Permutation-Based Pruning for Approximate K-NN Search

Hisham Mohamed and Stéphane Marchand-Maillet

Université de Genève, Geneva, Switzerland
{hisham.mohamed,stephane.marchand-maillet}@unige.ch

Abstract. In this paper, we propose an effective indexing and search algorithms for approximate K-NN based on an enhanced implementation of the Metric Suffix Array and Permutation-Based Indexing. Our main contribution is to propose a sound scalable strategy to prune objects based on the location of the reference objects in the query ordered lists. We study the performance and efficiency of our algorithms on large-scale dataset of millions of documents. Experimental results show a decrease of computational time while preserving the quality of the results.

Keywords: Metric Suffix Array (MSA), Permutation-Based Indexing, Approximate Similarity Search, Large-Scale Multimedia Indexing.

1 Introduction

Searching for similar objects in a database is a fundamental problem for many applications, such as information retrieval, visualization, machine learning and data mining.

In metric spaces [1], several techniques have been developed for improving the performance of searching, by decreasing the number of direct distance calculations [1]. One of the recent techniques is the *permutation-based indexing* [2,3]. The idea behind it is to represent each object by a list of permutations of selected neighboring items (reference points). The similarity between any two objects is then derived by comparing the two corresponding permutation lists. In this work, we propose an enhanced implementation of the Metric Suffix Array (MSA) proposed in [4] which is one of the recent data structure for permutation based-indexing. In [4], regardless of the number of permutations, the number of objects, and the number of K-NN which need to be retrieved, the complete MSA has to be scanned in order to retrieve the most promising results. Here, we propose an enhanced implementation of the MSA to avoid scanning the complete MSA. The main idea is to prune cells representing objects that have a high difference in their permutations ordering. Hence, only a small part of the array is scanned, which improves the running time. In addition, we propose different strategies for selecting the reference points. To validate our claims, we test the enhanced MSA and the selection strategies on a high dimensional large dataset containing several millions of objects.

The rest of the paper is organized as follows. Section 2 proposes a review of the related work. In sections 3 and 4, we introduce our modeling for permutation-based indexing and a formal justification for our proposed indexing and searching procedures. Finally, we present our results in section 5 and conclude in section 6.

2 Prior Work

The idea of *permutation-based indexing* was first proposed in [3,2]. Amato and Savino [3] introduced the metric inverted files (MIF) to store the permutations in an inverted file. Then, Mohamed and Marchand-Maillet [5] proposed three distributed implementation of the MIF. In [6], authors proposed the *brief permutation index*. The main idea is to encode the permutation as a binary vector and to compare these vectors using the Hamming distance. In [7], authors proposed the *prefix permutation index* (PP-Index). PP-Index stores the prefix of the permutations only and the similarity between objects is measured based on the length of its shared prefix. Furthermore, in [4] authors proposed the MSA, which is a fast and an effective data structure for storing the permutations, by saving half of the processing memory which is needed in [3,5,7]. The work presented in this paper is based on [4] considered as current state of the art. We provide an enhanced implementation of the MSA for fast and effective retrieval and we test it against [4].

3 Indexing Model

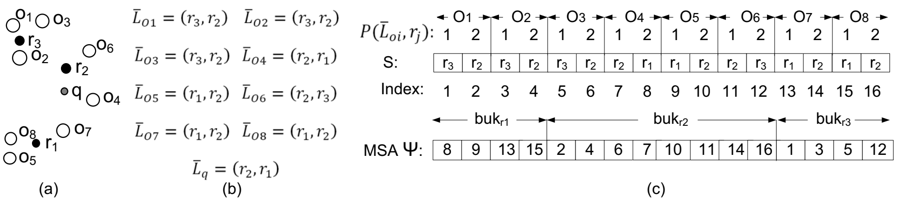
Permutation-based indexes aim to predict the proximity between elements according to how they order their distances towards a set of reference objects [2,3].

Definition 1. *Given a set of N objects o_i, $D = \{o_1, \ldots, o_N\}$, a set of reference objects $R = \{r_1, \ldots, r_n\} \subset D$, and a distance function which follows the metric space postulates, we define the* ordered list *of R relative to $o \in D$, L_o, as the ordering of elements in R with respect to their increasing distance from o:*

$$L_o = \{r_{i_1}, \ldots, r_{i_n}\} \text{ such that } d(o, r_{i_j}) \leq d(o, r_{i_{j+1}}) \ \forall j = 1, \ldots, n-1$$

Then, for any $r \in R$, $P(L_o, r)$ indicates the position of r in L_o. In other words, $P(L_o, r) = j$ such that $r_{i_j} = r$. Further, given $\bar{n} > 0$, $\bar{L}_o$ is the pruned ordered list *of the $\bar{n}$ first elements of L_o.*

Figure 1(b) gives the pruned ordered lists $\bar{L}_{o_i}$, where $\bar{n} = 2$, for D and R illustrated in Figure 1(a).



o1 o3 r3 o2 o6 r2 q o4 o8 o7 r1 o5
(a)
$\bar{L}_{o1} = (r_3, r_2)$ $\bar{L}_{o2} = (r_3, r_2)$
$\bar{L}_{o3} = (r_3, r_2)$ $\bar{L}_{o4} = (r_2, r_1)$
$\bar{L}_{o5} = (r_1, r_2)$ $\bar{L}_{o6} = (r_2, r_3)$
$\bar{L}_{o7} = (r_1, r_2)$ $\bar{L}_{o8} = (r_1, r_2)$
$\bar{L}_q = (r_2, r_1)$
(b)
O1 O2 O3 O4 O5 O6 O7 O8
$P(\bar{L}_{oi}, r_j)$: 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
S: r3 r2 r3 r2 r3 r2 r2 r1 r1 r2 r2 r3 r1 r2 r1 r2
Index: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
buk$_{r1}$ buk$_{r2}$ buk$_{r3}$
MSA Ψ: 8 9 13 15 2 4 6 7 10 11 14 16 1 3 5 12
(c)

Fig. 1. a) White circles are data objects o_i; black circles are reference objects r_j; the gray circle is the query object q b) Pruned ordered lists $\bar{L}_{o_i}$, $\bar{n} = 2$. c) Example of Metric Suffix Array and buckets $b_j = \mathsf{buk}_{r_j}$

In K-NN similarity queries, we are interested in ranking objects (to extract the K first elements) and not so much in the actual inter-object distance values. Permutation-based indexing relaxes distance calculations by assuming that they will be approximated in terms of their ordering when comparing the ordered lists of objects. Here, we consider the Spearman Footrule Distance (d_{SFD}) between ordered lists. Formally,

$$d(q, o_i) \overset{\text{rank}}{\simeq} d_{\text{SFD}}(q, o_i) = \sum_{\substack{r_j \in \bar{L}_q \\ r_j \in \bar{L}_{o_i}}} |P(\bar{L}_q, r_j) - P(\bar{L}_{o_i}, r_j)| \tag{1}$$

To efficiently answer users queries, the Metric Suffix Array (MSA) was proposed in [4]. Independent of the number of reference points, objects or K-NN, the complete MSA should be scanned in order to retrieve the most similar objects to the query. Here, we use our rank-based approximation to postulate that objects having similar ordered lists as the query (based on d_{SFD}) are candidates similar objects, which we could filter by direct distance calculation. Hence, as per Eq.(1), for a query q, if we organise the MSA cells to characterise objects such that $P(\bar{L}_q, r_j) = P(\bar{L}_{o_i}, r_j)$, we can avoid accessing the complete MSA to obtain the list C of candidate similar objects. We therefore propose below an enhanced structure for the MSA to help reducing the searching time. We first recall formally the construction of the MSA [4] .

Given all pruned ordered lists $\bar{L}_{o_i}$, we construct $S = \bigcup_{i=1}^{N} \bar{L}_{o_i} = \{r_{i_1}, \ldots, r_{i_M}\}$, where $M = \bar{n}.N$. The set S can then be seen as a string of length M on the alphabet R. A Metric Suffix Array Ψ acts like a Suffix Array [8,9,10] over S. More specifically, Ψ is a set of M integers corresponding to the permutation induced by the lexical ordering of all M suffixes in S ($\{r_{i_k}, \ldots, r_{i_M}\}$ $\forall k \leq M$). In [4], the MSA is sorted into buckets.

Definition 2. *A* bucket *for reference point r_j is a subset of the* MSA *Ψ from position b_j. The bucket is identified to its position in Ψ so that $b_j \equiv \Psi_{[b_j, b_{j+1}-1]}$.*

Bucket b_j contains the positions of all the suffixes of S of the form $\{r_j, \ldots, r_{i_M}\}$, i.e. where reference point r_j appears first.

For example, in Figure 1(c), the string S corresponding to the objects and reference points shown in Figure 1(a) is given. The MSA Ψ is also shown. The bucket b_2 for reference point r_2 (buk_{r_2}) contains the positions in S of suffixes starting by r_2.

At query time, $\bar{L}_q$ will be computed. From Eq.(1), we therefore need to characterise the objects

$$\{o_i \text{ s.t } r_j \in \bar{L}_{o_i}, \forall r_j \in \bar{L}_q \text{ } and \text{ } P(\bar{L}_{o_i}, r_j) = P(\bar{L}_q, r_j)\}$$

The MSA along with buckets encodes enough information to recover the relationships between an object o_i and a given reference point r_j. Given $r_j \in \bar{L}_q$, we scan Ψ at positions $k \in [b_j, b_{j+1} - 1]$ and determine i, and $P(\bar{L}_{o_i}, r_j)$ from Ψ_k as follows:

$$i = \left\lfloor \frac{\Psi_k}{\bar{n}} \right\rfloor + 1 \qquad P(\bar{L}_{o_i}, r_j) = (\Psi_k \bmod \bar{n}) + 1 \tag{2}$$

Within each bucket b_j, $P(\bar{L}_{o_i}, r_j) \leq \bar{n}$. In order to speed up further the scanning of buckets, we sort them according to the value of $P(\bar{L}_{o_i}, r_j)$. A sub-bucket $b_j^{(l)}$ points to objects o_i such that $P(\bar{L}_{o_i}, r_j) = l$ (see Figure 2(b)).

4 Practical Setup

4.1 Indexing

Algorithm 1 details the indexing process. Line 1 builds the MSA using pruned lists $\bar{L}_{o_i}$. In lines 2-3, buckets are sorted based on $P(\bar{L}_{o_i}, r_j)$ (Eq.(2)) using the *quicksort* algorithm. All the suffixes representing the objects that have the same $P(\bar{L}_{o_i}, r_j)$ are located next to each other, which makes it easy to divide each bucket into $\bar{n}$ sub-buckets $b_j^{(l)}$ (Line 4).

Algorithm 1

```
IN: Domain $D$ of $N$ , $R$ of $n$ ,$\bar{n}$
OUT: The MSA $\Psi$ with sub-buckets $b_j^{(l)}$
1.  Build the MSA $\Psi$ and buckets $b_j$
2.  For each $r_j \in R$
3.      quickSort($\Psi, b_j, b_{j+1} - 1$)
4.      Define the sub-buckets $b_j^{(l)}$ within each bucket $b_j$
```

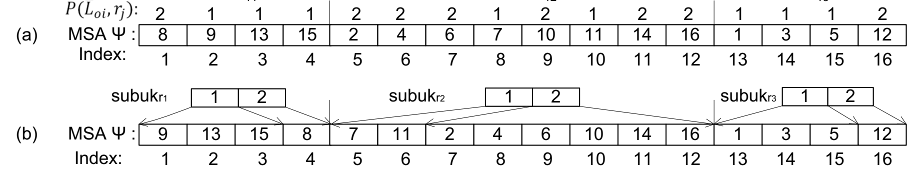
Theoretically, the indexing complexity is $O(\bar{n}\eta(1 + log\eta))$, where η is the average size of the sub-buckets ($\eta \leq N$). The average memory usage is $O(M + (\bar{n} \times n))$.

4.2 Searching

Equation (1) measures the discrepancy in ranking from common reference objects between each object and the query. In practice, it can be simplified by counting the co-occurrences of each object with the query in the same (or adjacent) sub-buckets. That is, each object o_i scores

$$s_i = \left|\left\{r_j \in \bar{L}_q \text{ such that } (r_j \in \bar{L}_{o_i} \text{ and } |P(\bar{L}_{o_i}, r_j) - P(\bar{L}_q, r_j)) \leq 1|\right\}\right| \quad (3)$$

Objects o_i are then sorted according to their decreasing s_i scores. This sorted candidate list provides an approximate ranking of the database objects relative to the submitted query. This approximate ranking can be improved by direct distance calculation (DDC). For a K-NN query, we apply DDC on the $K_c = \Delta \times K$ first objects in our sorted candidate list and call $\Delta > 1$ the *DDC factor*. The effect of Δ is explored in our performance evaluation (section 5). The search procedure is described in Algorithm 2.



buk_r1
buk_r2
buk_r3
P(L̄oi, rj): 2 1 1 1 2 2 2 1 2 1 2 2 1 1 1 2
(a) MSA Ψ : 8 9 13 15 2 4 6 7 10 11 14 16 1 3 5 12
Index: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
subuk_r1 1 2
subuk_r2 1 2
subuk_r3 1 2
(b) MSA Ψ : 9 13 15 8 7 11 2 4 6 10 14 16 1 3 5 12
Index: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Fig. 2. a) MSA sorted by bucket b) MSA sorted by sub-buckets

First $\bar{L}_q$ is computed (line 1). For each $r_j \in \bar{L}_q$, active sub-buckets are identified (line 3). For each object o_i pointed, a count $s[i]$ of co-occurrences is computed using (Eq.3) (lines 4-5). Line 6 sorts the candidate list in decreasing order of their score (counters s). Final DDC filtering is performed in lines 7-9.

Algorithm 2

```
IN: Query: $q$, $R$ of $n$, MSA, $b_j^{(l)}$, $s[0 \dots N]$
OUT: Sorted Objects list: out
1.  Create the query ordered list $\bar{L}_q$
2.  For $r_j \in \bar{L}_q$, $l = P(\bar{L}_q, r_j)$
3.      For $k = b_j^{(l)}$ to $b_j^{(l+1)} - 1$
4.          $i = \left\lfloor \frac{\Psi_k}{\bar{n}} \right\rfloor + 1$
5.          $s[i] = s[i] + 1$
6.  sort(s)
7.  $K_c = K_{NN} \times \Delta$
8.  $C \leftarrow s[K_c]$
9.  $out = \text{calc\_distance}(C, K_c, q)$
10. $sort(out)$
```

Theoretically, the computational complexity to retrieve the K_c is $O(2\bar{n}\eta)$.

4.3 Reference Points Selection

In [3,7], the selection of R is done randomly, we name this strategy *Random Selection* (RS). We propose three alternative strategies for selecting the reference points. The first strategy is the *distributed selection (DS)*. In *DS*, close reference points are neglected based on a certain threshold value. Hence, if one of the new selected points is close to an already selected point, the selection is ignored. Using this technique, we ensure that the points are well-distributed over the database. That leads to a relevant encoding of each object using the permutations. The second and the third strategies are based on the k-mean algorithm for clustering. We call them *post-clustering selection (PCS)* and *post-clustering distributed selection (PCDS)*. The main idea behind the two strategies is that the dataset is divided into a number of clusters to support the selection of the reference points. For instance, if we need 1,000 reference points and we create 5 clusters, 200 reference points are selected from each cluster. We thus ensure that the objects located in the same cluster have the same reference points as the primary items in their order lists. This helps to improve the identification of the objects for eliminating unwanted regions. The main difference between *PCS* and *PCDS* is that, *PCDS*, applies the *DS* strategy inside each cluster. We ensure that even within each cluster the reference points are not too close one to another. In section 5, we empirically compare the four strategies *(RS, DS, PCS, PCDS)* and use the best strategy for the rest of our experiments.

5 Experimental Results

The average recall (RE), average position error (PE) [1], and average indexing and running time are measured and compared with that listed in [4]. All the experiments were done using 250 different queries that were selected randomly from the dataset. The sub-bucket algorithm was implemented in C++. The experiments were done on a 2.70GHz processor, with 128Gb of memory and 512GB storage capacity. Our conducted experiments are based on a dataset of 5 millions visual shape features (21-dimensional), which were extracted from the 12-million ImageNet corpus [11].

Selecting the reference points (R): Figures 3a and 3b show the average *RE* and *PE* for the reference point selecting strategies (section 4.3) for 10 K-NN using MSA-Full [4].

The *RS* technique gives the lowest RE and the highest PE values. Using RS, there is a high probability that some reference points are close one to another. That makes them at the same distance from the objects, leading to inefficient encoding. *DS* gives a higher RE and lower PE compared to the *RS*, as the objects are identified using equally distributed reference points. For the *PCS* and the *PCDS* techniques, the dataset is clustered into 5 and 10 clusters. From the figures, we see that the RE and PE augment with the number of clusters. This is based on the dataset and the number of clusters that we can get out of the dataset. Also, we can see that the *PCDS* technique gives better RE and PE values than the *PCS*. The reason is that the reference points which were selected from each cluster are well-spread within the clusters.

Comparing the four strategies, when the number of reference points increases the RE increases and PE decreases until a certain limit. The *DS* and the *PCDS* give the best RE and PE because these techniques ensure a good distribution of the reference points. On the other hand, *DS* is better than the *PCDS* in terms of time consumption. There is no time used to cluster the data before selecting the reference points. We therefore apply *DS* in the rest of our experiments.

Sub-buckets and DDC Factor: Figure 4a shows the average RE and PE for different numbers of reference points using MSA-Full, MSA-NN [4] and the sub-bucket implementations ($\Delta = 100$) for 10 K-NN. From the figure, for the three algorithms, when the number of reference points increases, RE increases and PE decreases. For the sub-bucket algorithm, even with small number of reference points, we are able to achieve higher RE and lower PE than scanning the complete MSA using high number of

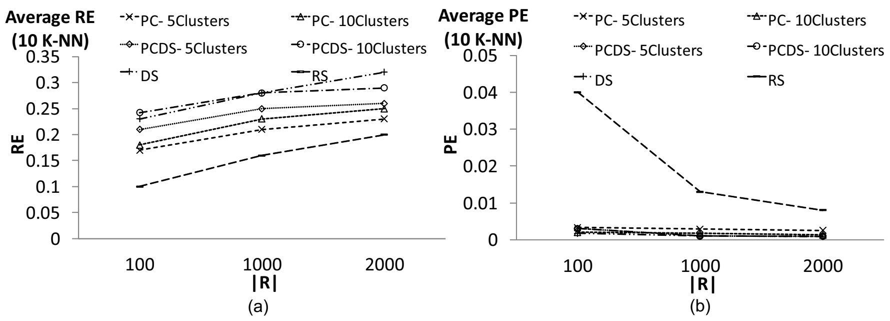


Fig. 3. Average *RE*(a) and *PE*(b) (250 queries) for top 10-NN using MSA-Full [4]

reference points for MSA-Full and MSA-NN. For instance, using $n = 100$ and $\Delta = 100$, for 10 K-NN, our algorithm is able to achieve higher RE than using $n = 2000$ for MSA-Full and MSA-NN, with a faster response time (next paragraph).

Figure 4b shows the RE and PE for different numbers of reference points using different values of Δ for 10 K-NN. As a reminder, for 10 K-NN and $\Delta = 40$, the algorithm calculates the distance between the query and the top $K_c = 400$ candidates objects. As we can see, when Δ increases, RE increases and PE decreases, as the number of the objects that are compared to the query increases.

Comparing MSA-Full, MSA-NN and MSA-sub buckets Algorithms: Table 1 shows the average indexing and searching time (in seconds) for the sub-bucket implementation (including scanning the MSA and accessing the hard-disk) compared to MSA-Full and MSA-NN proposed in [4]. For indexing, we can see that the indexing time for the three algorithms increases with the increase in the number of reference points.

Table 1. Indexing and searching time(in seconds) for sub-bucket (Δ=100) compared to [4]

$\|R\|$	**Index-Full**	**Search-Full**	**Index-NN**	**Search-NN**	**Index-Subbuckets**	**Search-Subbucket**
100	45	4	86	1.5	113	**0.35**
1000	517	46	735	12	1622	**0.45**
2000	1081	94	1651	25	3523	**0.76**

In addition, the indexing time for the sub-buckets algorithm is higher than that of the other algorithms. This is due to the sorting and definition of the sub-buckets after building the MSA. However, since the indexing process is an off-line process, this increase is accepted.

For searching, for different n, using $\Delta = 100$, it appears clearly from Table 1 that the sub-bucket technique is faster than [4], as the algorithm does not need to scan all the MSA cells nor all the database.

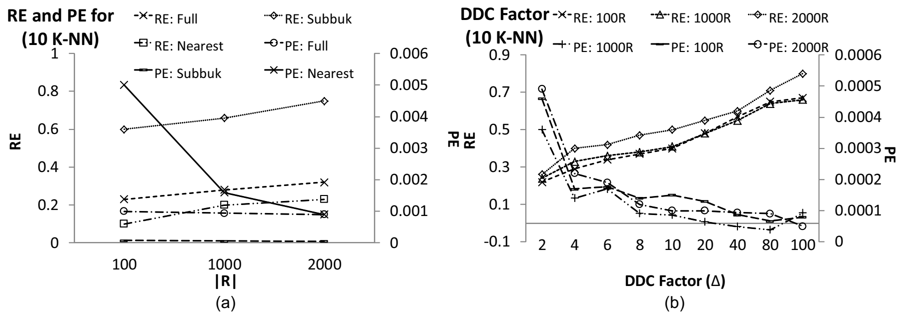


Fig. 4. Using 250 queries a) Comparing sub-bucket to [4]. b) Effect of Δ.

6 Conclusion

We have presented an enhanced indexing technique based on the Metric Suffix Array (MSA), representing the current state of the art implementation for *permutation-based indexing*. Our main idea is to prune the MSA cells which represent objects that have high difference in their permutations ordering. Hence, only a small part of the array is scanned. With a combination of direct distance calculations, we showed through an experimental analyses that our algorithm gives better results in terms of time and precision compared to that proposed in [4]. In addition, we empirically showed how the selection of the reference points can affect the performance. There is much to improve on this selection for *permutation-based indexing*. This is the subject of our future work.

Acknowledgment. This work is jointly supported by the Swiss National Science Foundation (SNSF) via the Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (IM2) and the European COST Action on Multilingual and Multifaceted Interactive Information Access (MUMIA) via the Swiss State Secretariat for Education and Research (SER).

References

1. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems, vol. 32. Springer (2006)
2. Gonzalez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(9) (September 2008)
3. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: International Conference on Scalable Information Systems, pp. 28:1–28:10 (2008)
4. Mohamed, H., Marchand-Maillet, S.: Metric suffix array for large-scale similarity search. In: ACM WSDM 2013 Workshop on Large Scale and Distributed Systems for Information Retrieval, Rome, IT (February 2013)
5. Mohamed, H., Marchand-Maillet, S.: Parallel approaches to permutation-based indexing using inverted files. In: 5th International Conference on Similarity Search and Applications (SISAP), Toronto, CA (August 2012)
6. Téllez, E.S., Chávez, E., Camarena-Ibarrola, A.: A brief index for proximity searching. In: Bayro-Corrochano, E., Eklundh, J.-O. (eds.) CIARP 2009. LNCS, vol. 5856, pp. 529–536. Springer, Heidelberg (2009)
7. Esuli, A.: Pp-index: Using permutation prefixes for efficient and scalable approximate similarity search. In: Proceedings of LSDSIR 2009, vol. i, pp. 1–48 (July 2009)
8. Manber, U., Myers, E.W.: Suffix arrays: A new method for on-line string searches. SIAM J. Comput. 22(5), 935–948 (1993)
9. Schürmann, K.B., Stoye, J.: An incomplex algorithm for fast suffix array construction. Softw., Pract. Exper. 37(3), 309–329 (2007)
10. Mohamed, H., Abouelhoda, M.: Parallel suffix sorting based on bucket pointer refinement. In: 5th Cairo International Biomedical Engineering Conference (CIBEC), pp. 98–102 (2010)
11. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR 2009 (2009)