M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 349
Submitted to Special Issue of Pattern Recognition on Image Database: Classification and Retrieval

Interactive learning using a "society of models" *

T. P. Minka and R. W. Picard
Vision and Modeling Group
MIT Media Laboratory
20 Ames Street; Cambridge, MA 02139
{tpminka, picard}@media.mit.edu

Abstract

Digital library access is driven by features, but features are often context-dependent and noisy, and their relevance for a query is not always obvious. This paper describes an approach for utilizing many data-dependent, user-dependent, and task-dependent features in a semi-automated tool. Instead of requiring universal similarity measures or manual selection of relevant features, the approach provides a learning algorithm for selecting and combining groupings of the data, where groupings can be induced by highly specialized and context-dependent features. The selection process is guided by a rich example-based interaction with the user. The inherent combinatorics of using multiple features is reduced by a multistage grouping generation, weighting, and collection process. The stages closest to the user are trained fastest and slowly propagate their adaptations back to earlier stages. The weighting stage adapts the collection stage's search space across uses, so that, in later interactions, good groupings are found given few examples from the user. Described is an interactive-time implementation of this architecture for semi-automatic within-image segmentation and across-image labeling, driven by concurrently active color models, texture models, or manually-provided groupings.

1 Issues for digital libraries

Digital libraries of images, video, and sound are a rich area for pattern recognition research. They also introduce a host of new problems and requirements, since the range of possible queries is immense and requires the utilization of many specialized features. Also, systems for retrieval, browsing, and annotation, i.e. classifying regions, often must perform with only a small number of examples from a user, i.e. an insufficient amount of training data by traditional requirements. Thus the area is doubly exciting since it presents the field of pattern recognition with new challenges while beckoning in new applications.

One important issue for digital libraries is finding good models and similarity measures for comparing database entries. A part of this difficulty is that feature extraction and comparison methods are highly data-dependent; see Figure 2 for an example with texture. Similarity measures are also user and task dependent, as demonstrated by Figure 3. Unfortunately, these dependencies are not, at this point, understood well enough, especially by the typical digital library user, to permit careful selection of the optimal measure beforehand. Note that the multi-resolution simultaneous autoregressive (MRSAR) model of [1], which fares poorly compared to the shift-invariant eigenvector (EV) model in the above two examples, scores clearly above the EV model on the standard Brodatz database [2] [3]. (On the same test data, but for a perceptually motivated similarity criteria based on periodicity, directionality, and randomness, both the EV and MRSAR models are beat by a new Wold-based model [4].) Attempts to use intuitive texture features, like coarseness, contrast, and directionality [5] [6], are appropriate in some cases, but do not fully determine all the qualities people might use in judging similarity. Thus an a priori optimal context-dependent selection among similarity measures, either by human or computer, seems unlikely.

Next, the scope of queries that databases need to address is immense. Current computational solutions attempt to offer location of perceptual content ("find round, red objects") and objective content ("find pictures of people in Boston"). Desirable queries also extend to subjective content ("give me a scene of a romantic forest"), task-specific content ("I need something with open space, to place text"), collaborative content ("show me pictures children like"), and more [7]. Answering such queries requires a variety of features, or metadata, to be attached to the data in a digital library, some of which may not be computable directly from the data. The implication for algorithms is that they cannot rely on one model or one small set of carefully-picked features but will have to drink from a veritable "feature hydrant" from which only a few drops may be relevant for the query.

Finally, there is a significant need for semi-automated, versus fully automated, tools. Human-computer synergy can make ill-defined tasks manageable and has the power to overcome many of the problems of current pattern recognition tools. An important application of semi-automated tools is to assist the population of a database, viz. the creation of metadata. A crucial technical issue for such tools is the selection and combination of existing features: which features are most useful for a given query or annotation, how should they be combined, and which combinations are useful for the system to remember, so that it gets smarter with increased use? This last point is important since not only are the queries immensely variable, but the amount of training data (i.e. examples provided by a user of what they do and don't want) available at any instant is usually limited. Hence, a tool should strive to improve its generalization ability.

*This work was supported in part by BT, PLC, Hewlett-Packard Labs, and NEC.

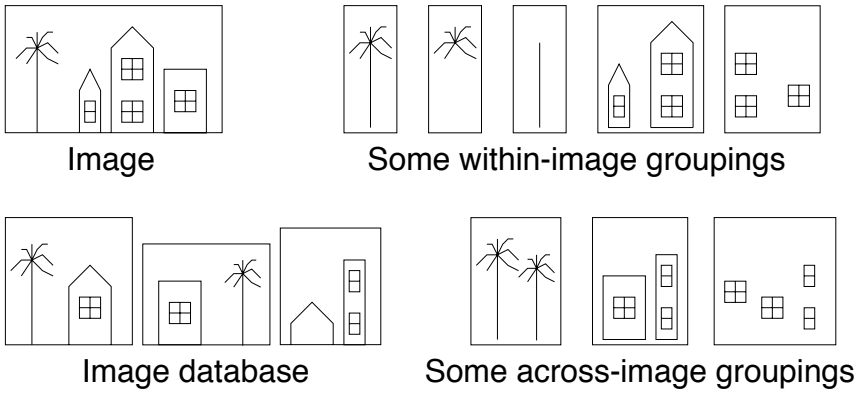

Figure 1: A basic task for image database retrieval and annotation tools, which is addressed in this paper: recovering useful within-image or across-image groupings. A grouping is just a set of related regions. Note that useful groupings generally cannot be captured by a single model, or even a single partition or hierarchy, and the similarity measure required to induce these groupings may be quite complex.

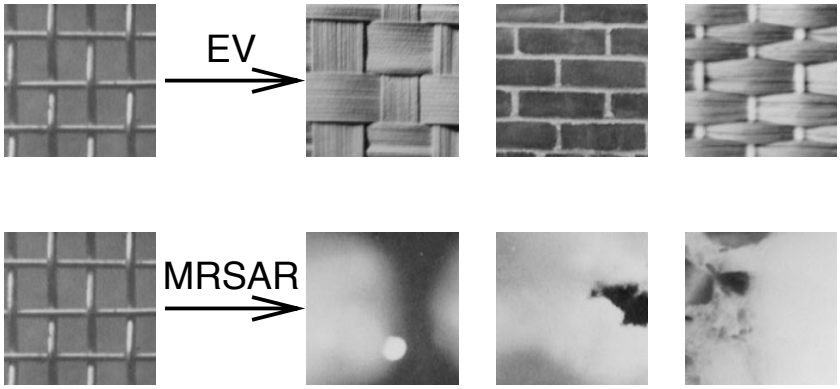

Figure 2: Data-dependent performance of texture models. The three patterns on the right are ordered by their similarity to the pattern on the left, given the particular model space EV or MRSAR. The MRSAR model, because it attempts to model fixed-size neighborhoods, misses the high-level structure that the EV model does not.

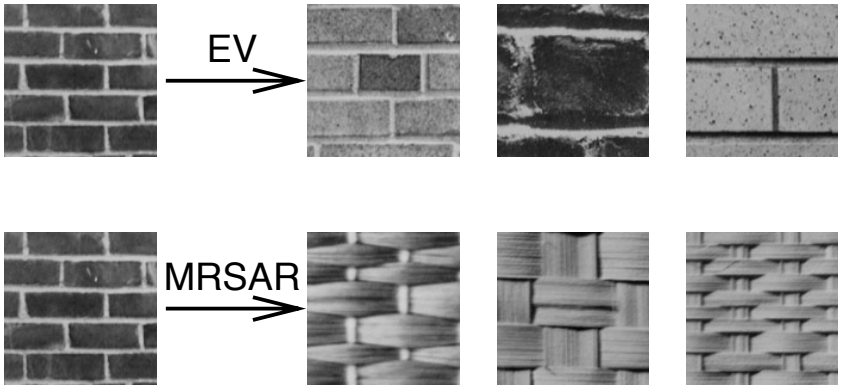

Figure 3: Task-dependent performance of texture models. The three patterns on the right are ordered by their similarity to the pattern on the left, given the particular model space EV or MRSAR. Both results capture the horizontal/vertical structure, but the EV returns a more semantically pleasing result since all images are bricks. However, these bricks are at different scales, and have different microtexture. Depending on the user's task, e.g. "find other images that look like bricks," the MRSAR result, or that of another model, may be preferable.

2 Multiple models

Dealing with these issues requires the use of multiple features, computed from the data or not, as well as ways to make informed, automatic selection of models and the features they describe. At this point in time, there seems to be no lack of specialized models, just a lack of knowing the best ways of utilizing them. Two well-known multiple model approaches are Bayesian combination and the rule-based blackboard, but this paper advocates a different approach which is more desirable for the interactive digital library setting.

2.1 Existing approaches

Bayesian combination for scene segmentation aims to represent images as a partition where the segment boundaries and interiors are represented by separate probabilistic models which are all estimated concurrently. Examples include the doubly-stochastic Markov random field (MRF) segmentations of [8] and [9], the auto-regressive model interiors and MRF model boundaries of [10], the Gaussian model interiors and active contour boundaries of [11], and the cooperative robust estimation of [12]. The basic idea of treating segment boundaries separately from their interiors is also at the heart of second-generation image coding techniques [13], where a variety of multiple-model strategies continue to be under investigation.

This joint optimization approach has an unfavorably large tradeoff of computation for accuracy. This is because it is highly susceptible to the combinatorial explosion of possible segmentations coupled with the possible models and their parameter assignments for each segment. Thus the research emphasis has been on sub-optimal iterative optimization algorithms, which often require assumptions on the number of regions and/or restrictions on the region interiors. The amount of approximations needed to make these work interactively (quickly and with little training) may defeat the benefits of using multiple models in the first place.

The rule-based blackboard for model selection has been advocated for "context-based vision" [14]. The method reduces the complexity of model selection via explicit, user-provided rules that determine when changes may be made to the blackboard (i.e. which models should be used at a given time) and what segmentation hypotheses should be removed from further consideration. This makes sure that only the most promising hypotheses are pursued and can conveniently return multiple segmentations of the scene along with their relative likelihoods.

A disadvantage of a rule-based method, while being computationally efficient, is that user-provided rules are expensive to produce, tend to be fragile, and are difficult to maintain when the rule set gets large. Rules are useful in limited domains, but these are crucial drawbacks for use in digital libraries supporting arbitrary data, features, and queries.

2.2 Proposed approach

The approach described in this paper allows many different models to be easily incorporated without the computational complexity that usually plagues multi-model methods. Like the rule-based blackboard, it tries to compile its decisions ahead of time, but instead of being manually given these decisions, it derives them directly from user interaction. The idea is to precompute many plausible groupings of the data, where groupings are induced by different models. Then the system selects and combines the groupings during user interaction. Relevance information, viz. which groupings were most useful, can then be fed back to modify these groupings or influence future grouping generation. In this way, the system is not only trained during individual example-based sessions with a user, but also trained across sessions to suit the tasks which it is asked to perform. This makes sure that the search space of groupings is always small but still contains desirable solutions.

An important optimization comes from the observation that when a reasonably large number of groupings is available, the correct groupings are usually present but are hard for the system to identify, given only a few training examples from the user. Therefore, the system can significantly improve itself just by changing the *relative weights* of groupings, not the groupings themselves. This optimization is realized by placing a separate weighting stage in between the generation and collection stages. Weighting does not change the size of the search space, but it does change the shape. The more detailed relevance information provided by the weighting stage can then serve to eventually modify groupings and grouping generation.

The three-stage method, illustrated in Figure 4, differs from conventional feature extraction and classification in three crucial ways. First, the feedback arc between the classifier and the features is performed by the computer, not the designer. This avoids the usual human cycle of trying lots of classification rules with lots of features, and trying to find the one combination that is best for the problem at hand. Second, each stage develops at different times and different rates, with the stages closest to the user changing fastest. This allows the computations to be distributed in time and space, facilitating interactive use and the incorporation of more complex models. This differs from Bayesian combination which essentially executes and adapts all stages at once, restricting the Bayesian approach to simple models for acceptable speed. Third, training is accumulated across sessions with the user, so that the system improves over time and can solve similar problems better, i.e. learn faster, the next time.

Like the other multiple model approaches mentioned above, this architecture is effective for a variety of classification tasks including within-image groupings, e.g., scene segmentation, and across-image groupings, e.g. locating similarly textured regions in a set of photos, or carving a path through an xyt volume of video.

This paper describes an interactive-time learning system, called "FourEyes," which assists a user in finding groupings both within and across images based on features from a society of models. The current implementation obtains groupings for still images from color models, texture models, and the user. For images from a sequence, optical flow groupings are also used. The grouping representation used by FourEyes allows for a variety of arbitrary models, and could easily be extended to include audio, text, or other data. However, the focus in this paper is on visual data.

3 User interface

The FourEyes interface (figure 12) is intended to allow selection of image regions without requiring the user to carefully outline the region of interest. The paradigm is similar to that of the perceptually organized editing program PerSketch [15].

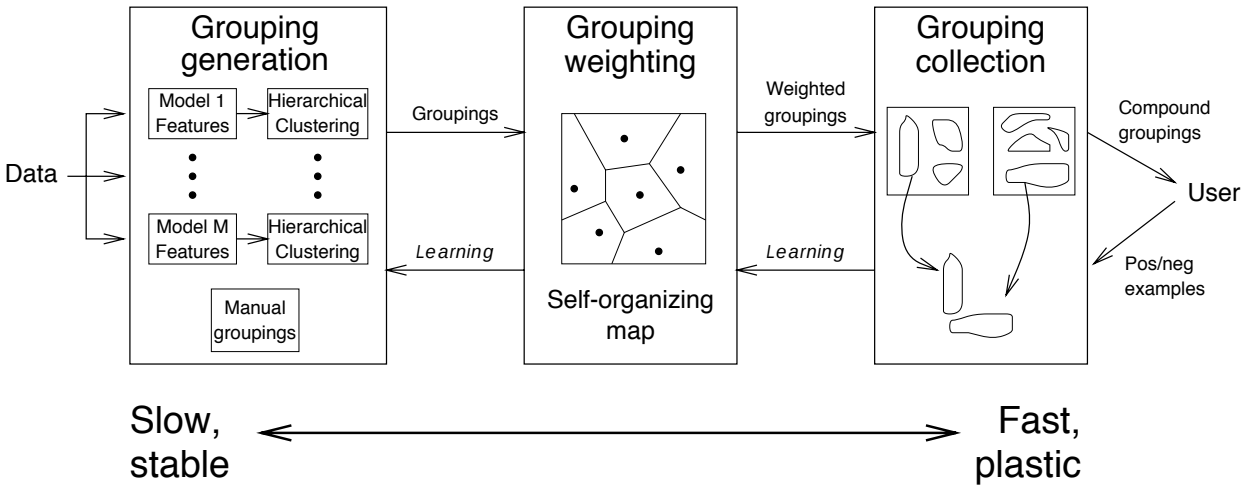

Figure 4: Interactive pattern recognition with a "society of models." The arrow at the bottom describes the rate at which the three stages learn.

Under this paradigm, the single object hierarchy of conventional paint programs is traded for multiple, possibly conflicting organizations. The amount of structure imposed by the system is mediated by an example-based interaction with the user. This makes image organization more like a process of discovery, for both the system and user. In PerSketch, the user can indicate the region of interest in a line drawing by making a gesture similar to its shape. In FourEyes, the user indicates the region of interest in an arbitrary color image by tapping pixels (mouse-clicking on them) or sweeping a path through the region. The touched pixels become positive examples which the system immediately attempts to generalize using its society of models (details below). Negative examples, i.e. pixels which are not in the interest region can be entered in the same fashion, but with a different mouse button.

It is important to allow a learning system a large space of possibilities (lots of models and features), and yet not allow it so much space that it cannot find a good solution in a reasonable amount of time. The key is the careful formulation of bias in the space of possibilities so that good solutions can be found in interactive time. To do this, FourEyes is provided with a large set of precompiled groupings of features from a society of models, along with a restricted method for choosing from or combining the groupings. Working in a carefully biased space, FourEyes can generate good generalizations of the user's selected regions in interactive time. The user continues the cycle of clicking on positive and negative examples until satisfied with all of the system's generalizations.

When the user is satisfied with the system's selection, FourEyes updates the weights of the groupings of which the selection is composed, as described in Section 6. This causes the groupings that were used to form the selection to be favored the next time a selection is made, so that in many cases only a single tap is needed to reselect a region or part of a region which has been operated on before.

FourEyes allows the attachment of a label to the selected region. This attachment is part of another example-based interaction: annotation of image regions throughout the database. As with segmentation, but this time across images instead of within them, precompiled groupings are selected and combined to extrapolate annotations. Attachment of a label currently adds a positive example for that label and also a negative example for all other labels. This exclusivity assumption, when correct, greatly reduces the total number of user examples needed to get a satisfactory labeling of a database. The attached labels could later be used to generate context-dependent semantic keys for querying and retrieving database contents.

There are other conceivable, but not yet implemented, operations on the selected region, besides labeling it. For example, paint tools could modify the color, move, or export a region, and database tools could retrieve similar regions (a special case of labeling) and paste them into the image.

The pattern recognition task for FourEyes is not to determine the "correct" model, "correct" grouping of database regions, or the "correct" segmentation of an image but rather

1. (Section 4) to contribute to a rich repertoire of reasonable groupings
2. (Section 5) to select from and/or combine these groupings to match an example set, with groupings from multiple models if necessary
3. (Section 6) to learn a weighting on groupings so that useful ones are recovered from few examples.

An appropriate performance metric is *the number of examples required before the user is satisfied with the response.* This assumes, of course, that not all possible generalizations from the user's examples are equally likely to be valid; otherwise no assistance could be provided. The challenge of FourEyes is to determine what the likelihood function actually is and to submit responses in accordance with it. An additional constraint is that this should all occur in interactive time. Since saving wall-clock time for database access is the objective, a system which processes 4 examples per second and requires 100 examples can be an order of magnitude better than a system which processes 4 examples per *minute* and requires 17 examples.

4 Generating groupings

A grouping is a set of image regions ("patches") which are associated in some way. The elements of a grouping may not necessarily come from the same image. This representation is useful since it admits different kinds of associations without adding complexity. For example, one set may represent "regions containing between 15% and 25% blue pixels" while another may represent "regions containing waterfalls" while yet another may represent "regions which were browsed very often this week." It also allows specific associations between patches to be weighted independently, since each set may have its own weight. This is important because, for example, lettering may be best grouped by shape whereas sky may be best grouped by brightness and location in the image.

Multiple hierarchies are used to contain the sets. Hierarchies allow efficient expression of sets which are the union of other sets and are the natural output of many clustering algorithms. The particular clustering algorithm used by FourEyes is based on shared neighbors [16]; it is a single-link method that tends to group areas of similar density in feature space. The method was chosen since it avoids the seemingly arbitrary cuts through regions of constant density made by complete-link methods, which try to minimize an aggregate, rather than local, error. This advantage of single-link clustering, which seems most appropriate for perceptual problems, has been demonstrated in the literature; see e.g. [17], [18]. In the experiments described here, k_t (the shared neighbor threshold) was zero and k (the number of neighbors) was steadily increased from 1 until all points formed a single cluster.

FourEyes computes within-image groupings from a model feature, such as color or texture, in three steps as illustrated in Figure 5. This is the first stage of Figure 4. This algorithm is used for its simplicity and generality and can easily be replaced by another grouping algorithm as better ones are developed.

1. A dense feature image is computed from the source image. Each point in the feature image is a feature vector (e.g. a histogram) computed from a neighborhood around the corresponding point in the source. For images in a sequence, the source image could be optical flow; otherwise it is the original color still. The feature image should ideally be at the same resolution as the source but may be coarser depending on computational constraints.
2. A coarse feature image is computed from the first one by computing a neighborhood average and covariance. This is the first step of segmentation, which performs local smoothing and obtains feature covariances for use of Mahalanobis similarity in the next step.
3. The coarse feature image is hierarchically clustered via the shared neighbor algorithm to produce within-image groupings. Note that the resulting groupings differ from those generated by traditional region-growing in that they can contain pixel patches that are not spatially adjacent.

The typical image size in our experiments is 512×512, with a coarse feature image of size 16×16. This size reduction significantly reduces the number of possible groupings, but still leaves 2^{256} to choose from (all subsets of 256 elements; the patches in a grouping need not be connected in the image). The result at this stage is a hierarchical set of image regions for each image, for each model. These may be used directly for segmentation, as well as for the next step: computation of across-image groupings.

Across-image groupings are computed from a hierarchical clustering of a feature measured over the within-image groupings. The within-image groupings need not have been generated by the same feature used for across-image grouping; they may have come from optical flow or even manual outlining. Even when using a single feature, the within-image groupings can use a variety of quantization sizes and arrangements, including individual pixels, not just the 16×16 tessellation used in this paper. In this way, many different scales and region shapes are allowed.

FourEyes is designed to not be contingent on the relevance of any one particular feature or segmentation algorithm. It can utilize groupings from another segmentation algorithm, which incorporates spatial relationships, edges, or a different sensitivity to scale. The within-image groupings simply provide information about which image regions should be usefully taken as a whole. For example, if a within-image grouping utilizes face detection to produce segments containing faces, the across-image grouping can use a face classifier. If the within-image groupings have different scales, it is up to the across-image features to remove scale dependence, if desired.

The advantage of incorporating within-image relationships for across-image annotation is described in [19]. For color-based annotation of image regions, that work demonstrated a clear quality improvement when scene-adaptive class thresholds, based on preserving the continuity of the within-image class-likelihood histogram, were used instead of fixed, universally optimized thresholds. FourEyes approximates this behavior by forming its across-image groupings from within-image groupings. Moreover, the shared neighbor clustering algorithm used by FourEyes behaves similarly to the histogram splitting used in [19], so the within-image groupings generated by both methods similarly preserve class-likelihood continuity. This is a major difference with our previous annotation system [20], which did not use within-image groupings. Another difference is the ability to learn weights on groupings and to self-improve, as described in Section 6.

The within-image and across-image groupings are computed off-line, before the user begins interaction with the system. This separation of functionality is important for practical implementation in a real image database retrieval system. For example, when clustering happens off-line, it can perform extensive cross-validation, noise sensitivity, and stability checks, possibly utilizing several different algorithms. This level of evaluation is currently infeasible for on-line use, but the off-line use allows state-of-the art results from pattern recognition to be incorporated, improving the overall system performance. Feature extraction routines, since they run off-line, can likewise use larger neighborhoods, more accurate estimators, and have more diversity. New feature extraction or clustering methods can be developed independently of work on the other components. Such engineering concerns are important to those who would construct real systems.

A disadvantage of precomputing groupings is that these must be recomputed when a novel image is added to the data set. In FourEyes, this means a full reclustering for all of the features, or some sub-optimal "patching in" of the novel image. However, since queries occur orders of magnitude

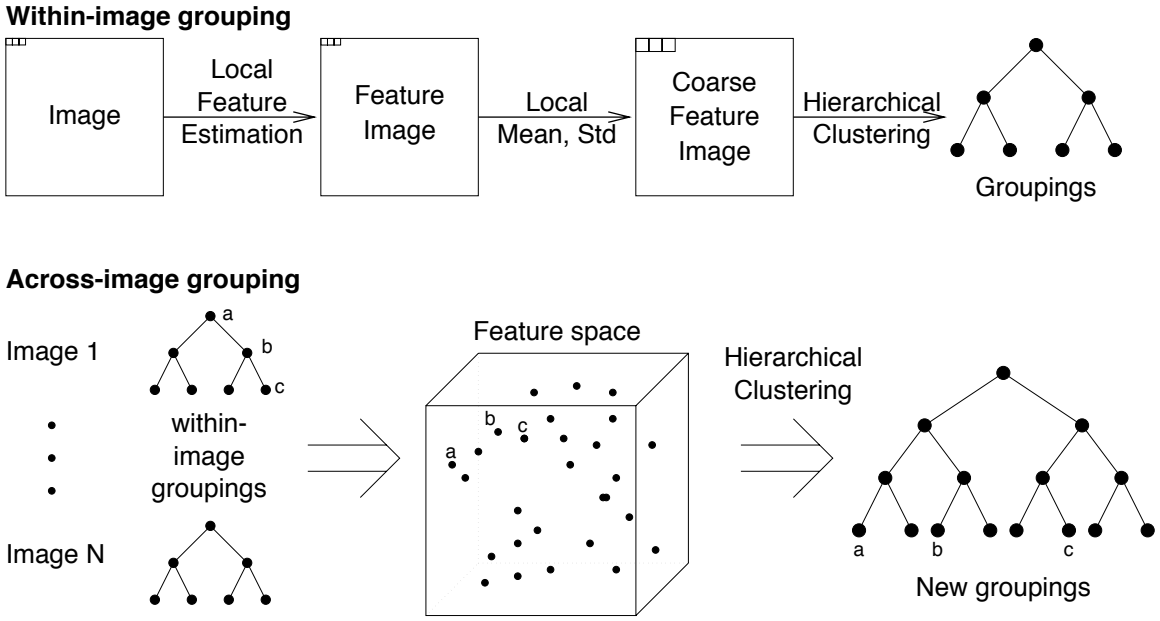

Figure 5: Computing within-image and across-image groupings. In image 1, grouping a contains b, which contains c; e.g. they might be house, door of the house, window on the door. When projected into feature space, they are considered individually, and look different. The resulting clustering says that a looks more like b than c.

more often than additions to the database, the interactive speedup can offset the recomputation cost. Groupings which are not computed, i.e, do not come from parametric model features, but might come from human specification, must be manually modified when novel images are added.

The version of FourEyes described here does not recompute groupings automatically during interaction; this need is alleviated by the weighting and selection mechanisms. For example, if the set of groupings is sensitive to clustering or model parameters, then multiple sets of groupings can be used, with different choices of these parameters, just as though these were different models. Adding extra models to the society of models paradigm does not cause the same combinatorial explosion of possibilities that it would ordinarily cause in the Bayesian combination paradigm mentioned in Section 2. The later stages can automatically determine which groupings were actually useful and exclude those which were not. In this sense, FourEyes can also help learn which models are of greatest use for a given problem and set of data. In a later version of the program, a background task continuously eliminates groupings with low weight (a forgetting mechanism) and replaces them with new ones. This adds a link from the second stage to the first and is described in [21].

Since the later stages of the system only see groupings, not feature values, it is not necessary for numerical similarity features to be used. For example, this is advantageous for incorporating subjective associations among content. For humans, it is often easier to specify groupings of image regions than to attach meaningful and consistent attributes to them.

5 Collecting groupings

Once a set of groupings has been formed, the next task is to select or combine these to form compound groupings for the user. This is the third stage of Figure 4, referred to below as "the learner." At every point in the interaction, the learner must try to generalize from a set of examples provided by the user. The result is a set of image regions which contains all of the positive examples, and none of the negative. This set is formed from multiple groupings and so is called a compound grouping.

In the terminology of the machine learning literature, the compound grouping that the learner is searching for is a "concept" which is consistent with the examples, i.e. includes all positives and no negatives. The performance of any learner is crucially dependent on its *inductive bias:* "any basis for choosing one generalization over another, other than strict consistency with the observed training examples" [22]. Bias is determined by both the extent of a learner's concept space as well as the relative weights assigned a priori to different concepts. The latter has a close correspondence with the prior in Bayesian learning [23]. These two components of bias may be expressed procedurally (by an algorithm) or declaratively (say, by weights). Either may change during the problem or across different problems.

The approach taken in FourEyes is to use a simple concept language (pure disjunctions, i.e. set union) with an adaptive weighting mechanism. This makes a great deal of the inductive bias declarative and hence easy to change dynamically (i.e. the learner is "malleable"). This is in contrast to a learner with a powerful concept language but limited weighting mechanism, such as ID3 [24] or CART [25], which can simulate arbitrary set operations but can only change their bias via splitting or pruning parameters, and so are difficult

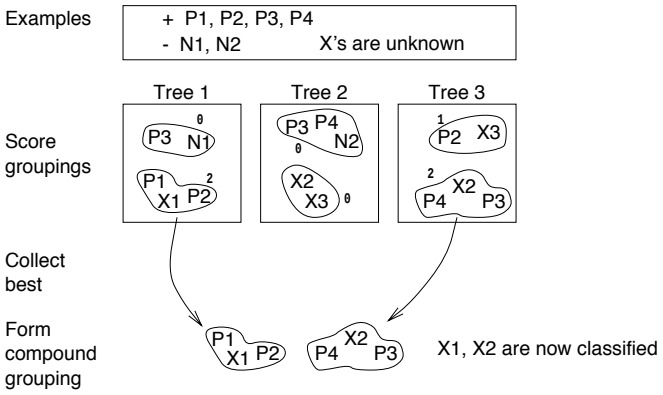

Figure 6: Collecting groupings

to steer in desired directions.

The learning algorithm used in FourEyes descends from AQ [26]. AQ is a greedy method that collects groupings one at a time, such that each one includes no negative examples but their union includes all positive examples. Starting from an empty union, the grouping which adds the most positive examples but no negative ones is iteratively added. Since the hierarchies generated in the first stage include the smallest-scale patches at the leaves, this algorithm can always satisfy any set of examples, no matter how arbitrary.

The algorithm used in FourEyes differs from AQ in its evaluation of the next grouping to add. Instead of choosing the grouping which simply maximizes the number of positive examples (as in our previous work [20]), it maximizes the product of this number and the *prior weight* of the grouping. This means that, e.g., a grouping with twice the prior weight can cover half as many positive examples before it is chosen. Thus the prior weights directly influence the learner's inductive bias. The prior weights are determined from statistics collected over multiple learning sessions, which will be described in Section 6.

Figure 7 graphs the performance of the learning algorithm for learning texture classes in the Brodatz [2] album. Each of the 112 textures in the album was equally divided into 9 128x128 non-overlapping images; the desired classification corresponds to the 112 original texture classes. The learner begins with all images unclassified. The learner was trained by repeatedly querying it for the classes of all images, tallying the errors, choosing an erroneously labeled image at random, and then informing the learner of the proper class of that image. The learner only gets examples which will be relevant, because it has erred on them, instead of getting an arbitrary off-line selection of examples. This incremental presentation of examples is similar to the way training would occur with a user (who sequentially selects one or more of the 128×128 images as positive or negative examples) but is different from conventional pattern recognition, where classification is done by comparing to heavily pretrained prototypes or feature distributions, without on-line feedback.

At each step, an image which was unclassified by the learner was scored as one error; an image which was misclassified by the learner was scored as two errors, to make blind guessing disadvantageous. Since the learner never forgets the examples it is given and it assumes that classes are disjoint, it always converges to zero error in at most 1008 steps; the objective is to get it to converge considerably faster. The minimum number of examples required is 112, corresponding to exactly one image from each class, since the learner does not know how many classes there are and cannot guess the names of unseen classes. Even though there is a random element in the training algorithm, the error traces vary little over repeated executions (no more than 2%). Getting the traces to improve over time will be handled in the next section.

Four experiments are shown in Figure 7, each with an equal prior weight for all groupings. The first experiment provides a baseline; the learner only had available a single randomly-generated hierarchy. This hierarchy had 632 groupings containing more than one element. Given this feeble bias, it required all 1008 examples to reach zero error. The second experiment had available the same hierarchy plus a hierarchy generated by clustering the images by EV features [27] (the hierarchy contained 427 groupings). This extra bias let the learner reach 100% accuracy after 699 examples. (The random hierarchy served as "grouping noise," meaning irrelevant groupings, which are to be expected in digital libraries.) The third experiment added another hierarchy, this time generated by clustering the images by MRSAR (the hierarchy contained 255 groupings). The MRSAR has demonstrated excellent matching performance on this database in earlier experiments [3], so we would expect learning to proceed even faster. This was indeed the case; the learner reached 100% accuracy after 487 examples. The MRSAR is so clearly superior that the behavior was identical when both the randomly-generated and the EV-generated hierarchies are left out, i.e. these two now play the role of "grouping noise." This case also illustrates the use of FourEyes to identify a model which is best suited to a problem. The fourth experiment added an "Ideal" hierarchy which explicitly contained the desired 112 classes as groupings (148 groupings total), bringing the total number of hierarchies to four and the total number of groupings to 1462. The learner quickly exploited this extra knowledge, reaching 100% accuracy after 303 examples. To actually get zero error with the minimum number of examples (112), the learner would have to either have been given the correct 112 groupings and no others, or been given a prior weighting which favors these groupings over the others. The latter case is examined in Section 6; the former case could arise through adaptation of the grouping generation stage, as explored in [21].

The dominance of some models over others is obvious in these four experiments, but it need not be so in general. For example, if two roughly equally performing models, say a Euclidean gray-level histogram distance and the tree-structured wavelet (TSW) transform [28] are used, the result is better than either one alone (from 916 and 858, respectively, to 785 examples to reach zero error).

These experiments demonstrate the ability of the learner to tolerate grouping noise and quickly locate the most useful groupings for generalization. Adding more random or inferior groupings does not substantially affect the results described here, until a significant fraction of all possible groupings are accounted for. At that point, the learner has too many options (i.e. too little bias) and so, with equal weights on groupings, can do no better than random guessing.

When using all 1462 groupings, the learner processed over 20 examples per CPU second; it has been benchmarked with

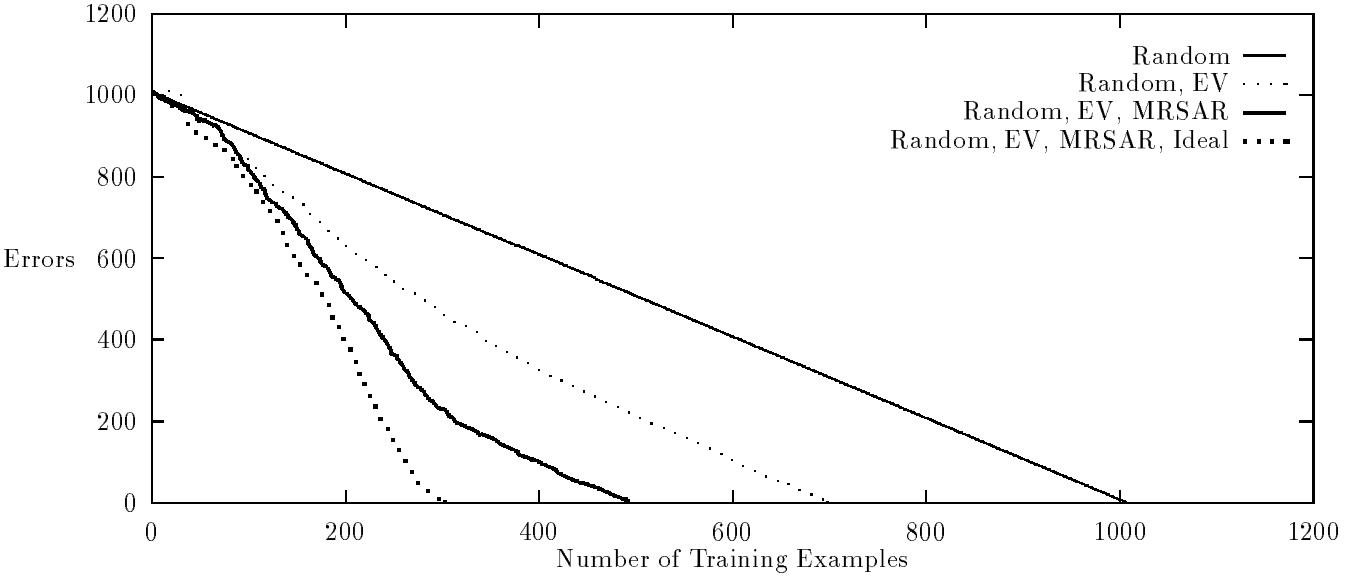

Figure 7: Learning performance for different sets of groupings. The faster the curve drops, the better the performance.

up to 120,000 groupings on 6400 patches, where it could still process up to 5 examples per CPU second on an HP 735/99 workstation. The time complexity for constructing a compound grouping from an example set is linear in the number of examples, the number of trees, and the *height* of each tree; it is not dependent on the total number of groupings or the total number or size of patches, when suitable hashing schemes are employed. The time complexity for retrieving all the patches in a compound grouping is linear in the size of that grouping.

6 Weighting groupings

As described in Section 5, the learner tries to find the best compound grouping according to consistency with the user's examples and an inductive bias. When the number of examples is large, consistency alone can serve to isolate good groupings. In such a case, the need for bias is low; many so-called nonparametric learning algorithms exploit this phenomenon by requiring little knowledge of the problem but many training examples. However, the low-bias approach is not suitable for user interaction since each example is expensive in terms of the user's time. When the number of examples is small, many groupings will be consistent; consequently, the bias is crucial in determining which groupings are chosen.

FourEyes solves the biasing problem by giving the learner adaptive prior weights which change between interactions with the user, so that the groupings which were satisfactory this time will be selected earlier (i.e. with fewer examples) next time. If instead of this solution, just one vector of grouping weights is used and updated, over time the components will average. This is because each task has its own "best" weight-vector; each of these will pull in a different direction and they will cancel each other out. The multiple weight-vectors we use avoid this problem; each one can specialize on and be trained on tasks in a particular region of weight-space, as shown in Figure 8. As the system interacts with the user, it can determine which weight-vector is most relevant and then use it for learning. When the interaction is complete, the chosen weight-vector is updated. This way the learner can adapt to many different tasks without blurring its experience.

Selecting prior weights after seeing some training data corresponds to learning by analogy with previous problems. Since it allows faster convergence to plausible groupings, making an analogy gives the learner more training examples for the current problem. It does this not by carrying over the literal training examples from a single previous problem, but rather carrying over the agglomerative characteristics of the training examples from a set of previous problems. An important issue here is the comparison between weight-vectors in order to determine when two learning tasks are similar; this is $s(b)$ given below.

6.1 Modeling weight-space

FourEyes classifies learning problems by clustering weight-space. Currently this is done via a self-organizing map (SOM) [29]. During user interaction, each SOM unit (stored vector of weights) competes for consistency with the user's examples; the winning unit propagates its weights over the groupings. When the user is satisfied with the output of the learner, the winning unit is updated to more closely match the examples. In this way, the SOM defines a clustering of the weight-vectors for the problems it has seen, where each SOM unit is a cluster center. Note that a self-organizing map is typically used for the classification of feature vectors in a learning problem; here it is being used for classifying learning problems themselves, in terms of the grouping weights they favor. Each SOM unit then represents a prototypical learning problem.

Each SOM unit stores statistics about how often certain patches appear as positive or negative examples. Specifically,

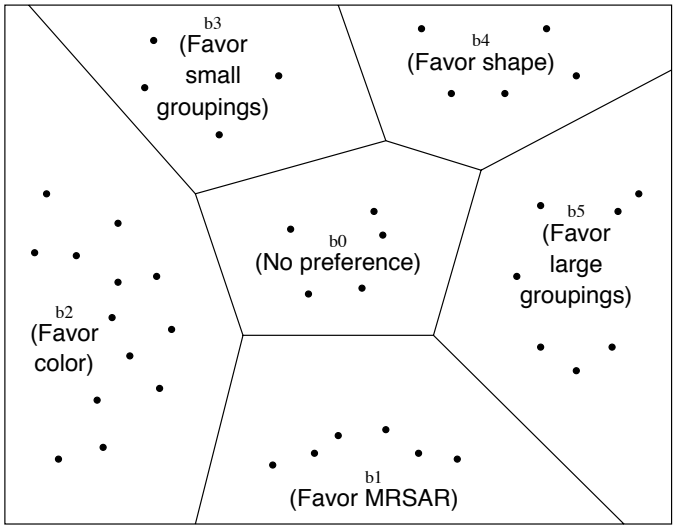

Figure 8: Hypothetical weight-space for learning. Each point is a vector of weights for all of the groupings. The optimal weights for different problem domains will fall into distinguished regions. These regions can be approximated by the Voronoi cells (in bold) of units in a self-organizing map, which clusters all of the points it sees. A unit which "favors color" weights most highly those groupings which come from a particular color model, from a combination of color models, or from non-color models that happen to be consistent with color.

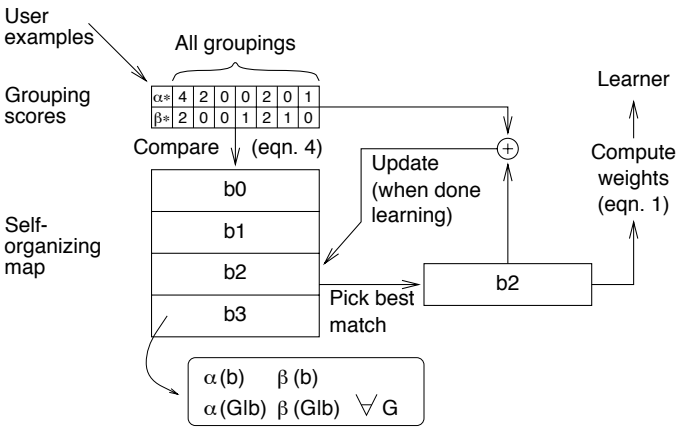

Figure 9: Self-organizing map used for modeling weight-space.

each unit b stores:

- $\alpha(b)$ = the number of positive examples contributed to b ("contribute" defined below)
- $\alpha(G|b)$ = the number of positive examples from $\alpha(b)$ contained in grouping G
- $\beta(b)$ = the number of negative examples contributed to b
- $\beta(G|b)$ = the number of negative examples from $\beta(b)$ contained in grouping G

When a unit is updated, the total set of examples received from the user contributes, i.e. adds, to these values. Note that $\sum_G \alpha(G|b)$ is not necessarily equal to $\alpha(b)$, since groupings may overlap. These values are used for both selecting the appropriate unit and determining the prior weights for groupings, once a unit has been chosen. Since the number of groupings can be quite large, the number of values each unit must store can get prohibitive. In such cases, a sparse vector representation can be used, since many of the example counts will be close to zero.

First will be described the formula for prior weight, once a unit has been chosen. The prior weight on a grouping, as used by the learner, is intended to be a heuristic measure of its expected contribution toward the learning goal. Let P be the hypothetical set of patches (or one such set) which if returned to the user would be satisfactory; let N be its complement. Then the learning goal is to cover all of P but none of N, given only a few examples from each. The heuristic used in FourEyes for the prior weight $w \in [0,1]$ of a grouping G given weighting unit b is

$$w(G|b) = \frac{\alpha(G|b)+1}{\alpha(b)+2}\left(1 - \frac{\beta(G|b)+1}{\beta(b)+2}\right) \qquad (1)$$

The first term of $w(G|b)$ is an estimate of the expected fraction of P contained in G and the second term is an estimate of the expected fraction of N *not* contained in G. The offsets provide non-singular initial conditions; since exactly one unit is in effect at any time, only the relative weights of groupings are significant. This heuristic formula for w is not proposed to be optimal in all cases, but has proven better in our experiments than several alternative formulations.

Units in the SOM are chosen by maximizing the match value $s(b)$ between a unit b and the current set of examples. Thus $s(b)$ corresponds to the notion of problem similarity in making analogies. Define:

- α^* = the number of positive examples provided by the user
- $\alpha^*(G)$ = the number of positive examples from α^* contained in G
- β^* = the number of negative examples provided by the user
- $\beta^*(G)$ = the number of negative examples from β^* contained in G

Finding the best unit corresponds to finding the best match between $\alpha^*(G)$ and $\alpha(G|b)$ (or $\beta^*(G)$ and $\beta(G|b)$) over all groupings G and all units b. A normalized correlation, i.e. weight-vector angle cosine, is a logical choice for similarity but only after some modification. This is because not all groupings G should be considered equally for matching; e.g. a grouping which has equal counts for α^* and β^* is not characteristic of the user's examples and so should be ignored.

Therefore the normalized correlation is done between $r_1(G)$ and $r_2(G|b)$, two measures of the relevance of a grouping (clipped to zero if negative):

$$r_1(G) = \frac{\alpha^*(G)}{\alpha^*} - \frac{\beta^*(G)}{\beta^*} \tag{2}$$

$$r_2(G|b) = \frac{\alpha(G|b)+1}{\alpha(b)+2} \div \frac{\beta(G|b)+1}{\beta(b)+2} \tag{3}$$

$$s(b) = \frac{\sum_G r_1(G) r_2(G|b)}{\sum_G r_1(G) \sum_G r_2(G|b)} \tag{4}$$

Here is some intuition why r_2 should differ from w. When the SOM is searching for the weights to use (using r_2), it should be picky about details, and pay close attention to negative examples. After the learner has decided on weights and is utilizing them (with w), it should have faith in its choice, and pay more attention to positive examples. This is why r_2 divides by the negative example ratio, making it more sensitive to negative examples than w is. Alternative arrangements, e.g. swapping r_2 and w or making them the same, degrade performance in our experiments.

New units are added via the following method. Initially, only one unit is present: a special immutable unit containing a flat weighting. If this unit is the winner then this means none of the available weightings are appropriate. In this case, a new unit is created and initialized with the current example counts (α gets α^*, β gets β^*). A method for adding new units which avoids monopoly, e.g. "wincount" [30], could also be used. Another possible extension is the relaxation of the winner-take-all constraint, to allow multiple units to contribute and/or be updated, e.g. via a neighborhood around each unit [29], which would provide output interpolation. A mechanism for the elimination of unnecessary units (forgetting) may also be useful. These are the incremental analogs of merge/split rules in batch clustering algorithms.

6.2 Learning speedup

The learning speedup provided by using a SOM of grouping weights is demonstrated in the following three experiments. The learner described in Section 5 was modified in two ways:

1. After every e examples received, the SOM was consulted for each class to provide a prior weight to be used when selecting groupings for that class. The choice of e is a time/accuracy tradeoff, since SOM lookups are expensive; the experiments used $e = 10$.
2. When the learner was signaled that the learning task was completed, for each class it updated the SOM unit whose prior weight was selected for that class.

In the first test of learning speed-up, the Brodatz classification task was repeated. The learning curves on the second run for the same classification problem are shown in Figure 10. Except for the random hierarchy alone, all curves reduced their learning time by about 160 examples. On the first run, the SOM was empty (except for the special flat weighting). After the first run, the number of units created in the SOM was 112; each class obtained its own section of weight-space. On the next and later runs, the SOM eventually matched up each class with the proper unit, without creating new ones. Even though estimates of class statistics continued to improve in the SOM, the learning performance did not improve significantly after the second run; the learner reached its peak early, since there was only one problem to learn about. Since exactly the same classification was desired both times, this test should be viewed as the best learning improvement that can be expected by only changing weights on groupings. Equipped with the ideal hierarchy and a SOM with the appropriate weights, the learner almost reached the theoretical optimum of 112.

Notice that the curve for the appropriately biased "Random, EV" learner is better than that of the weakly biased "Random, EV, MRSAR" learner, shown in Figure 7. This illustrates that weighting existing groupings effectively can be better than having more groupings available, even groupings from a "better" model such as the MRSAR. Good models are just one component of a good classifier.

Next, the learner's performance was measured when applied to similar classification problems instead of the same problem. Three categories of similar problems can be distinguished:

1. Problem A's classes are unions of Problem B's classes. (For example, B discriminates between red blocks, green blocks, red balls, and green balls; A discriminates between red and green only.)
2. Problem A's classes are partitions of Problem B's classes. (The reverse scenario.)
3. Problem A's classes are unions of partitions of Problem B's classes. (An all-encompassing transformation.)

Tests were made for the first two cases, starting with the 112-class problem, by randomly pairing up all classes and then uniting pairs. Successive application of this rule produced a 56-class, 28-class, and 14-class problem, so that, e.g. the 14 classes are unions of pairs of the 28 classes. Then each problem was run with a SOM trained on a single run on another problem. The number of examples until zero error, for each of these combinations, using the MRSAR hierarchy is shown in Table 1. The behavior is similar for other hierarchies, though the numbers are larger. The important characteristics of this table, revealed along the diagonal and off-diagonals, indicate (1) some training is always better than none, (2) the more similar the problems, the better the speedup, and (3) speedup is better when trained on a problem with fewer classes than the current problem (lower left diagonal of the table).

The latter observation means that when training on "A" and testing on "B," the SOM is better at case 1 above than cases 2 or 3. This is probably because of the winner-take-all rule; exactly one stored weight-vector can be used per class. The learner generally gets more information when these weights were trained on a class which is a superset of the desired class than a class which is a subset of the desired class. This means it is better at learning "apple" given weights for "fruit" than vice versa. This imbalance might be avoided by, for example, generating the weights from a combination of the k best matching SOM units where $k > 1$. Then SOM units trained on "apple," "orange," and "banana" could all contribute to learning "fruit."

Finally, the SOM's ability to retain simultaneous knowledge of different problems was tested. Ten classification problems were created, each one constructed from 14 randomly chosen unions of 8 of the 112 Brodatz classes. Thus each problem had 14 disjoint classes over the 1008 images. Each of the problems, while having subsets in common, differed greatly in how these are arranged and so fell under similarity

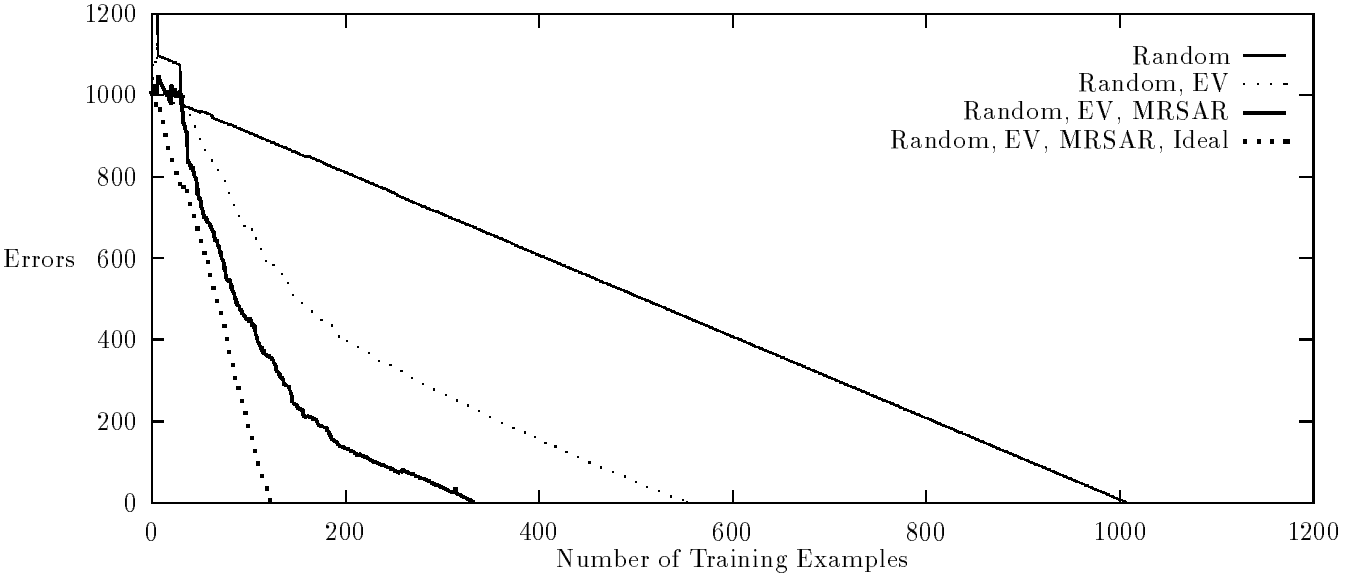

Figure 10: Learning performance for the task in Figure 7, on the second run.

	Trained on				
Run on	14	28	56	112	none
14	357	463	469	477	483
28	362	360	457	472	483
56	377	376	376	457	489
112	402	403	388	369	493

Table 1: Learning performance for similar problems; note the behavior along the diagonal and off-diagonals.

case 3 above. In general, training the SOM on one of the ten problems offered no assistance for another problem, i.e. the number of examples to reach zero error was effectively unchanged. Instead, the SOM was trained on each problem in turn and then re-run on each problem again, consecutively. Since the problems were reasonably independent, different sets of weightings would likely be needed for each one; hence, this tests the memory's ability to model weight-space.

Figure 11 shows the number of examples until zero error for two passes made consecutively through the ten problems, compared to the "optimum" result when the SOM is trained specifically for each problem. After one pass from left to right, the SOM automatically grew to 27 units by the end (this number is order-dependent, as in most self-organizing clustering algorithms). As can be seen in the graph, memory from the first pass was good enough to get most of the way to the optimum on the second pass. Successive iterations did not add any more units to the SOM or alter performance beyond 5%.

7 Performance on natural scenes

The performance of FourEyes in a realistic situation was measured by its labeling performance on the natural scenes in the "BT images." In these images, the regions are of irregular shapes and sizes, and contain many different scales and inhomogeneous textures. Three human subjects were asked to freehand outline regions in 25 of the natural scenes and assign the seven labels "building," "car," "grass," "leaves," "person," "sky," and "water" to them. They were not asked to make precise boundaries or make decisions on a strictly perceptual basis (both of which would have aided FourEyes). Then a majority vote among the subjects was used to derive a single, approved ground-truth segmentation and labeling of those images. Since within-image groupings were computed using a 16×16 tessellation, the ground-truth segmentations were quantized to that resolution. Note that finer tessellation-sizes could be used, or overlapping tessellations, or even single pixels, but that this level of detail is usually not necessary for tasks such as retrieval. Finer resolutions, or even different resolutions for each model, can be used without change to the framework here if the application requires them. The resulting ground-truth is shown in Figures 13 and 14.

Given this ground-truth, we could present it all at once to the computer, as is done in traditional pattern recognition in a training phase. However, the goal is to benchmark the learner as if it were being used by a person, incrementally picking 32×32-pixel patches of regions of interest. This is a more realistic scenario for database retrieval and annotation, where the user gradually decides what he or she wants while browsing the data. However, it tends to make the problem harder in that there is very little training data in the beginning, and yet the system has to use what's available and learn continuously.

Four experiments were conducted with different sets of groupings available to the learner. Patch size varied in the groupings computed by stage 1, but the results in Table 2 are given in terms of 32×32-pixel patches only. There were 4546 labeled 32×32-pixel patches and 7 classes so these are the theoretical maximum and minimum numbers of examples

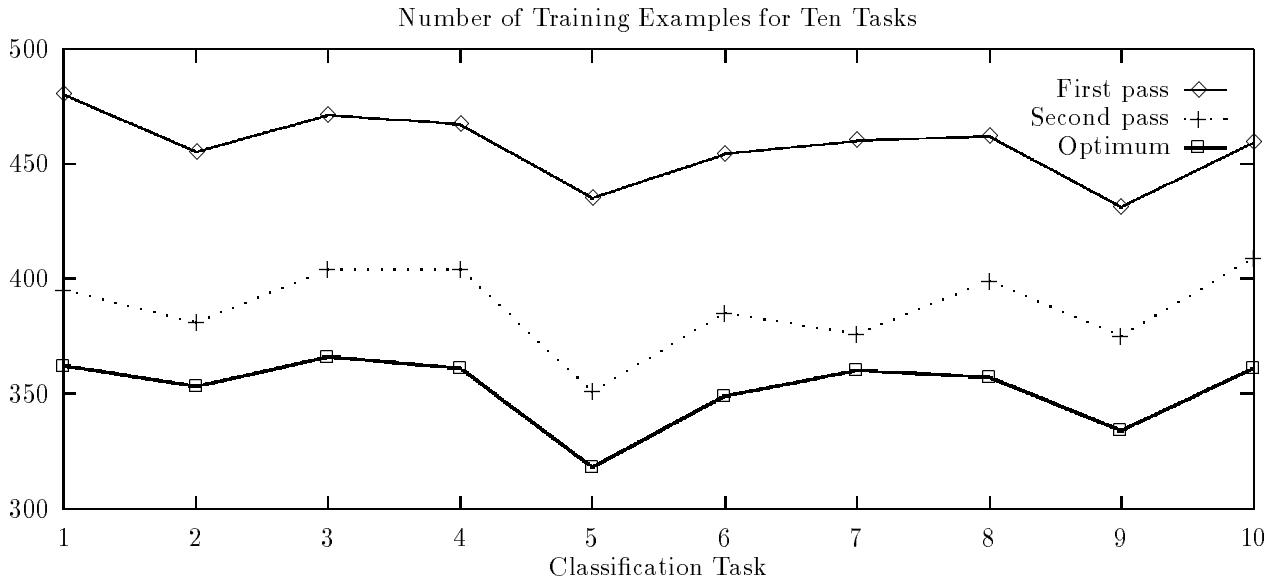

Figure 11: Simultaneous learning performance for ten different problems

Groupings	Zero error	25% error
8×8	1.6	1.8
plus MRSAR	2.0	3.3
plus Ohta	2.4	5.7
second run	2.9	8.8
plus human	2.6	20
second run	2.9	28
plus ideal	38	47
second run	69	107

Table 2: Annotation savings for natural scenes. Numbers are the ratio between the total number of correctly labeled 32×32-pixel patches (4546 for zero error, 3410 for 25% error) and the number of examples. The higher the ratio, the more help the system is to the user.

required to reach zero error. The baseline experiment (row 1 in Table 2) used a set of 1600 groupings corresponding to an 8×8 tessellation of each image, i.e. into groups of four 32×32-pixel patches. This corresponds to a simple bias toward giving nearby patches the same label. It required 2924 examples to reach zero error, for an annotation savings of 1.6:1. Next, within-image groupings computed from the MRSAR texture feature over 64×64-pixel patches were added (1740 groupings, or about 70 per image), which allowed the system to achieve a savings of 2:1. Third, within-image groupings computed from the Euclidean distance between unnormalized histograms of 32×32-pixel patches in the Ohta color space [31] were added (1663 groupings), which raised the savings to 2.4:1. When run again on the same problem, the weights stored in the SOM raised the savings to 2.9:1, which is therefore the most that can be expected with these two models. The learning curves exhibited diminishing returns after reaching 25% error; the last experiment spent 75% of its examples after this point. This indicates that the system is most effective at getting a quick first-cut labeling rather than a perfect labeling.

Interestingly, adding across-image groupings computed from the MRSAR or Ohta histogram features did not improve performance. This indicates that the across-image perceptual variations in this data's semantic classes were high enough to confuse these image models. Another cause might be the scale-sensitivity of these particular across-image features.

The human-provided labelings were quite semantic and seem difficult to capture only with local feature measurements and no common-sense knowledge. Therefore, the final test added human-provided within-image groupings to the first stage of FourEyes. This test would correspond to the system forming new groupings to better match that person's preferences. The new groupings were provided by one of the sets from which the ground-truth was derived, but deliberately did not match exactly the ground-truth used in our tests. This raised the zero-error annotation savings slightly and allowed the learner to reach 25% error much faster. The factor of 20 savings while descending to 25% error but relatively low savings for zero error indicates that the human-provided groupings were almost right but had to eventually be rejected as they could not perfectly match the ground-truth regions. An alternative grouping combination rule, which allowed more than just disjunctions, or was softer, could alleviate the need for each grouping to be a subset of a desired grouping, and improve performance in this case. If the correct within-image groupings were added (an ideal situation, the last two rows) the learner improved itself by an order of magnitude. The learner could approach the theoretical limit of 7 examples or 650:1 savings if ideal across-image groupings also became available, or were learned.

8 Related work

Some recent systems which perform retrieval on image data are QBIC [6], SWIM [32], Photobook [33], and CORE [34]. A notable quality of these systems is that they present many different ways of organizing the data but offer little assistance in actually choosing one of these organizations or making a

new one. Users are often forced to determine what features will be relevant to their intent, if any, instead of addressing their intent directly. Since intentions can vary greatly and features can be very opaque, another solution is needed. The example-based interaction in FourEyes, coupled with a learning element that selects and constructs organizations, provides such an alternative.

The need for a learning component between the user and image features is described in [35]. In that work, positive and negative pixels were used to define a classification rule for new pixels. The classification rule was a conjunction of thresholds on one-dimensional feature values, where the thresholds and features are chosen to maximize the separation between positive and negative. FourEyes differs from that work in three important ways. First, FourEyes does not perform its analysis strictly on lone pixels. By using within-image groupings as the analysis elements, it addresses the need for spatial context as outlined in [19]. Second, FourEyes can incorporate information from multi-dimensional or non-numerical features such as subjective clusterings provided by the user. Third, and most important as the number of features gets large, FourEyes can learn a strong bias on groupings. FourEyes' groupings implicitly quantize and the weightings prioritize the thresholds used in [35]. This allows FourEyes to improve its performance over time and over new problems, despite growth in the number of features.

FourEyes employs hierarchically-organized sets, produced by off-line clustering, for efficient retrieval of plausible groupings. A possible alternative is the hierarchical self-organizing map discussed in [36], which can reduce high-dimensional vector spaces into arbitrary hierarchical topologies (a hierarchy of two dimensional topologies was used in that paper). The principal advantage of the algorithm is that it is trained on-line and might be modified to optimize a classification criterion, as in LVQ [37]. This admits the possibility of modifying the groupings based on information obtained by the learner and the memory of weights, without a full reclustering step. Using a SOM to represent groupings could unify the implementation of the first two stages of the system and perhaps even the third.

9 Summary

The "FourEyes" learning system for assisting users in digital library segmentation, retrieval, and annotation, has been described. Digital library access requires the use of many context-dependent or noisy features, whose relevance is not always obvious. FourEyes addresses this problem on multiple fronts:

1. It first makes tentative organizations of the data, in the form of groupings. The grouping representation provides a common language for different measures of similarity. Groupings can be manually provided, induced by color/texture models, derived from optical flow information, etc. FourEyes uses both within-image groupings and across-image groupings composed of these.

2. The user no longer has to choose features or set feature control knobs. Instead, the user provides positive and negative examples which allow FourEyes to choose groupings (hence, similarity measures) automatically. The interaction is more like a conversation where both parties give each other prompt and relevant feedback in order to resolve ambiguities.

3. With many groupings to choose from, the number of examples required to isolate good groupings can get large. FourEyes circumvents this by having *prior weights* on the groupings and preferring groupings with more weight. These weights are learned across interactions with users, so that the system gets better, i.e. learns faster, from repeated use.

4. Since the optimal weights on groupings changes with context, FourEyes employs a self-organizing map to remember useful weight settings. As the user interacts with it, FourEyes chooses the most appropriate weights in the map. This way, FourEyes can improve its joint performance on a wide range of tasks.

5. FourEyes offers a practical way to get interactive performance, by explicitly separating these grouping generation, weighting, and collection stages. It does this without sacrificing adaptability or the use of multiple models, because feedback between the stages allows the whole system to learn, though each stage at a different rate.

10 Software

All three stages of FourEyes, plus the image database management, were written in C and Tcl and run on Unix machines. The first stage is a collection of off-line feature computation and clustering programs to which new programs can be easily added. A copy of FourEyes for educational or research purposes can be obtained by contacting the authors.

References

[1] J. Mao and A. K. Jain, "Texture classification and segmentation using multiresolution simultaneous autoregressive models," *Patt. Rec.*, vol. 25, no. 2, pp. 173–188, 1992.

[2] P. Brodatz, *Textures: A Photographic Album for Artists and Designers.* New York: Dover, 1966.

[3] R. W. Picard, T. Kabir, and F. Liu, "Real-time recognition with the entire Brodatz texture database," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, (New York), pp. 638–639, June 1993.

[4] F. Liu and R. W. Picard, "Periodicity, directionality, and randomness: Wold features for image modeling and retrieval," *IEEE T. Patt. Analy. and Mach. Intell.*, To appear. Also MIT Media Laboratory Perceptual Computing TR#320.

[5] H. Tamura, S. Mori, and T. Yamawaki, "Textural features corresponding to visual perception," *IEEE T. Sys., Man and Cyber.*, vol. SMC-8, no. 6, pp. 460–473, 1978.

[6] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin, "The QBIC project: Querying images by content using color, texture, and shape," in *Storage and Retrieval for Image and Video Databases* (W. Niblack, ed.), (San Jose, CA), pp. 173–181, SPIE, Feb. 1993.

[7] D. Romer, "The Kodak picture exchange," April 1995. seminar at MIT Media Lab.

[8] F. Cohen and D. Cooper, "Simple parallel hierarchical and relaxation algorithms for segmenting noncausal Markovian random fields," *IEEE T. Patt. Analy. and Mach. Intell.*, vol. PAMI-9, pp. 195–219, Mar. 1987.

[9] J. K. Goutsias and J. M. Mendel, "Simultaneous optimal segmentation and model estimation of nonstationary noisy images," *IEEE T. Patt. Analy. and Mach. Intell.*, vol. II, pp. 990–998, Sept. 1989.

[10] C. Bouman and B. Liu, "Multiple resolution segmentation of textured images," *IEEE T. Patt. Analy. and Mach. Intell.*, vol. PAMI-13, no. 2, pp. 99–113, 1991.

[11] S. C. Zhu, T. S. Lee, and A. L. Yuille, "Region competition: Unifying snakes, region growing, energy/Bayes/MDL for multi-band image segmentation," in *Int. Conf. on Computer Vision*, (Boston, MA), pp. 416–423, 1995.

[12] T. Darrell and A. P. Pentland, "Cooperative robust estimation using layers of support," *IEEE T. Patt. Analy. and Mach. Intell.*, 1995.

[13] M. Kunt, A. Ikonomopoulos, and M. Kocher, "Second-generation image-coding techniques," *Proc. IEEE*, vol. 73, no. 4, pp. 549–574, 1985.

[14] T. M. Strat and M. A. Fischler, "Context-based vision: Recognizing objects using information from both 2-d and 3-d imagery," *IEEE T. Patt. Analy. and Mach. Intell.*, vol. 13, pp. 1050–1065, Oct. 1991.

[15] E. Saund and T. P. Moran, "Perceptual organization in an interactive sketch editing application," in *Proc. Fifth International Conference on Computer Vision*, (Cambridge, MA), pp. 597–604, June 1995.

[16] R. A. Jarvis and E. A. Patrick, "Clustering using a similarity measure based on shared near neighbors," *IEEE T. Comp.*, pp. 1025–1034, Nov. 1973.

[17] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[18] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. Wiley-Interscience, 1973.

[19] E. Saber, A. M. Tekalp, R. Eschbach, and K. Knox, "Annotation of natural scenes using adaptive color segmentation," *IS&T/SPIE Electronic Imaging*, Feb. 1995. San Jose, CA.

[20] R. W. Picard and T. P. Minka, "Vision texture for annotation," *Journal of Multimedia Systems*, vol. 3, pp. 3–14, 1995.

[21] T. Minka, "An image database browser that learns from user interaction," Master's thesis, MIT, 1996.

[22] T. M. Mitchell, "The need for biases in learning generalizations," Tech. Rep. CBM-TR-117, Rutgers University, May 1980.

[23] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, ch. 3, p. 58. John Wiley & Sons, 1973.

[24] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[25] L. Breiman, J. H. Freidman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[26] R. S. Michalski, "A theory and methodology of inductive learning," *Artificial Intelligence*, vol. 20, no. 2, pp. 111–161, 1983.

[27] R. W. Picard and T. Kabir, "Finding similar patterns in large image databases," in *Proc. IEEE Conf. on Acoustics, Speech, and Signal Proc.*, (Minneapolis, MN), pp. V–161–V–164, 1993.

[28] T. Chang and C. C. J. Kuo, "Texture analysis and classification with tree-structured wavelet transform," Tech. Rep. USC-SIPI198, University of Southern California, Los Angeles, CA, February 1992.

[29] T. Kohonen, *Self-Organization and Associative Memory*. Berlin, Heidelberg: Springer, 1984. 3rd ed. 1989.

[30] T. Uchiyama and M. A. Arbib, "An algorithm for competitive learning in clustering problems," *Pattern Recognition*, vol. 27, pp. 1415–1421, October 1994.

[31] Y.-I. Ohta, T. Kanade, and T. Sakai, "Color information for region segmentation," *Comp. Graph. and Img. Proc.*, vol. 13, pp. 222–241, 1980.

[32] H.-J. Zhang and S. W. Smoliar, "Developing power tools for video indexing and retrieval," in *Proceedings SPIE Storage and Retrieval for Image and Video Databases II* (W. Niblack and R. C. Jain, eds.), (San Jose, CA), pp. 140–149, SPIE, Feb. 1994. Vol. 2185.

[33] A. Pentland, R. W. Picard, and S. Sclaroff, "Photobook: Tools for content-based manipulation of image databases," in *SPIE Storage and Retrieval of Image & Video Databases II*, (San Jose, CA), pp. 34–47, Feb. 1994.

[34] J. K. Wu, A. D. Narasimhalu, B. M. Mehtre, C. P. Lam, and Y. J. Gao, "CORE: a content-based retrieval engine for multimedia information systems," *Multimedia Systems*, vol. 2, pp. 25–41, Feb. 1995.

[35] R. L. Delanoy and R. J. Sasiela, "Machine learning for a toolkit for image mining," Lincoln Laboratory 1017, MIT, Lexington, MA, March 1995.

[36] H.-J. Zhang and D. Zhong, "A scheme for visual feature based image indexing," in *SPIE Conference on Storage and Retrieval for Image and Video Databases*, (San Jose, CA), Feb. 1995.

[37] T. Kohonen, "Learning Vector Quantization," *Neural Networks*, vol. 1, no. Supplement 1, p. 303, 1988.

Figure 12: The FourEyes computer-assisted annotation tool. The user has mouse-clicked some patches of sky in the two right images, and assigned them the label "sky." Within-image groupings allowed FourEyes to grow those labeled patches into larger "sky" regions (indicated by cross-hatching). Across-image groupings allowed FourEyes to also place tentative labels on the two left images. The menu buttons allow the user to control which sets of groupings are available to the learner.

Figure 13: The first twelve natural scenes and their ground-truth labelings. Regions labeled "building" are colored black; "car" is yellow, "grass" is green, "leaves" is cyan, "person" is red, "sky" is blue, and "water" is purple. Unlabeled regions are white.

Figure 14: The remaining thirteen natural scenes and their ground-truth labelings.