasing training set sizes.

**Table 3.** Summary of recognition rates of digits and upper-case letters, with partitions of the NIST SD3 database (“easy test”), and with the whole SD3 for training and SD7 for test (“hard test”). In some experiments the training set has been augmented with 4 slanted versions of each character. The parameters used were  $k=4$  and  $\epsilon = 1.5$ .

Training set / Test set	Rec. rate digits (%)	Rec. rate uppers. (%)
SD3 / SD3 (easy test)	99.21	94.64
SD3+slanted / SD3 (easy test)	99.43	95.78
SD3 / SD7 (hard test)	95.16	89.43
SD3+slanted / SD7 (hard test)	96.28	92.34

one pixel the original (128x128) digit images were appended to the training set (which was then  $4+2=6$  times larger than the original one). The results for the “easy test” did not improve, and in the “hard test” the recognition rate increased by 0.31%, reaching 96.59%.

## 6 Conclusions

The experimental results obtained suggest that fast approximate  $k$ -nearest neighbours search can be a practical approach to handwritten optical character recognition. Previous results [26] indicating that the error rate is more than halved each time the database size is increased tenfold have been confirmed, and preliminary work on the idea of inserting distorted characters into the database has been shown to improve significantly the accuracy with moderate increases of the search times.

Obviously, many potentially useful distortions are possible, and there is a practical limit on the number of prototypes in the database. Therefore a method to reduce its size without compromising the results should be found. Condensing methods are clear candidates (see section 1), and a simple way to reduce the number of distorted characters entering the database could be to test each one before inserting it, and discard points that do not convey discrimination power (those with  $k'$  neighbours of the same class, for example). Keeping all the original prototypes, and inserting only artificial characters meeting a certain criterion seems a safe and efficient tradeoff, which we plan to test in the immediate future.

## References

1. S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.
2. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The  $r^*$ -tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD Conf. on the Management of Data 90, Atlantic City.*, May 1990.
3. J.S. Beis and D.G. Lowe. Indexing without invariants in 3d object recognition. *IEEE Trans. on PAMI*, 21(10):1000–1015, October 1999.
4. J.L. Bentley, B.W. Weide, and A.C. Yao. Optimal expected time algorithms for closest point problems. *ACM Trans. on Math. Software*, 6:563–580, 1980.
5. S. Berchtold, D.A. Keim, and H.P. Kriegel. The  $x$ -tree: An index structure for high-dimensional data. In *Proc. 22nd Very Large Database Conference, Bombay, India*, pages 28–39, 1996.
6. M. Bern. Approximate closest-point queries in high dimensions. *Pattern Recognition*, 45:95–99, 1993.
7. S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Inter. Conf. on Very Large Data Bases*, pages 574–584, 1995.
8. P. A. Devijver and J. Kittler. On the edited nearest neighbour rule. pages 72–80. Proceedings of the 5th International Conference on Pattern Recognition, IEEE Computer Society Press, Los Alamitos, CA, 1980.

9. L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, 1996.
10. J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm finding best matches in logarithmic expected time. *ACM Trans. Math. Software*, 3:209–226, 1977.
11. K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing  $k$ -nearest neighbors. 24:750–753, 1975.
12. S. Geva and J. Sitte. Adaptive nearest neighbor pattern classification. *IEEE Trans on Neural Networks*, 2(2):318–322, 1991.
13. P.J. Grother and G.T. Candela. Comparison of handprinted digit classifiers. In *NISTIR*, 1993.
14. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *UCB, Elec.Res.Lab, Res.R. No.M83-64, with Stonebraker, M.*, 1983.
15. T.M. Ha and H. Bunke. Off-line, handwritten numeral recognition by perturbation method. *IEEE Trans. on PAMI*, 19(5):535–539, May 1997.
16. P.E. Hart. The condensed nearest neighbor rule. *IEEE Trans. on Information Theory*, 125:515–516, 1968.
17. R. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th Symposium on Theory of Computing*, 1998.
18. B. S. Kim and S. B. Park. A fast  $k$  nearest neighbor finding algorithm based on the ordered partition. *IEEE Trans. on PAMI*, 8:761–766, 1986.
19. T. Kohonen. *Self Organization and Associative Memory*. Springer-Verlag, 1988.
20. E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *30th Symposium on Theory of Computing*, 1998.
21. G. Loizou and S. J. Maybank. The nearest neighbor and the Bayes error rates. *IEEE Trans. on PAMI*, 9:254–262, 1987.
22. L. Miclet and M. Dabouz. Approximative fast nearest neighbor recognition. *Pattern Recognition Letters*, 1:277–285, 1983.
23. J.C. Perez and E. Vidal. An approximate nearest neighbours search algorithm based on the extended general spacefilling curves heuristic. In *Workshop on Statistical Pattern Recognition SPR-98, Sydney, Australia.*, 1998.
24. J.C. Perez and E. Vidal. The extended general spacefilling curves heuristic. In *Intl. Conf. on Pattern Recognition ICPR-98, Brisbane, Australia.*, 1998.
25. H. Robinson. *Database Analysis and Design*. , 1981.
26. S.J. Smith, Sims K. Bourgojn, M.O., and H.L. Voorhees. Handwritten character classification using nearest neighbor in large databases. *IEEE Trans. on PAMI*, 16(9):915–919, September 1994.
27. R.F. Sproull. Refinements to nearest-neighbor searching in  $k$ -dimensional trees. *Algorithmica*, 6:579–589, 1991.
28. C. Tomasi and R. Manduchi. Stereo matching as a nearest-neighbor problem. *IEEE Trans. on PAMI*, 20(3):333–340, March 1998.
29. D.L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Trans. on Systems, Man and Cybernetics*, 2:408–420, 1972.
30. P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *4th ACM Symp. on Discrete Algorithms*, pages 311–321, 1993.
31. P. Zakarauskas and J.M. Ozard. Complexity analysis for partitioning nearest-neighbor searching algorithms. *IEEE Trans. on PAMI*, 18(6):663–668, June 1996.