Appears in Proc. IEEE Conference in Computer Vision and Pattern Recognition, Kauai, Hawai, 2001

Image Indexing with Mixture Hierarchies

Nuno Vasconcelos
Compaq Computer Corporation
Cambridge Research Laboratory
nuno.vasconcelos@compaq.com

Abstract

We present an image indexing method based on a hierarchical description of the density of each of the image classes in a given database. The method is similar in spirit to traditional agglomerative clustering procedures but produces a complete mixture density, instead of a representative point, at each node of the indexing tree. Estimation of the density at a given node only requires knowledge of the mixture parameters of the children nodes, not the original data. The process is very flexible and efficient, therefore suited to problems involving large databases where existing groupings may have to be combined, or new groupings created, frequently. Experimental results show that the new indexing structure consistently outperforms a linear search when both efficiency and retrieval accuracy are taken into account.

1 Introduction

A significant amount of work has been recently devoted to the topic of image retrieval in the vision and image processing literatures. While substantial attention has been devoted to aspects such as finding suitable features [9], robust representations for visual appearance [13, 5], appropriate metrics of similarity [11], and relevance feedback mechanisms [6], much smaller progress has been achieved on the indexing problem, i.e. the development of mechanisms that guarantee sub-linear complexity in the database size. In fact, the assumption that a linear search through the entire database is acceptable is still commonly adopted. When indexing mechanisms are employed, they tend to be standard solutions from the database and text retrieval literatures, e.g. filtering [4], combination of scalar indexes via join operations [12], or traditional clustering techniques [9, 17]. When applied to image data these techniques present various limitations, such as not scaling well with the dimension of the feature space, not providing sub-linear growth on the database size, or not being able to accommodate feature representations other than points on a metric space.

Even more problematic is the assumption that indexing must always be driven by visual similarity. We contend that this is not really the case, since indexes can also be derived from image groupings determined by other information sources. Some examples are:

- text: probably the most common. Images can be labeled either manually or automatically by analyzing associated documents, e.g. web pages.
- image acquisition: most digital imaging devices possess some sense of context, e.g. date or GPS information, which is stored in image headers.
- automatic grouping: can be generated in multiple ways. For example, it is relatively easy to segment a video stream into its component shots.

Particularly interesting are those scenarios where groupings are not obtained by visual similarity but reflect some sort of semantic image categorization. Using semantic groupings to constrain the visual similarity search has various benefits. First, retrieval accuracy is likely to increase, since semantic similarity is usually what users are looking for. Second, even when there are errors, if the images are returned according to the grouping structure, the results appear coherent. E.g. if in response to a query for birds the system returns all images or airplanes followed by all images of birds, the results are easier to interpret ("it thinks that airplanes are birds") than if images of birds, airplanes, helicopters, and a few other classes all appear interspersed. This coherence leads to less confusing interaction for naive users. Third, there is more flexibility for interface design. For example, a mode where only a representative picture from each class is returned, allowing the user to quickly zoom in on the classes of interest and limiting subsequent searches to those.

When such class groupings exist (or even if they are derived from image similarity) it makes sense to rely on hierarchical indexing mechanisms. The basic idea is to group images into classes and then perform queries at the class level instead of the image level. Once the class that best explains the query is identified, the images belonging to that class can be searched for the best match. The process can, of course, be iterated by grouping classes into meta-classes and so on. In the extreme, a complete tree can be built. If there are I images and, on average, C images are grouped into each class a tree search will have complexity $O(C \log I)$ instead of $O(I)$.

The main limitation of traditional hierarchical indexing mechanisms, e.g. hierarchical clustering, is their inability to deal with representations other than points. For image databases this implies one of two approaches: segmenting each image into smaller regions that can be represented as vectors and indexed individually [9], or compressing that image into a global description that can be stored as a vector, e.g. a color histogram [17, 7]. Both approaches have significant problems: indexing of individual segments makes the problem yet more complicated (because there are many more of them) and creates serious difficulties for queries involving more than one segment; global descriptions by a single vector tend to discard too much information and can hurt retrieval performance.

This paper extends hierarchical indexing techniques, e.g. agglomerative clustering, to functional image descriptors, namely the complete probability density function of the features extracted from each image class. This description combines the advantages of the two previous approaches: on one hand a detailed characterization of local appearance is still available (through the feature values), on the other that characterization is summarized into a compact representation (via a probability density). When compared to color histograms, the feature densities now considered have the additional advantage of capturing properties like texture or local surface curvature, therefore providing a significantly more accurate image representation [14].

We adopt the Gaussian mixture as the density model, and present an algorithm for propagating mixture parameters through an indexing tree. The algorithm is akin to traditional agglomerative clustering procedures, but proceeds by clustering Gaussians, instead of points. This leads to a very efficient procedure for building indexing structures, since the estimation of the parameters of each node only requires the manipulation of the parameters of the nodes immediately below, not the original data. Hence, the procedure is specially suited for applications where large indexing structures must be updated frequently, as is the case of image databases. The resulting hierarchical density estimates are shown to have interesting properties, the most salient of which is an intrinsic regularization mechanism which guarantees that generalization power increases as one moves up in the hierarchy. In result, hierarchical searches exhibit consistently better performance than linear searches: for sparsely populated databases they are significantly more accurate, for densely populated databases they are significantly more efficient.

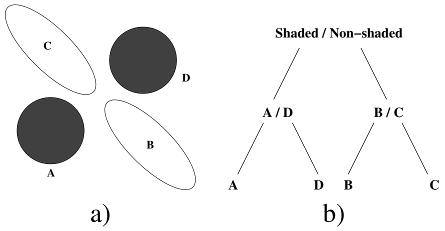

Figure 1: a) Two-level classification problem, with four images (A to D) from two classes. b) The corresponding hierarchical decision structure.

2 Hierarchical indexing

Given a database with many image classes (one class per image in the extreme case) and a query image, the role of the retrieval system is to identify the database class that best explains the query. Decision theory provides a sound formulation of the problem, making it possible to derive optimal decision functions. In particular, when the goal is to minimize the probability of retrieval error, the problem becomes one of classification, for which various solutions are available. In this context, hierarchical indexing is a problem of hierarchical classification where classes in the higher levels of the hierarchy contain more images than those in the levels below. Once a particular classification architecture is selected, the classification process is straightforward: the decision rule is first applied at the top level, the best class selected, the decision rule is re-applied to its children and so on. This is illustrated by Figure 1 a) where we depict an hypothetical two-level hierarchical classification problem.

In the figure, the densities of four images (A to D) are represented by circles and ellipses. The images are grouped into two classes, depicted as “shaded” vs “non-shaded”. As shown in Figure 1 b), the problem can be captured by a two-level hierarchical decision structure. Notice that while each circle is linearly separable from the remaining image classes when viewed in isolation, this property is lost when one considers the whole shaded class. This illustrates how the top-level classification problem can be arbitrarily more complex than the low-level ones. The main goal of hierarchical indexing is to devise a hierarchical classification architecture with 1) performance equivalent to, or better than, that of a flat classifier while 2) achieving significant computational savings.

3. Hierarchical classifiers

Existing hierarchical classification architectures can be grouped into two major categories: top-down vs. bottom-up. Top-down architectures start by designing the classifier at the top level, using all the available data, and then recursively subdivide each of the top classes to create the lower levels. They include popular techniques such as decision-trees [1], tree structured quantizers [3], and mixtures of experts [8], among others. Bottom-up procedures start from a classifier at the bottom level and recursively re-arrange the decision boundaries to obtain the classifiers at the higher levels. The most common example of such techniques is the combination of agglomerative clustering with a nearest neighbor classifier [2]. Here, each image is represented as a point in some high-dimensional space and a set of similar points are combined to form a class. A representative point is then selected for the class, e.g. the centroid of its constituents, and the process iterated. For retrieval, the query is compared to the representative of each class and the closest one is selected.

Both top-down and bottom-up architectures present problems for hierarchical indexing. The main limitation of top-down procedures is that, in the retrieval context, classes are rarely defined at the outset. In fact, both the database and the classes themselves are continuously changing, through the addition of either new images or new grouping information (e.g. keywords). It is therefore crucial to rely on architectures that can be regularly updated without an overwhelming amount of computation. Since, for top-down architectures, the design of the classifiers that compose each level of the tree requires processing a training set containing all the images in the database, the overall cost is proportional to the product of the total number of feature vectors in the database and the tree depth. This is usually too high to allow frequent updates.

On the other hand, existing bottom-up procedures are seldom applicable when 1) images are not represented as points on a vector space or 2) class groupings are not driven by visual similarity. The latter problem is illustrated by Figure 1, where the "shaded" and "non-shaded" classes are contrary to the idea of combining an image with its nearest neighbors. Since each shaded circle is closer to the non-shaded ellipses than to the other circle, grouping the two circles implies that the corresponding images are grouped with their "most distant neighbors". While, given these groupings, one could still use standard agglomerative procedures to find class representatives, such a strategy would be unlikely to work well. In the example, this would imply representing both ellipses and circles by their centroids and using the mean of these centroids as the class representative. In result, the two class representatives would end up in the exact same spot, the least desirable situation from the classification point of view.

Given the limitations of both top-down and standard bottom-up solutions, there is a clear need for alternative architectures. We next consider an extension of bottom-up strategies to probability densities.

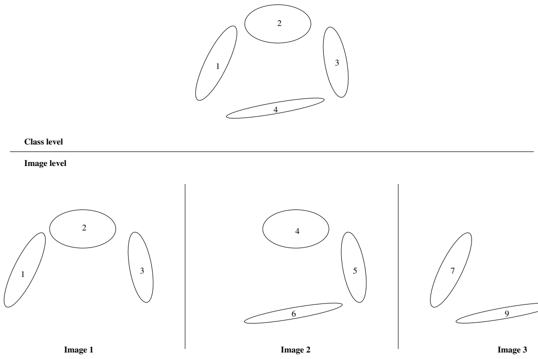

Figure 2: A two-level mixture hierarchy: one mixture of four Gaussians at the class level, three mixtures of three Gaussians at the bottom.

4. Gaussian mixture hierarchies

Solving the problem of Figure 1 with Gaussian mixtures is trivial. Consider, for example, the shaded class. Simply adding up the Gaussian densities that characterize each of the images (and scaling by 1/2) is enough to obtain the mixture density for the entire class. The simplicity of this solution is, however, an artifact of the lack of overlap between the individual densities in this example. A more realistic situation is depicted in Figure 2, where we have a mixture of four Gaussians as the class density and three images, each with density containing only three Gaussians. In practice this situation could arise from occlusion, e.g. an object with four differently textured parts, only three of which are visible in any given view.

While simply adding up the individual Gaussians would still lead (after proper re-scaling) to the true class density, the resulting representation would not be computationally efficient. In fact, one would end up with a total of 9 Gaussians (some sharing the same parameters) instead of 4, and there would be no benefit of performing a hierarchical over a linear search. On the contrary, the cost of the former would be greater than that of the latter. To obtain 4 Gaussians one needs some sort of clustering procedure. The main difficulty is that, unlike standard clustering applications, the goal is to cluster Gaussians, not points.

4.1. Hierarchic model

In order to address this problem we need to introduce some notation. We denote by $\mathcal{M}_l$ the set of mixture parameters for the density of level l (level l being the parent of $l+1$), and by $P(\mathbf{x}|\mathcal{M}_l)$ the density itself. The total number of mixture components at level l, C^l is assumed to be known[1]

$$P(\mathbf{x}|\mathcal{M}_l) = \sum_{i=1}^{C^l} \pi_i^l P(\mathbf{x}|\mathbf{z}^l = \mathbf{e}_i, \mathcal{M}_{l,i}) \tag{1}$$

where $\mathbf{e}_i$ is the i^{th} element of the canonical basis of R^{C^l}, $\mathbf{z}^l$ an indicator vector such that $\mathbf{z}^l = \mathbf{e}_i$ if and only if $\mathbf{x}$ is a sample from the i^{th} component, $\mathcal{M}_{l,i}$ the parameters of that component, and $\pi_i^l = P(\mathbf{z}^l = \mathbf{e}_i|\mathcal{M}_{l,i})$. Given a collection of children densities at level $l+1$, the model for that level is obtained by summing them and normalizing the π_i^{l+1} so that they sum to one. E.g., in Figure 2, $l \in \{0, 1\}$ with $C^0 = 4$ and $C^1 = 9$.

Models in two consecutive levels are related by a *permutation matrix* $\mathbf{P}$ such that $\mathbf{z}^{l+1} = \mathbf{P}\mathbf{z}^l$, and $p_{i,j} = 1$ indicates that i^{th} component of $\mathcal{M}_{l+1}$ is is a copy of the j^{th} component of $\mathcal{M}_l$,

$$P(\mathbf{x}|\mathbf{z}^{l+1} = \mathbf{e}_i, p_{i,j} = 1, \mathcal{M}_{l+1,i}) = P(\mathbf{x}|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_{l,j}). \tag{2}$$

This condition is sufficient to guarantee the consistency of the hierarchical representation since, when it holds,

$$\begin{aligned}
&P(\mathbf{x}|\mathcal{M}_{l+1}) = \\
&= \sum_{i=1}^{C^{l+1}} \pi_i^{l+1} P(\mathbf{x}|\mathbf{z}^{l+1} = \mathbf{e}_i, \mathcal{M}_{l+1,i}) \\
&= \sum_{i=1}^{C^{l+1}} \pi_i^{l+1} \sum_{j=1}^{C^l} P(\mathbf{x}|\mathbf{z}^{l+1} = \mathbf{e}_i, p_{i,j} = 1, \mathcal{M}_{l+1,i}) \\
&\quad P(p_{i,j} = 1|\mathbf{z}^{l+1} = \mathbf{e}_i) \\
&= \sum_{j=1}^{C^l} P(\mathbf{x}|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_{l,j}) \sum_{i=1}^{C^{l+1}} P(\mathbf{z}^l = \mathbf{e}_j|\mathbf{z}^{l+1} = \mathbf{e}_i)\pi_i^{l+1} \\
&= \sum_{j=1}^{C^l} P(\mathbf{x}|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_{l,j})\pi_j^l = P(\mathbf{x}|\mathcal{M}_l),
\end{aligned}$$

i.e. the entire density of the children is propagated to the parent. In addition to it, we impose a condition equivalent to sampling with replacement

$$P(p_{i,j} = 1|\mathbf{z}^{l+1} = \mathbf{e}_i) = \pi_j^l \tag{3}$$

or, equivalently,

$$P(\mathbf{z}^l = \mathbf{e}_j|\mathbf{z}^{l+1} = \mathbf{e}_i) = \pi_j^l \tag{4}$$

i.e. the probability that $\mathcal{M}_{l+1,i}$ is a replica of $\mathcal{M}_{l,j}$ does not depend on i. Since, when it holds,

$$\begin{aligned}
&P(\mathbf{x}|\mathbf{z}^{l+1} = \mathbf{e}_i, \mathcal{M}_l) = \\
&= \sum_{j=1}^{C^l} P(\mathbf{x}|\mathbf{z}^{l+1} = \mathbf{e}_i, p_{i,j} = 1, \mathcal{M}_l) \\
&\quad P(p_{i,j} = 1|\mathbf{z}^{l+1} = \mathbf{e}_i) \\
&= \sum_{j=1}^{C^l} \pi_j^l P(\mathbf{x}|\mathbf{z}^{l+1} = \mathbf{e}_i, \mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_l) \\
&= \sum_{j=1}^{C^l} \pi_j^l P(\mathbf{x}|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_{l,j}) = P(\mathbf{x}|\mathcal{M}_l)
\end{aligned}$$

this condition guarantees that the density at level l is not affected by reordering the components at level $l+1$. Such reordering only affects the permutation matrix $\mathbf{P}$.

4.2. Propagating parameters

Given the parameters of the mixture model at the bottom of the tree and the number of mixture components in the remaining levels, the goal is to infer the parameters of the mixtures at all those levels. Consider levels $l+1$ and l and assume that $\mathcal{M}_{l+1}$ is known. It is therefore possible to draw a sample of independent observations from $P(\mathbf{x}|\mathcal{M}_{l+1})$ consisting of a sequence of pairs $\{(\tilde{\mathbf{x}}_m, \tilde{\mathbf{z}}_m^{l+1})\}_{m=1}^N$. These can be grouped into the sequence $\{(\hat{\mathbf{x}}_i, \mathbf{e}_i)\}_{i=1}^{C^{l+1}}$, where $\hat{\mathbf{x}}_i = \{\tilde{\mathbf{x}}_m|\tilde{\mathbf{z}}_m^{l+1} = \mathbf{e}_i\}$. We next evaluate the likelihood of the sample $\hat{\mathbf{x}} = \{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_{C^{l+1}}\}$ under the model of level l,

$$P(\hat{\mathbf{x}}|\mathcal{M}_l) = \prod_{i=1}^{C^{l+1}} P(\hat{\mathbf{x}}_i|\mathcal{M}_l). \tag{5}$$

From (1), this is a problem of estimating the parameters of a mixture, that can be solved by expectation-maximization (EM). In the E-step we compute the assignment of the $\hat{\mathbf{x}}_i$ to the $P(\hat{\mathbf{x}}|\mathcal{M}_{l,j})$,

$$\begin{aligned}
h_{ij} &= P(\mathbf{z}^l = \mathbf{e}_j|\hat{\mathbf{x}}_i, \mathbf{z}^{l+1} = \mathbf{e}_i, \mathcal{M}_l) \\
&= \frac{P(\hat{\mathbf{x}}_i|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_l)\pi_j^l}{\sum_k P(\hat{\mathbf{x}}_i|\mathbf{z}^l = \mathbf{e}_k, \mathcal{M}_l)\pi_k^l}.
\end{aligned}$$

The key quantity to compute is therefore $P(\hat{\mathbf{x}}_i|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_l)$. Taking its logarithm

$$\log P(\hat{\mathbf{x}}_i|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_l) = \tag{6}$$

$$\begin{aligned}
&= M_i[\frac{1}{M_i}\sum_{i=1}^{M_i} \log P(\tilde{\mathbf{x}}_i^m|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_l)] \\
&= M_i E_{\mathcal{M}_{l+1,i}}[\log P(\mathbf{x}|\mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_l)],
\end{aligned} \tag{7}$$

where we have used the law of large numbers, M_i is the cardinality of $\hat{\mathbf{x}}_i$, $M_i = \lfloor \pi_i^{l+1} N \rfloor$, and $E_{\mathcal{M}_{l+1,i}}[\mathbf{x}]$ the

[1] An assumption that always holds for the children densities, but usually does not for the parent. Determining automatically the number of parent Gaussians is a topic for future work, but we will later show that the procedure now proposed is very robust with respect to incorrect guesses for this parameter.

expected value of $\mathbf{x}$ according the i^{th} mixture component of $\mathcal{M}_{l+1}$ (the one from which the observations in $\hat{\mathbf{x}}_i$ are drawn). It can be shown [16] that for Gaussian components, $\mathcal{M}_{l,j} = \{\mu_j^l, \mathbf{\Sigma}_j^l\}$, it leads to

$$h_{ij} = \frac{\left[\mathcal{G}(\mu_i^{l+1}, \mu_j^l, \mathbf{\Sigma}_j^l) e^{-\frac{1}{2} trace\{(\mathbf{\Sigma}_j^l)^{-1} \mathbf{\Sigma}_i^{l+1}\}}\right]^{M_i} \pi_j^l}{\sum_k \left[\mathcal{G}(\mu_i^{l+1}, \mu_k^l, \mathbf{\Sigma}_k^l) e^{-\frac{1}{2} trace\{(\mathbf{\Sigma}_k^l)^{-1} \mathbf{\Sigma}_i^{l+1}\}}\right]^{M_i} \pi_k^l}, \tag{8}$$

where $\mathcal{G}(\mathbf{x}, \mu, \mathbf{\Sigma})$ is a Gaussian with mean μ and covariance $\mathbf{\Sigma}$. The M-step consists of maximizing

$$Q = \sum_{i=1}^{C^{l+1}} \sum_{j=1}^{C^l} h_{ij} \log(\pi_j^l P(\hat{\mathbf{x}}_i | \mathbf{z}^l = \mathbf{e}_j, \mathcal{M}_l)) \tag{9}$$

subject to the constraint $\sum_j \pi_j^l = 1$. In the Gaussian case it leads to the following parameter updates [16]

$$\pi_j^l = \frac{\sum_i h_{ij}}{C^{l+1}} \tag{10}$$

$$\mu_j^l = \sum_i w_{i,j} \mu_i^{l+1}, \text{where } w_{i,j} = \frac{h_{ij} \pi_i^{l+1}}{\sum_i h_{ij} \pi_i^{l+1}} \tag{11}$$

$$\mathbf{\Sigma}_j^l = \sum_i w_{i,j} \left[\mathbf{\Sigma}_i^{l+1} + (\mu_i^{l+1} - \mu_j^l)(\mu_i^{l+1} - \mu_j^l)^T\right] \tag{12}$$

Note that neither (8) nor (10) to (12) depend explicitly on the underlying sample $\hat{\mathbf{x}}_i$ and can be computed directly from the parameters of $\mathcal{M}_{l+1}$. The algorithm is thus very efficient from a computational standpoint.

5. Similarity function

One important issue for hierarchical retrieval is the choice of similarity function, which should be able to account for partial matches. Given a query image, it is unlikely that it will exhibit all the variation found in the classes of which it is a member. Hence metrics that integrate some function of the pointwise distance between two densities over their entire support, e.g. correlation or L^p norms, are doomed to be unsuccessful.

A better alternative is to evaluate the likelihood of the query image under the database density. For a set $\mathbf{F} = \{\mathbf{f}_i\}_{i=1}^M$ of query features and a collection of database densities $\{P_i(\mathbf{x})\}_{i=1}^I$ the retrieval criteria becomes maximum likelihood (ML)

$$i^* = \arg\max_i \sum_k \log P_i(\mathbf{f}_k). \tag{13}$$

Under ML a good match can be achieved even when the features $\mathbf{f}_k$ populate only a fraction of the region of support of $P_i(\mathbf{x})$. There is, however one problem: the linear cost of ML on the cardinality M of the query. Computationally more efficient similarity functions can be derived from the fact that, for large M, ML is equivalent to minimizing Kullback-Leibler divergence (KLD)

$$i^* = \arg\min_i KL(P_q(\mathbf{x}) || P_i(\mathbf{x})), \tag{14}$$

where $P_q(\mathbf{x})$ is the density of the query and $KL(p||q) = \int p(\mathbf{x}) \log[p(\mathbf{x})/q(\mathbf{x})] d\mathbf{x}$ the KLD between densities p and q. The main difficulty is that the KLD has no closed form expression when p and q are mixtures. One way to circumvent the problem is to use approximations. We rely on the asymptotic likelihood approximation (ALA) [15]. For a query mixture $P(\mathbf{x}|\mathcal{M}_q)$ with $\mathcal{M}_q = \{\pi_{q,j}, \mu_{q,j}, \mathbf{\Sigma}_{q,j}\}_{j=1}^{C_q}$ and a database mixture $P_i(\mathbf{x}|\mathcal{M}_i)$ with $\mathcal{M}_i = \{\pi_{i,j}, \mu_{i,j}, \mathbf{\Sigma}_{i,j}\}_{j=1}^{C_i}$,

$$ALA(P||P_i) \approx \sum_j \pi_{q,j} \left\{ \log \pi_{i,\alpha(j)} + \left[\log \mathcal{G}(\mu_{q,j}, \mu_{i,\alpha(j)}, \mathbf{\Sigma}_{i,\alpha(j)}) - \frac{1}{2} trace[\mathbf{\Sigma}_{i,\alpha(j)}^{-1} \mathbf{\Sigma}_{q,j}] \right] \right\},$$

where

$$\alpha(j) = k \iff ||\mu_{q,j} - \mu_{i,k}||^2 < ||\mu_{q,j} - \mu_{i,l}||^2, \forall l \neq k.$$

The ALA is equivalent to the KLD when all the covariances are zero, i.e. when the density models are vector quantizers. In the generic Gaussian mixture case, it has been shown to provide a good approximation to the KLD when the features are high-dimensional [15].

6 Experimental evaluation

To evaluate the efficiency of hierarchical indexing we conducted experiments with two databases: the Columbia object database and the Corel database of stock photography. Columbia is a set of images of 100 objects each shot in 72 different views obtained by rotating the object in $3D$ in steps of 5^o. For computational simplicity, we only considered a subset of 9 views per object (separated by 40^o). This subset was split into two subgroups, a *query* database containing the first image of each object and a *retrieval* database containing the remaining 8. In the case of Corel, we selected 15 image classes[2] with 100 images each. Once again we created a query and retrieval database, this time by assigning each image to the query set with a probability 0.2.

[2] "Arabian horses," "auto racing," "coasts," "divers and diving," "English country gardens," "fireworks," "glaciers and mountains," "Mayan and Aztec ruins," "oil paintings," "owls," "land of the pyramids," "roses," "ski scenes," "religious stained glass."

6.1 Image representation

All experiments were based on the following set-up. First, all images were converted from the RGB to the YBR color space [10]. We then extracted a collection of 8×8 blocks from each image (with half-block of overlap in each dimension between neighboring blocks) and computed the discrete cosine transform [10] of each block. Coefficients from the spatially co-located blocks of the three color channels were then interleaved according to the pattern YBRYBR..., leading to a 192 dimensional vector, of which only the 64 lower frequency coefficients were retained. A Gaussian mixture was fitted, by EM, to the sample extracted from each image. To avoid overfitting, we relied on a fixed number (8) of Gaussians which were constrained to have diagonal covariance. While some of these choices may seem arbitrary, most have a theoretical justification and were found to perform well in practice. See [14] for details.

6.2 Experimental set up

We compared three retrieval strategies: linear search (LS), hierarchical search with the Gaussian mixture hierarchy learned with the algorithm of section 4.2 (HGM), and hierarchical search with all models learned directly from data (HData). A two-level hierarchy, determined by the properties of the database under study, was used for both HGM and HData. For each class, the top-level density was obtained by pooling all the individual image densities in that class. Hence, for HGM, learning each model of level 0 had complexity $O(KC^0C^1)$, where K is the average number of images per class and C^i the number of mixture components at level i. On the other hand, HData had complexity $O(KC^0M)$ where M is the number of features per image. Since $M >> C^1$ the time required to learn a model under HData was significantly higher than that required by HGM. On Columbia, classes consisted of multiple views of the same object, leading to 100 Gaussian mixtures in the top level, each having 8 children models. On Corel, all images from each of the 15 classes enumerated above were grouped together, leading to 15 top-level models each having, on average, 80 children. No images from the query database were used for the construction of the hierarchical models.

6.3. Computational cost

The dramatic reduction in learning complexity of HGM over HData is visible in the top plot of Figure 3, where we show learning times as a function of C^0 for Columbia[3]. On this database, the learning cost of HGM was about 40 times smaller than that of HData, e.g. 21.1 seconds per model for the former vs. 14.2 minutes for the latter, when $C^0 = 64$.

[3]Because HData is so expensive we could not even use it with the larger Corel database.

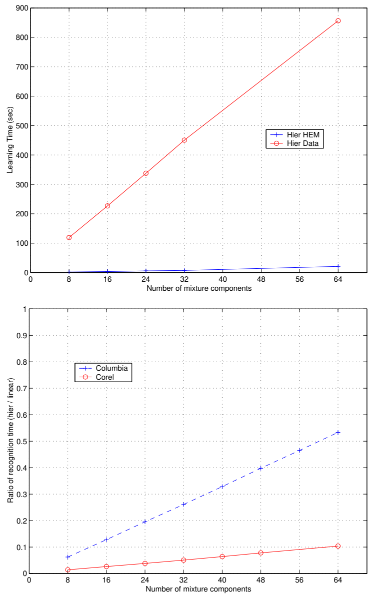

Figure 3: Computational complexity as a function of C^0. Top: Learning times for HGM and HData. Bottom: Ratio of hierarchical/linear search time for Columbia and Corel.

Because, like HData, most of the top-down procedures discussed in section 3 require processing the original data to build the classifiers of each intermediate level, these results provide a powerful illustration of the advantages of bottom-up strategies.

The computational advantages of hierarchical over linear search are illustrated in the bottom plot of the figure, where we present the ratio hierarchical/linear query time as a function of C^0 for both databases. Clearly, the gains can be substantial (e.g. between one and two orders of magnitude when there are 8 components in the top-level mixtures) and are determined by the structure of the database. Columbia has sparsely populated classes (8 images per class) and, therefore has a smaller ratio between the number of mixture models in the two levels (1 at the top per 8 at the bottom, against 1 per 80 on Corel). Densely populated classes, such as those of Corel, have the largest potential for computational savings provided that it is possible to reduce C^0 to the minimum possible. This emphasizes the importance of hierarchical clustering.

6.4. Retrieval accuracy

The results observed so far are not surprising, mostly confirming that hierarchical searches are more efficient and HGM has smaller learning times than HData. A more interesting question is, how much loss in retrieval accuracy is associated with going from a linear to a hierarchical search and from a hierarchical search using HData to one using

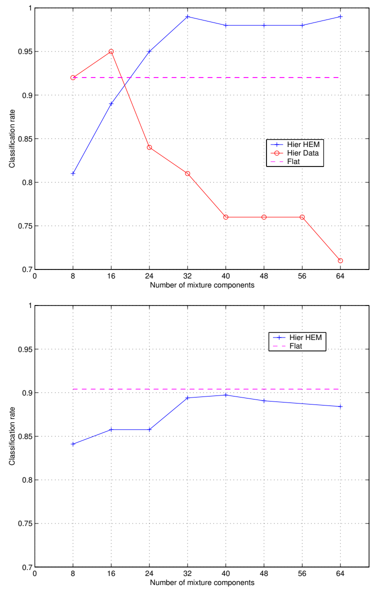

Figure 4: Recognition rate as a function of C^0. Top: Columbia. Bottom: Corel. In both cases the rate achieved with LS is shown as an horizontal dashed line.

HGM? Intuitively, losses seem inevitable: how could 1) learning densities hierarchically beat learning them directly from the data? and 2) a search among class models beat an exhaustive search among all image models? Contrary to this intuition, the results in Figure 4 show that, given enough mixture components to achieve reasonable density estimates, *the retrieval performance of HGM is indeed superior to that of HData and equal or superior to that of LS*.

In fact, the results on Columbia show that the improvements can be quite significant: while 1) LS only achieves 92% accuracy, and 2) HData achieves 95% but can also perform very poorly, HGM achieves 99% and shows great robustness to variations in the number of mixture components at the top level. On Corel, LS and HGM have similar accuracy. These results indicate that, in terms of retrieval accuracy, HGM will 1) actually beat the other approaches when classes are sparsely populated, and 2) achieve equivalent performance when they are densely populated.

The ability to better handle sparsely populated classes, suggests that HGM generalizes much better than the two other techniques. To understand the advantages over LS we return to Figure 2. Suppose that one of the images, e.g. image 3, is used as a query while the other two are stored in the database. Since the query only has 2 Gaussians in common with each of the database images, there is always one Gaussian that cannot be explained. None of the two database images is therefore a very good match and the probability that some image from another class will be considered more similar to the query is not negligible. On the other hand, even if learned only from two of the images, the top-level model will contain all four Gaussians (albeit not correctly weighted) and will provide a similarity score that is significantly harder to beat[4].

To understand the gains of HGM over HData we go back to (9). Notice that the covariance of the Gaussian at level l is a weighted sum of the covariances and scatter of its children densities. This implies that the variances of each parent are lower-bounded by the corresponding children variances and, therefore, variances can never decrease as one moves up the hierarchy. This regularization makes estimates higher up in the hierarchy much more robust than those at the bottom. Since no such regularization is in place for HData, some of the Gaussians tend to specialize on small clusters of non-typical points leading to over-fitting. Notice, in the top plot of Figure 4, how performance quickly decays after the "optimal" number of components and contrast it with that of HGM which is very robust to variations of C^0.

Further evidence that HGM provides good generalization is presented in Figure 5, which depicts four queries at the class level. In each case, the query image is shown in the top left, and representative images from each of the top classes are then shown in raster-scan order[5]. These top classes tend to be objects that are visually similar to that in the query.

Overall, the experimental results show that HGM is the superior method, among the three evaluated, when both computational efficiency and retrieval accuracy are taken into account. For sparse databases, HGM provides some computational savings and significant improvements in retrieval accuracy. For dense databases, accuracy is equivalent to that of LS but the computational savings become very significant.

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman & Hall, 1993.

[2] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[3] A. Gersho and R. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Press, 1992.

[4] J. Hafner, H. Sawhney, W. Equitz, M. Flickner, and W. Niblack. Efficient Color Histogram Indexing for

[4] Notice that the performance of LS would not necessarily improve by using a k-nearest neighbors type of solution, i.e. taking the two image matches into account through some voting mechanism. The fact is that both image models fail to explain one third of the query and, therefore, they can both receive low similarity ranks. We have actually tried combining similarity scores through various majority voting rules and none worked better than the simple LS on the two databases considered.

[5] Class representative were selected for display only, and play no special role in the retrieval operation itself.

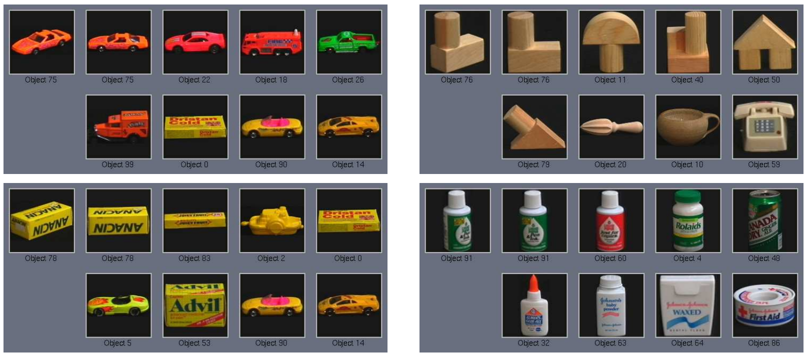

Figure 5: Examples of retrieval from class models.

Quadratic Form Distance Functions. *IEEE Trans. on Pattern. Analysis and Machine Intelligence*, 17(7):729–736, July 1995.

[5] J. Huang, S. Kumar, M. Mitra, W. Zhu, and R. Zabih. Spatial Color Indexing and Applications. *Int. Journal of Computer Vision*, 35(3):245–268, December 1999.

[6] T. Huang and X. Zhou. Image Retrieval and Relevance Feedback: From Heuristic Weight Adjustment to Optimal Learning Methods. In *IEEE Int. Conf. on Image Processing*, Thessaloniki, Greece, 2001.

[7] G. Iyengar and A. Lippman. Clustering Images Using Relative Entropy for Efficient Retrieval. In *International workshop on Very Low Bitrate Video Coding, Urbana, Illinois*, 1998.

[8] M. Jordan and R. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6:181–214, 1994.

[9] T. Minka and R. Picard. Interactive learning using a "society of models". *Pattern Recognition*, 30:565–582, 1997.

[10] W. Pennebaker and J. Mitchell. *JPEG: Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[11] Y. Rubner, C. Tomasi, and L. Guibas. A Metric for Distributions with Applications to Image Databases. In *International Conference on Computer Vision, Bombay, India*, 1998.

[12] J. Smith and S. Chang. VisualSEEk: a fully automated content-based image query system. In *ACM Multimedia, Boston, Massachussetts*, pages 87–98, 1996.

[13] M. Swain and D. Ballard. Color Indexing. *International Journal of Computer Vision*, Vol. 7(1):11–32, 1991.

[14] N. Vasconcelos. *Bayesian Models for Visual Information Retrieval*. PhD thesis, Massachusetts Institute of Technology, 2000.

[15] N. Vasconcelos. On the Complexity of Probabilistic Image Retrieval. In *Proc. International Conference on Computer Vision*, Vancouver, Canada, 2001.

[16] N. Vasconcelos and A. Lippman. Learning Mixture Hierarchies. In *Neural Information Processing Systems, Denver, Colorado*, 1998.

[17] A. Vellaikal and C. Kuo. Hierarchical Clustering Techniques for Image Database Organization and Summarization. In *SPIE Multimedia Storage and Archiving Systems III, Boston*, 1998.