Efficient clustering and matching for object class recognition

Bastian Leibe
ETH Zurich
Zurich, Switzerland

Krystian Mikolajczyk
University of Surrey
Guildford, UK

Bernt Schiele
TU Darmstadt
Darmstadt, Germany

Abstract

In this paper we address the problem of building object class representations based on local features and fast matching in a large database. We propose an efficient algorithm for hierarchical agglomerative clustering. We examine different agglomerative and partitional clustering strategies and compare the quality of obtained clusters. Our combination of partitional-agglomerative clustering gives significant improvement in terms of efficiency while maintaining the same quality of clusters. We also propose a method for building data structures for fast matching in high dimensional feature spaces. These improvements allow to deal with large sets of training data typically used in recognition of multiple object classes.

1 Introduction

Many of todays models and approaches for object class recognition are based on local features. Local features are typically extracted from images and subsequently grouped into appearance clusters [1, 12, 23]. Besides reducing the size of the feature space, appearance clusters allow to capture a larger variability of local image structure than individual features, as well as to focus on parts which re-occur on many instances of the object class and consequently generalize over new instances. While appearance clusters seem to be an essential component of several successful approaches, they have been applied only to a relatively small number of object classes using small training and test sets. A typical feature detector might extract 10s-100s of features per image. Learning models for 100s of object classes using 10s or even 100s of images per class would imply that the approach has to deal with 100,000s to 1,000,000s of features during training. This number is but a conservative estimate, since we might want to scale to many more object classes or use far more images in the context of unsupervised learning or topic discovery [24]. It is however unclear if and how approaches based on appearance clusters can deal with such massive amounts of high-dimensional data.

In general, clustering is a powerful tool for finding structure in large data sets [10]. However, the question what is a good clustering method cannot be answered without the context of a task. Our main interest is the use of clustering for object class recognition using local-feature based approaches. When using appearance clusters for building object models we can differentiate three aspects: 1) *clustering* to obtain the appearance clusters, 2) *matching* during training and recognition, and 3) the *recognition* method. In this paper we focus on the first two aspects, namely *clustering* and *matching*.

In computer vision, frequently used clustering strategies are k-means [23, 26] and agglomerative clustering [1, 12, 17]. Other methods like Mean Shift [6] also become more and more popular. However, their performance has not been compared for computer vision tasks, and no guidelines are available for judging the tradeoffs in representational

capacity, accuracy, and run-time. K-means is frequently used because of its computational simplicity. However, the clustering solution is suboptimal when the number of outliers in the point distribution is large. Moreover, the solution depends on an arbitrarily set number of clusters and random initialization, which makes it less attractive for object categorization. In the agglomerative clustering scheme the number of clusters is automatically determined. However, both the runtime and memory requirements are often significantly higher for agglomerative methods. Given the large amounts of data that need to be processed, an efficient implementation of the clustering algorithm is therefore crucial for its applicability.

The second important aspect is to efficiently match features to appearance clusters. Numerous methods have been proposed for efficient search [3, 19]. In high-dimensional spaces, however, these methods are no longer effective. Many methods therefore use approximate nearest-neighbor search techniques [2, 9, 14]. In object recognition and categorization, however, we are interested in matches within a similarity distance to a feature point. This type of volume search is much harder to do efficiently. In contrast to k-means clustering, the result of agglomerative clustering can be used directly to obtain a data structure for efficient volume search, namely a ball-tree [20]. In experiments we observe speedup factors of 20- 200 for matching through the use of this technique.

In this paper we introduce several improvements to agglomerative clustering (Sec. 3), making it tractable for large data sets while preserving cluster quality. We use the clustering results to build a data structure for efficient volume search in high-dimensional spaces (Sec. 4). In the experiments we show significant speedup factors for matching and we also compare the performance of k-means and the agglomerative scheme, both in terms of computational cost and recognition performance (Sec. 5).

2 Recognition approach

We briefly describe two main stages of a recognition algorithm which are similar in many state-of-the art approaches [1, 12, 17]. While there exist differences between the individual approaches, the following describes an object class representation and a matching procedure which may be seen as a common basis of many approaches.

Object representation. In our approach clustering is used to build an appearance model in which each cluster is represented by its center. For each cluster, an occurrence distribution is computed, specifying where and at which scales the local appearance occurs on the objects. The location distribution significantly increases the discriminative power of the representation and it allows to localize the object within the image.

Matching. The next stage of many recognition methods is matching. Given a query image, features are detected and matched to the object model represented by the appearance clusters. Typically, this stage involves a distance measure, a similarity threshold, and a search technique. The distance measure and similarity threshold depend on the feature descriptor at hand. A fast search method is necessary if the object representation contains a large number of clusters. The clusters that match to query features cast votes for possible object identities, locations, and scales based on the learned location distribution. Finally, local maxima are searched in multi-dimensional voting spaces. Additional stages can be applied to refine the hypotheses and improve detection precision [12].

3 Clustering methods

In this section, we present an efficient method for clustering large numbers of features. We discuss two main clustering techniques, namely partitional K-means and agglomerative method. We propose an efficient algorithm for the latter and introduce a multi-stage procedure combining the benefits of both techniques.

K-means. The k-means algorithm [15] is one of the simplest and most popular clustering methods. It is initialized randomly by k seed points for the clusters. In all following iterations, each data point is assigned to the closest cluster center, where the centers are computed as the means of associated data points. In practice, this process converges to a local optimum within a few iterations. Many approaches employ k-means because of its computational simplicity, which is convenient for large data sets [23, 26]. Its time complexity is $O(Nk\ell d)$ when clustering N data points of d dimensions with k centers and ℓ iterations. However, the complexity is high when k is comparable with N. It can be improved by using kd-trees [22] or triangular inequality [8]. K-means is often initialized randomly, which may result in different clustering solution from run to run. Several methods [21] were proposed to overcome this problem but they add computational overhead to k-means. Finally, there is no guarantee that the obtained clusters are visually compact. Because of the fixed value of k, some cluster centers may lie in-between several real clusters, so that the centers are not representative.

Agglomerative clustering. Agglomerative clustering builds the solution by initially assigning each point to its own cluster and then repeatedly selecting and merging pairs of clusters. Thus, it builds a hierarchical merging tree from the bottom (leaves) towards the top (root). The key parameter here is the criterion used for selecting clusters to be merged. We focus on the Group Average criterion (UPGMA in [11]), which measures the similarity of two candidate clusters as the average pairwise similarity between their members. Thus, the average-link criterion allows to specify the size or compactness of the resulting clusters. This property is very useful in building compact appearance clusters and makes the algorithm robust to outliers. A similarity threshold that produces visually compact clusters only depends on the employed feature descriptors, thus can be estimated experimentally and used on different data sets. Another advantage of agglomerative methods is that given the clustering trace from a full hierarchical clustering, i.e. the indices of clusters merged in every step and the similarities between them, we can rebuild the clusters for a different similarity threshold at almost no computational cost.

The main drawback of the standard average-link algorithm is its $O(N^2 \log N)$ run-time and $O(N^2)$ space complexity. This comes from the requirement that clusters should be merged in decreasing order of similarity and that the distances must be recomputed after each agglomeration. In order to make agglomerative clustering applicable to large data sets, both complexities have to be reduced. The improvement proposed here is based on the insight from [4] that for some criteria the same clustering solution can be achieved with different merging order. Furthermore, the similarities between clusters can be efficiently recomputed based only on the centers and variances.

RNN algorithm. The improved clustering method is based on the construction of *reciprocal nearest neighbor* pairs (RNN pairs), that is of pairs of points a and b, such that a is b's nearest neighbor and vice versa [4]. RNN is applicable to clustering criteria that fulfill *reducibility property* [5] :

$$d(c_i,c_j) \leq \inf(d(c_i,c_k),d(c_j,c_k)) \Rightarrow \inf(d(c_i,c_k),d(c_j,c_k)) \leq d(c_i \cup c_j,c_k)$$

Algorithm 1 Average-Link algorithm with RNNs for R points.

$last \leftarrow -1$
while $R \neq \emptyset$ **do**
  if $last < 0$ **then** *// Initialize a new chain with a random point $v \in R$.*
    $last \leftarrow 0$; $Chain[last] \leftarrow v \in R$; $R \leftarrow R \backslash \{v\}$; $Sim[last] \leftarrow 0$; (1)
  $s \leftarrow$ findNearestNeighbor($Chain[last]$, R); $sm \leftarrow sim(Chain[last], s)$ (2)
  if $sm > Sim[last]$ **then** *// No RNNs, add s to the chain.*
    $last \leftarrow last + 1$; $Chain[last] \leftarrow s$; $R \leftarrow R \backslash \{s\}$; $Sim[last] \leftarrow sm$; (3)
  else *// Found RNNs → agglomerate the last two points in the chain*
    if $Sim[last] > SimThreshold$ **then**
      $s \leftarrow$ agglomerate($Chain[last]$,$Chain[last-1]$); $R \leftarrow R \cup \{s\}$; $last \leftarrow last - 2$; (4)
    else $last \leftarrow -1$ *// Discard the current chain.*

where c_i, c_j and c_k are clusters and $d(c_j, c_k)$ is a distance measure. This property effectively states that the agglomeration of a RNN pair does not alter the nearest-neighbor relations of other clusters. It is fulfilled for the average-link criterion regardless of the employed similarity measure. The key to an efficient implementation is therefore to ensure that RNNs can be found with as little computation as possible. This can be achieved by building a *nearest-neighbor chain* [4]. An NN-chain consists of an arbitrary point, followed by its NN, which is again followed by its NN from among the remaining points, and so on. Thus, each NN-chain ends with an RNN pair. The strategy of the algorithm is thus to start with an arbitrary point (Alg. 1, step (1)) and build up an NN-chain (2,3). As soon as an RNN pair is found, the corresponding clusters can be agglomerated (4). The reducibility property guarantees that after the last two clusters from the chain are merged, the NN assignments stay valid for the remaining chain members, which can then be used in the next iteration. Whenever the current chain is empty, a new chain is started with another random point (1). When a new cluster is created by merging an RNN pair, its new distance to other clusters has to be recomputed. Instead of expensively computing the average of all distances between cluster members, we use the following equivalence:

$$sim_{Euclid}(c_x, c_y) = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} (x^{(i)} - y^{(j)})^2 = \sigma_x^2 + \sigma_y^2 + (\mu_x - \mu_y)^2$$

where x and y are the cluster members, μ_x and μ_y are the centroids, σ_x^2 and σ_y^2 are the variances. Both the mean and variance of the new cluster can then be computed incrementally:

$$\mu_{new} = \frac{N\mu_x + M\mu_y}{N+M}, \;\; \sigma_{new}^2 = \frac{1}{N+M} \left(N\sigma_x^2 + M\sigma_y^2 + \frac{NM}{N+M} (\mu_x - \mu_y)^2 \right)$$

An amortized analysis shows that this algorithm has a computational complexity of $O(N^2 d)$ with only linear space requirements. This is an important improvement compared to the standard algorithm, since it makes it possible to cluster 100,000s of data points, which was not feasible before. However, the time complexity is still high when N is large. In the following, we present a strategy to further improve also the run-time efficiency.

Combined partitional-agglomerative algorithm (CPA). The idea of this improved algorithm is to first partition the set of features and perform agglomerative clustering within each partition independently [27]. However, there are several issues with this method. The first is how to set the partitions so that they contain features that cluster well. A possible solution is to use a natural partition of the data points, stemming from properties of the employed interest point detector. The scale invariant interest points are

detected at local maxima and minima of the Laplacian [13, 16]. If e.g. SIFT descriptors are used, which make a clear distinction between bright and dark structures, these extrema form two distinct groups which do not intersect. For other descriptors, this property has to be verified. Anothers suitable partitioning method is k-means. The number of initial partitions has to be small, otherwise k-means is not efficient. A problem occurs if a real cluster is split over several partitions, since the agglomeration is initially done between points which are NNs within one partition only. This can alter the cluster centers and variances and thus produce a different clustering tree. To reduce the impact of this effect on the final clustering solution, we agglomerate clusters within each partition only up to a certain similarity threshold. Next, given the cluster centers and variances obtained from all partitions, we continue the agglomeration up to the root of the tree. If the similarity threshold for initial clustering is smaller than for the final appearance clusters, then the initial agglomeration provides small building blocks used by the next level. However, the initial threshold should produce a number of clusters which is significantly lower than N, otherwise the complexity reduction would be limited.

To summarize the approach, we first partition features on two sets of Laplacian maxima and minima. Then we apply k-means to each set to further partition the features. Agglomerative clustering is applied within each partition. Finally, the agglomerative method is applied once more on all the cluster centers computed in the previous step. This combined partitional-agglomerative method leads to an approximate clustering solution, but as the experimental results show, the difference from the exact solution is negligible.

4 Fast matching

In this section, we propose a data structure for fast search in high dimensional feature spaces. Many fast NN search methods are based on hypercube or hyperrectangle approximations [2, 19]. They partition each feature dimension independently and trim the candidates for NN dimension by dimension. However, in this approach the efficiency depends critically on the size of the hypercube. It also relies on the fact that a single NN is searched in the whole space, for which the L_{inf} (hypercube) norm can be used. However, in object class recognition we are often interested in finding all features which are similar to our query point, for which the L_2 (hypersphere) norm is needed. Although L_2 is bounded by L_{inf}, in high-dimensional spaces the corners of the hypercube contain far more volume (data points) than the inscribed hypersphere. A solution to this problem is a data structure based on the L_2 norm. We describe a fast data structure and an algorithm for range search based only on a triangle-inequality-obeying distance metric.

4.1 Ball tree search

Ball tree structure. A ball tree (or metric tree) is a hierarchical structure for representing a set of points with the only assumption that the distance function between points is a metric [25]. Each node $(a...r)$ of the tree is represented by two parameters: center and radius (Fig. 1(a)). The node center is a mean vector of all the children nodes, and the radius is determined by the point farthest from the center. The radius can also be smaller if we are ready to accept a subset of the points similar to the query in return for a possible speedup. We propose to set the radius as a quantile of ordered feature distances from the node center.

Building ball trees. The problem of building an optimal ball tree structure is inherently similar to that of agglomerative clustering [18, 20]. In the agglomerative tree each node

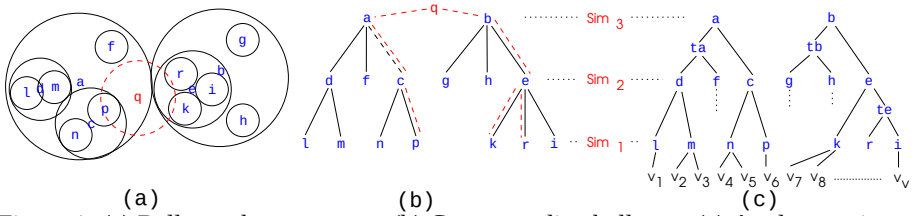

Figure 1: (a) Ball tree data structure. (b) Corresponding ball-tree. (c) Agglomerative tree.

contains two child nodes, since the algorithm merges two clusters at a time. However, given the clustering trace which contains the indices of merged clusters and their similarities, we can easily reconstruct a tree in which the number of children of a node is determined as a function of their similarity. This is illustrated in Fig. 1(b,c). Intermediate nodes ta, tb and te are merged with corresponding a, b and e. The size of the nodes is increasing from the leaves to the root of the tree. Thus we obtain a ball-tree structure from the agglomerative clustering trace with minimal additional cost.

Ball-tree search. A range search is a simple recursive procedure, which is illustrated in Figs. 1(a) and 1(b). We start by computing the similarity of a query point q to the top nodes a and b and use the triangle inequality property. The search is continued if the distance to the node center minus the nodes radius is less than the query radius, i.e. if the query ball intersects with the node ball. The search is continued further to all children nodes that intersect with the query ball. Exhaustive search is applied within each node. The speed of the search thus depends on the number of tree levels, the node radii, and the query radius. The number of levels and the node radii can be chosen experimentally at a low cost using the precomputed clustering trace. If we are only interested in the NNs, the search can be made more efficient, since the search radius can be progressively reduced with each new NN candidate that is found.

5 Experiments

In this section we present and discuss the evaluation results.

5.1 Test Data

Our test data consists of 1,000,000 scale invariant features provided by Harris-Laplace and Hessian-Laplace detectors [16] with SIFT descriptor [13]. Features are detected in 5,000 images from the Caltech database and the PASCAL set[1], containing pedestrians, cars, motorbikes, faces, and cows. To validate the results, we also compare the recognition performance of the baseline approach using the proposed clustering and matching methods and the UIUC multi-scale car set [1]. Additional experiments on more object classes can be found in [17].

5.2 Clustering

Similarity measure. As described in Section 3, the agglomerative clustering method is driven by the similarity measure and a threshold. To produce meaningful clusters we determine a reasonable range for the similarity distance using the evaluation protocol from [16], originally developed for matching pairs of images. It computes precision (i.e. the ratio of correct to false matches) and recall of matches with respect to the similarity threshold. Precision is high up to a given similarity threshold and decreases for larger

[1] http://www.pascal-network.org/challenges/VOC/

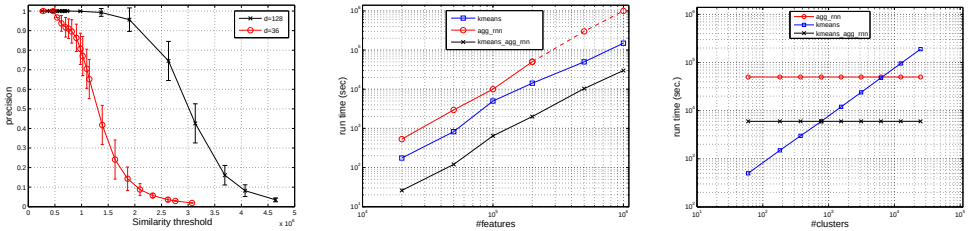

Figure 2: (a) Matching precision vs. similarity distance. (b) Run-time vs. number of features. (c) Run time vs. number of clusters.

thresholds (cf. Fig. 2(a)). The useful thresholds are in the steep part of the curve. For small thresholds, only very similar features match, resulting in a poor generalization of the model to new object instances. For large thresholds, false matches dominate, thus the recognition performance is low and the complexity increases. The above method provides a reliable and computationally inexpensive insight on the similarity thresholds that can be used for agglomerative clustering. In contrast, the size and distribution of k-means clusters depend on the k parameter, which is difficult to optimize if the real distribution of features is unknown.

Run-time. Given a set of features, we first run the RNN method with a fixed similarity threshold obtained from Fig. 2(a), which results in a number of clusters. We then run standard k-means for the same number of clusters with the maximum number of iterations set to 25. Finally, we run the CPA method with initial number of k-means partitions set to $\#features/20000$, and the initial agglomerative threshold set to half the one obtained from Fig. 2(a). Thus the methods are compared for the same number of features and the same number of clusters. Fig. 2(b) shows the run-time with respect to the number of features in the database. The run-time of CPA is an order of magnitude lower than for k-means and 2 orders of magnitude lower than the RNN algorithm. For example, clustering of 1M features takes 555h for RNN[2], 41h for k-means, and 5h for CPA.

Fig. 2(c) shows the run-time with respect to the number of resulting clusters, using 200k features. For k-means, the run time increases linearly with k. This is to be expected since the complexity is directly related to the number of clusters if the exhaustive search is used during clustering. However, it is important to note that this is the upper bound, since the run time can be shorter if convergence is obtained in less iterations, $k \approx N$, fast NN search techniques [22], or other speedups [8] are used. The run-time of the RNN is high but almost independent of the number of clusters, since most of the computation is spent at the bottom of the clustering tree, when the number of clusters is still large. For a large number of clusters, the run-time for k-means exceeds the one for RNN. From our experience, the compression ratio $\#features/\#clusters$ which gives the best recognition performance is in the range, where the proposed CPA method outperforms k-means.

Cluster quality. Fig 3(a) displays the average intra-class variance of clusters obtained with the three methods. The results are reported with respect to the number of features, and using the same number of clusters as in Fig. 2(b). Single-member clusters were discarded from this experiment in order not to bias the results. The diagram shows that the variance of clusters obtained for both agglomerative methods is lower than for k-means

[2]The run-times for RNN agglomerative clustering in the range of 500,000-1,000,000 points are estimated since we were not able to run the clustering due to time constraints.

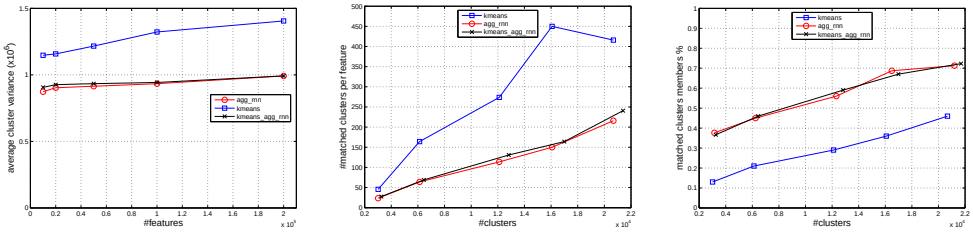

Figure 3: (a) Average variance of clusters. (b) Average number of matched clusters per feature. (c) Average percentage of matched features per matched cluster.

clusters. The variances for RNN and CPA are nearly the same. To compare the distribution of cluster centers and the compactness of the clusters, we carried out an additional experiment. We count the number of cluster centers which are within a given similarity radius of a query point (cf. Fig. 3(b)). For k-means, the number of matched clusters per query feature is significantly larger than for RNN and CPA. In Fig. 3(c) we measure how many cluster members do indeed match to the query feature. The percentage of matched cluster members is higher for agglomerated clusters. Together, these results show that the k-means clusters are less compact and therefore match to more features compared to agglomerated clusters, and that k-means cluster centers are less representative for the cluster members.

5.3 Matching

In this section we compare the efficiency of the ball tree algorithm to exhaustive search. We report the speedup factor as the ratio of run-times for 1,000 random queries. The efficiency depends on several parameters: the number of features in the dataset, the number of tree levels, the node radii, and the query radius. We have chosen experimentally 10 levels between the size of the appearance clusters (bottom nodes) and the size of the top node. The impact of the other parameters on the speedup is investigated in the following experiments. We use 200k of 128 dimensional descriptors and 200k of 36 dimensional descriptors obtained with PCA. To show the results for different numbers of features, we also use a set of 50k points with 128-dim. descriptors.

Fig. 4(a) shows the speedup factor with respect to the fraction of lost matches. We vary the radius of the nodes and compare the efficiency and the returned matches with exhaustive search. If we are looking for the exact matches, the ball tree is nearly 80 times faster than exhaustive search (for 200k features of 128 dim). This factor significantly increases up to 200 with 20% of lost matches[3]. The gain is smaller for the dataset of 50k points and for low dimensional features, which indicates that we can expect further improvement with increasing number of features and dimensions. Fig. 4(b) shows the results for different query radii (as a fraction of the top node size). The efficiency significantly drops as the size of the query increases, since many more nodes have to be examined. In most of our recognition experiments, the root node radius was 10 times larger than the size of the appearance clusters. Thus, the useful query radius is in the range of 0.1-0.2.

5.4 Recognition performance.

Finally, we compare the recognition performance of object class representations obtained with the different clustering methods. We use the UIUC multi-scale car database and

[3]While it is difficult to make a general claim how many lost matches are acceptable we experimentally observed that we can accept 10% and more lost matches without any loss in recognition performance

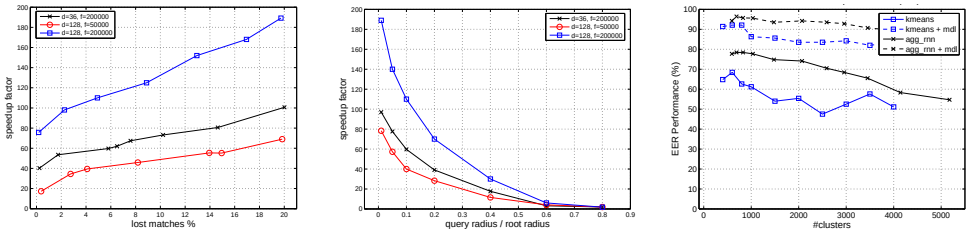

Figure 4: (a) Ball-tree speedup factor vs. number of lost matches. (b) Ball-tree speedup factor vs. query radius. (c) Recognition performance.

the evaluation criteria from [1]. We learn object representations on a training set of 50 car images from the PASCAL collection (cf. Sec. 5.1), from which we extract a total of 10,351 features with 36 dimensions. We use the evaluation criteria from [1] based on the overlap of ground truth and detected bounding boxes.

Fig. 4(c) shows precision-recall performance at the equal error rate (EER) as a function of the the number of clusters for both k-means and the agglomerative method. The solid curves depict the performance when the simple recognition approach is used (cf. Sec. 2); this performance can then still be improved by applying the MDL verification [12], as shown by the dashed curves. We make three observations. First, the recognition score is higher for agglomerated clusters (EER: 78.4%) than for k-means (EER: 68%). The methods reach different performance levels initially, but can both be taken to approximately the same performance (EERs: 96.4% and 92.1%) by the verification stage. Second, for both clustering schemes the performances degrade gracefully for different number of clusters, which is a result of our soft matching within a search hypersphere. Third, since the cost of the soft matching increases with the number of clusters that fall inside the search radius and k-means does in fact produces many more such matches for the same number of clusters (see Fig. 3(b)) we conclude that agglomerative clustering is preferable to k-means in terms of recognition costs.

Conclusions

Many of todays object class recognition approaches use clustering and matching of local features to build object models. While k-means is the most popular method, this paper shows that agglomerative clustering has several inherent properties that make it highly attractive for object class recognition: first, matching can be done efficiently using ball-tree search in high-dimensional spaces and with large numbers of clusters; second the clusters reflect the distribution of features resulting in fewer matches and lower complexity; and third, recognition performance is often better than for k-means clusters.

This paper introduces various improvements of agglomerative clustering in the context of processing large numbers of high-dimensional features. In addition, it shows how to use the clustering result to build a data structure for efficient matching. These improvements result not only in a practically feasible and efficient clustering scheme (we report clustering results up to 1,000,000 features), but also in significant speedups for matching (up to 200 times faster). Last but not least, the proposed algorithms and the expected improvements are experimentally validated.

Acknowledgments. This work has been funded, in part, by the EU project CoSy (IST-2002-004250).

References

[1] S. Agarwal, A. Awan, and D. Roth, Learning to detect objects in images via a sparse, part-based representation. *PAMI*, 26(11):1475–1490, 2004.

[2] J. Beis and D. Lowe, Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In *CVPR*, pages 1000–1006, 1997.

[3] J.L. Bentley, Multidimensional binary search trees used for associative searching. In *Communications of the ACM*, 18(9):509–517, 1975.

[4] J.P. Benzécri, Construction d'une Classification Ascendante Hiérarchique par la Recherche en Chaîne des Voisins Réciproques. *CAD*, 7(2):209–218, 1982.

[5] M. Bruynooghe, Méthodes Nouvelles en Classification Automatique des Données Taxinomiques Nombreuses. *Statistique et Analyse des Données*, 3:24–42, 1977.

[6] D. Comaniciu and P Meer, Mean Shift: A Robust Approach toward Feature Space Analysis *PAMI*, 24(5):603–619, 2002.

[7] W.H.E. Day and H. Edelsbrunner, Efficient Algorithms For Agglomerative Hierarchical Clustering Methods. *Journal of Classification*, 1:7–24, 1984.

[8] C. Elkan, Using the Triangle Inequality to Accelerate KMeans *In ICML*, pages 147-153, 2003.

[9] P. Indyk, R. Motwani, Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *STOC*, pages 604–613, 1998.

[10] A.K. Jain and R.C. Dubes, Algorithms for Clustering Data, Prentice-Hall, 1988

[11] G.N. Lance and W.T. Williams, A General Theory of Classificatory Sorting Strategies: II. Clustering Systems. *Computer Journal*,10:271–277, 1967.

[12] B. Leibe, E. Seemann, and B. Schiele, Pedestrian detection in crowded scenes. In *CVPR*, pages 878–885, 2005.

[13] D. Lowe, Distinctive image features from scale-invariant keypoints. *IJCV*, 2(60):91–110, 2004.

[14] T. Liu, A. Moore, A. Gray, and K. Yang, An Investigation of Practical Approximate Nearest Neighbor Algorithms. In *NIPS*, pages 825–832, 2004.

[15] J. MacQueen, Some Methods for Classification and Analysis of Multivariate Observations. In *Symp. on Math. Statistics and Probability*, pages 281–297, 1967.

[16] K. Mikolajczyk and C. Schmid, A performance evaluation of local descriptors. *PAMI*, 27(10):1615–1630, 2005.

[17] K. Mikolajczyk, B. Leibe and B. Schiele, Multiple Object Class Detection with a Generative Model. *CVPR*, 2006.

[18] A.W. Moore, The Anchors Hierarchy: Using the Triangle Inequality to Survive High Dimensional Data. In *UAI*, AAAI Press, pages 397–405, 2000.

[19] S. Nene, S. Nayar, A Simple Algorithm for Nearest Neighbor Search in High Dimensions. *PAMI*, 19(9):989–1003, 1997.

[20] S.M. Omohundro, Five balltree construction algorithms. Technical Report TR-89-063, 1989.

[21] K. Popat and R.W. Picard, Cluster-Based Probability Model and Its Application to Image and Texture Processing. *TIP*, 1997.

[22] V. Ramasubramanian and K.K. Paliwal, A Generalized optimization of the k-d tree for fast nearest neighbour search. *TENCON*, pages 565-568, 1989.

[23] J. Sivic and A. Zisserman, Video google: A text retrieval approach to object matching in videos. In *ICCV*. pages 1470–1478, 2003.

[24] J. Sivic, B. Russell, A. Efros, A. Zisserman, and W. Freeman, Discovering object categories in image collections. In *ICCV*, 2005.

[25] J.K. Uhlmann, Satisfying general proximity/similarity queries with metric trees. In *Information Processing Letters, 40*, pages 175–179, 1991.

[26] M. Weber, M. Welling, and P. Perona, Unsupervised Learning of Models for Recognition. In *ECCV*, pages 628-0641, 2000.

[27] Y. Zhao and G. Karypis, Evaluation of hierarchical clustering algorithms for document datasets. In *CIKM*, pages 515–524, 2002.