# Toolkit for Image Mining: User-Trainable Search Tools

Richard L. Delanoy

■ **A computer environment called the Toolkit for Image Mining (TIM) is being developed to assist users in creating search tools for pattern-matching tasks such as content-based image retrieval. TIM provides users who have diverse interests and skill levels with the ability to create and refine search tools in an interactive process. The user simply points at examples and counterexamples of the object of interest. A learning algorithm then uses these inputs to build a model of the user's intentions incrementally, from which a search tool is constructed and a visual feedback of search results is presented to the user. The user may then point at mistakes made by the search tool to refine performance further.**

**Search tools are constructed in the form of functional templates, which are generalized matched filters capable of knowledge-based image processing. The ability of this system to learn the user's intentions from experience contrasts with other existing approaches to content-based image retrieval that perform searches on the characteristics of a single input example or on a predefined and semantically constrained textual query. Currently, TIM is capable of learning spectral and textural patterns, but should be adaptable to learning shapes as well. Other possible applications of TIM include quantitative image analysis, generation of metadata for annotating images, prioritization or reduction of data in bandwidth-limited situations, and construction of components for more complex computer-vision algorithms.**

As the costs of collecting, storing, and transmitting images have decreased, the sizes of image collections have dramatically increased. Searching and retrieving data in large image archives is cumbersome and inefficient, and image retrieval has become the limiting factor in the exploitation of images. The severity of the problem and how to deal with it have become subjects of national discussion [1, 2] and conferences [3].

One way to structure the rapid retrieval of archived images is through the use of metadata—a collection of keywords and computed indices that are used to annotate each image. For example, an image can be tagged with information about its generation. These kinds of metadata successfully support a query such as "Find all Landsat images of central Florida collected during June of 1993."

While metadata help reduce the scope of a search, they are not suitable for searches in which the content must be examined. Examples of this kind of search include finding images of movable targets in radar imagery, prospecting for minerals in hyperspectral image data, locating tropical storms in weather satellite images, or identifying a face in airport-surveillance camera images.

In the past, the most common approach to content-based image searching and retrieval was customized algorithms, which were designed to look for specific signatures. While such algorithms were usually effective as search tools, they were expensive to create, were suitable only for the original search problem, and required an expert in computer vision or image processing to adapt the algorithm to changing environmental conditions or goals. For example, an algo-

rithm for detecting deciduous forests in satellite data, originally designed to work on images that were collected in the summer, would require a computer-vision expert to adapt it for use with images taken during the winter.

Academic and research institutions have been actively developing image-database retrieval tools that do not require the user to have computer programming skills. Examples include the QBIC System by M. Flickner and colleagues at IBM [4]; MultiSpec by D. Landgrebe and colleagues at Purdue University [5, 6]; Image Context Vectors by S.I. Gallant and colleagues at Belmont Research, Inc. [7, 8]; an eigenfilter approach by R.W. Picard and T. Kabir at MIT [9]; and Chabot by V.E. Ogle and M. Stonebraker at the University of California at Berkeley [10].

In general, these tools permit only a single input example to define a search. The input example varies—it can be a word or phrase with a predefined meaning (e.g., "mostly yellow") [4, 10], a predefined menu item (e.g., a selected texture from a texture sampler) [4, 10], a user-drawn shape [4], a user-selected set of pixels [5, 6], a whole image [7–9], and others. In each case, the input example is characterized by a set of feature attributes, which can be compared with the tagged values of archived images. Matches between images and the input are evaluated according to a distance measurement, such as Euclidean or Hamming distances, and ranked. Images with the highest match scores are returned to the user.

Although useful in many applications, these algorithms have disadvantages. Many are tailored to the task of matching the example to an entire image [4, 7, 8, 10], making the search for a pattern that occupies a small portion of an image difficult. Another problem is a lack of flexibility and adaptability. Because queries and menus are based on a predefined syntax, the available options may only approximate future needs, perhaps largely limiting such algorithms to searches that developers had anticipated.

The biggest problem, however, is that a single example limits classification performance. A single-example input does not provide a way to establish what attributes consistently appear in the search target—the object or region being sought—and what features consistently discriminate the target from all others. The user must search with all feature attributes that were predefined by the tool developers, even if most of the feature attributes are irrelevant and decrease search performance.

In addition, users cannot incrementally refine a search tool. When search performance is unacceptable, the only recourse a user has is to provide a different example. What the user needs is a way for the search tools to learn the user's intentions interactively through experience.

We have created a new approach to content-based image search and retrieval, embodied in a prototype Toolkit for Image Mining (TIM) [11]. In this environment, the user interacts with a learning algorithm to create and refine customized search tools quickly. The algorithm learns from experience, building search tools that are adapted to the user's intentions.

The matching operation is done on a pixel-by-pixel basis, allowing detection even in cases in which the object or region being sought is a small part of an image. The trainable search tools are simple in structure so that they can be easily exported as independent agents to search remote image databases automatically. Moreover, the highly modular search tools can be linked to produce more complex functionality. Finally, TIM is not limited to only those applications anticipated by the algorithm developer.

In this article, we first describe the look and feel of the prototype TIM environment. Then we discuss functional-template correlation and how it is being used as the basis of the new approach to machine learning used in TIM. Finally, we show examples from our initial work in spectral pattern recognition in remotely generated images of the Earth's surface.

## The Toolkit Environment

TIM enables users who have some knowledge of an image-exploitation application, but little or no specialized computer knowledge, to construct customized search tools for a variety of applications. Users must have images on hand that allow them to identify examples of what they want to find. From these examples, TIM defines a boundary surface that divides examples from counterexamples in an $n$-dimensional data space, in which $n$ is the number of attributes used to characterize a region or an object.

TIM is meant to work on any image data, regardless of the sensor used to collect the imagery. While the current system described in this article is constrained to the task of learning spectral and textural patterns, we believe that the technique can be adapted to other pattern domains as well, including shapes.

Figure 1 shows the user interface of TIM at the start of a learning session. The menu window provides options for accessing image data. The *Dataset Selection* menu refers to the sensor type, such as "ir3," "sar," or a specific problem application such as "elk." In this example, the word "elk" refers to an elk habitat study using Landsat Thematic Mapper data.

Once a dataset has been selected, the set of images available for that dataset is shown in the *Image Selection* menu. Once an image has been selected, the available spectral bands in the *Band Selection* menu and other pixel-registered images, available through the *Registered Images* menu, can be selected as candidate attributes for learning. The pixel-registered im-

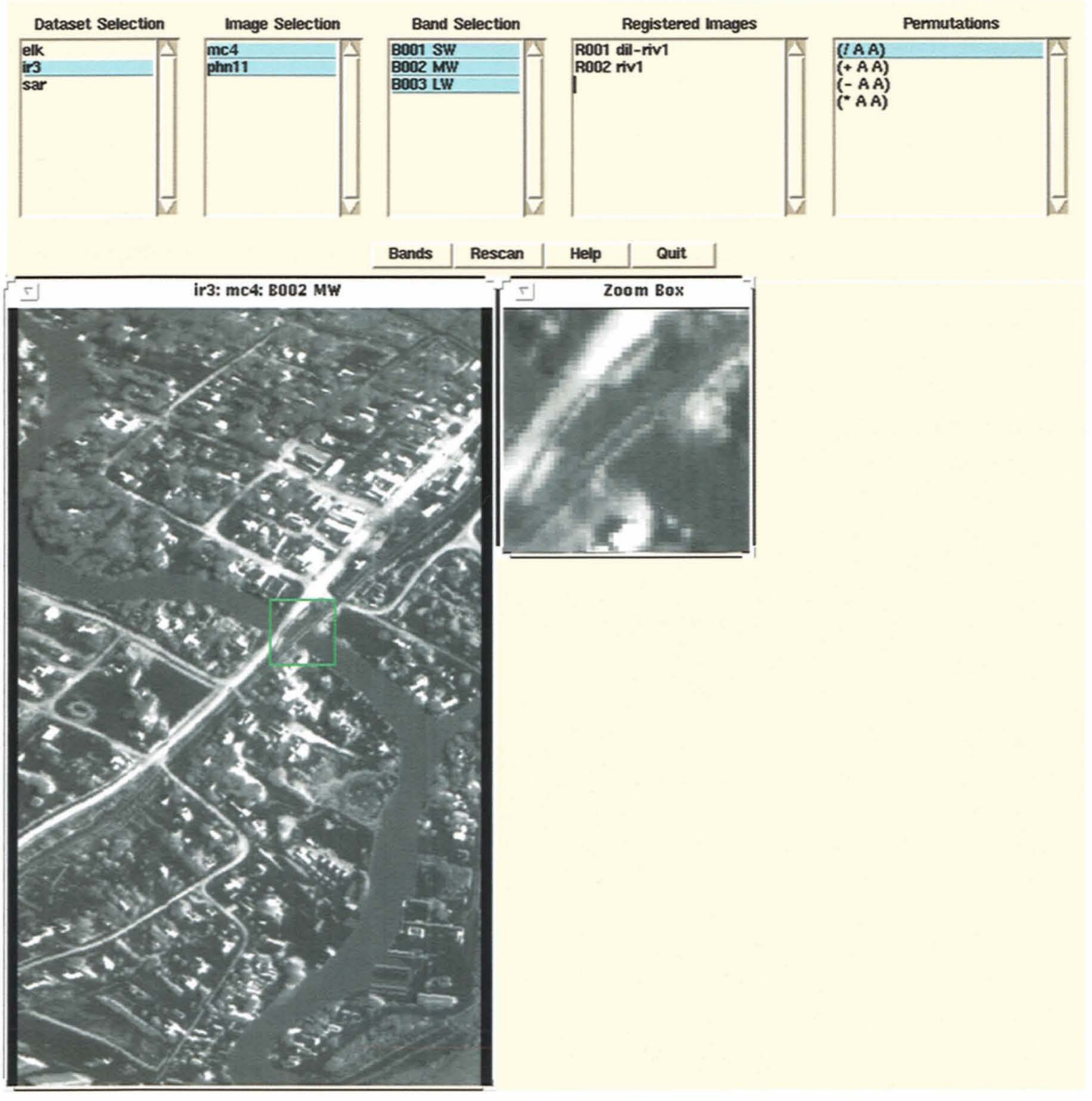


**FIGURE 1.** Initial display screen of the Toolkit for Image Mining (TIM). The control window on the left shows a selected band of a three-band infrared image of a town. The portion of the image within the green square has been selected by the user and enlarged for search-tool training (right).

ages are typically transformations of the input images, each providing a pixel-level map for a locally computed texture or shape attribute. Such transformations of the input images are generated either from the outputs of previously created search tools or from a menu of texture or shape analysis options defined elsewhere in TIM (not shown), or are generated outside of TIM and imported into the environment.

The *Permutations* menu defines ways in which the available spectral bands and registered images can be combined as additional candidate attributes for learning. For example, the pairwise ratios of all bands and registered images are generated with the (/AA) option, where the letter A stands for "All" bands and registered images.

This example displays a particular band from the selected dataset and image: the control window labeled ir3:mc4:B002 MW refers to dataset ir3, image name mc4, and band B002 (medium wavelength). A movable zoom box magnifies a portion of the image and selects a particular subimage for learning.

Once the selections have been made from the image-selection window, the user can then choose which bands or registered subimages to display during learning, by using the *Tile Display Manager* menu in the *Learning Environment* window, as shown in Figure 2. In this figure, which illustrates how a search tool is trained to detect water, all bands of a three-band infrared image are displayed with the abbreviations SW, MW, and LW for short-, medium-, and long-wave infrared, respectively.

The displayed subimage contains two bridges crossing a river. Feedback is presented to the user in the form of an interest image—a spatial map of evidence. This feedback is shown in the current interest window, or CURINT. Shades of gray indicate how

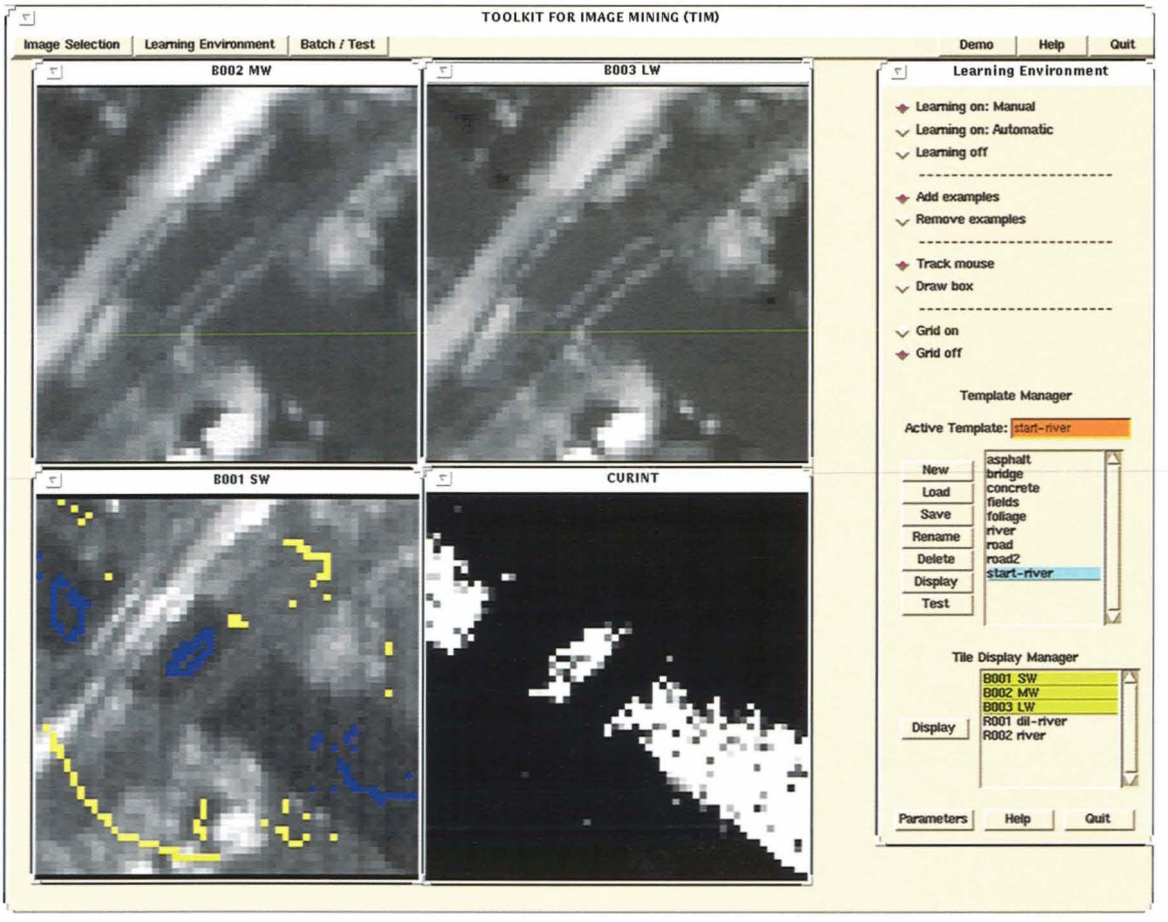


**FIGURE 2.** Example of a TIM learning session. Examples (blue pixels) and counterexamples (yellow pixels) in a subimage extracted from the image in Figure 1 are used to train a search tool to detect water.

well each pixel matches the pattern encoded in the search tool. Initially, when no examples or counterexamples are provided, this window is all black. In the CURINT window, each pixel is represented by a vector containing six components: the three raw spectral band values, and the three relevant pairwise ratios (SW/MW, SW/LW, and MW/LW) selected with the (/AA) permutations option.

After making the various selections, the user presses the New button in the Template Manager to initialize a new functional template, and then presses the Learning on: Manual button. In this figure, the user is creating a search tool to find images with water. The user guides the cursor to an area of the river in one of the subimages and presses the left mouse button. In response, TIM colors the indicated pixel blue, as shown in the B001 SW window in Figure 2. TIM then builds a search tool from the single input, applies the search tool to the selected subimages, and displays the match results in the CURINT window.

With a search tool based on only a single example pixel, the algorithm generally highlights much more than just river. To refine the search tool, the user can introduce a counterexample by moving the cursor in the CURINT window to a high-value pixel that is not part of the river, and pressing the middle mouse button. This counterexample pixel is colored yellow, as shown in Figure 2.

TIM adds this counterexample to the set of previously collected examples and counterexamples, constructs a new search tool, applies the search tool to the selected subimages, and again displays the result as feedback to the user. Figure 2 shows the result of several input examples and counterexamples in which TIM has converged on a reasonable search tool for water in the selected images. Each cycle takes about one second to complete on a Sun SPARCstation 10.

This two-way dialogue between the user identifying mistakes and TIM providing feedback on the characteristics of the newest search tool continues until the user is satisfied with detection performance. At that time, the user can test the search tool on the whole image from which the subimages were extracted or other images found in the dataset. Any mistakes that appear in other subimages can be added as additional examples or counterexamples.

Once the search tool is complete it can be saved, either for future use in TIM or for export outside of TIM. For content-based image retrieval, the search tool could be sent to an image database to explore images as an agent of the user. For the agent to make a retrieval decision, the user would also select one of several optional criteria for image selection. For example, the user might request ten images with the highest summed match scores, or request all images that have more than some threshold average match score. The user would explore the returned images to assess the performance of the agent and to look for interesting object or region variants found by the agent. The user always has the option of reentering the learning environment to refine the agent further.

## Knowledge-Based Image Processing

The core technology of TIM is a new machine-learning algorithm based on two techniques of knowledge-based image processing—interest images and functional-template correlation. These two techniques were originally developed in the context of automatic target recognition [12], but have been successfully extended to other areas, particularly to the problem of automatically detecting hazardous weather in Doppler radar images [13–15].

An *interest image* is a spatial map of evidence for the presence of a feature [16]. To be useful in a search, the feature must be relatively characteristic of the region or object being sought. Pixel values in interest images range from 0 to 1. Higher pixel values indicate a greater confidence that an intended feature is present at that location.

One role for interest images is to serve as a common denominator for data fusion. Input data from a variety of sensor sources can be transformed into one or more interest images that highlight a set of features thought to be indicative of a target. With simple or arbitrarily complex rules of arithmetic, fuzzy logic, or statistics, these individual interest images can then be combined into a single interest image.

The conditions under which an interest image is a good discriminant can also be used to guide the weighting of each individual interest image during data fusion. Ultimately, a detection algorithm with complex properties can be assembled from a set of

simple feature detectors, each of which generates an interest image. In addition to their role in data fusion, interest images provide a means of representing information for human interpretation. Highlighting interesting features draws human or algorithmic attention to a particular image location.

*Functional-template correlation* (FTC) is a generalization of matching filtering and is used to create customized, knowledge-based image processing operations [17, 18]. By using aspects of fuzzy set theory, FTC transforms raw image data into maps of match scores (interest-image values). FTC can most easily be understood as an extension of autocorrelation, which is described below. Given an input image $I$, an output image $O$ is generated in autocorrelation by matching a kernel $K$ (essentially a subimage of the image being searched) to a local neighborhood surrounding each pixel location $I_{xy}$.

Each element of the kernel $K$ is a literal image value expected to be present for a good match. When $K$ is tested at each pixel location $I_{xy}$, each element of $K$ superimposes an image value in the local neighborhood of $I_{xy}$. The sum of the products of these superimposed kernel and image pairs is normalized and becomes the match score placed in $O_{xy}$. If the shape to be matched can vary in orientation, then the pixel $I_{xy}$ is probed by $K$ at multiple orientations. The score assigned to $O_{xy}$ is the maximum across all orientations.

FTC is fundamentally the same operation as autocorrelation, with one important exception: whereas each kernel element of autocorrelation is an image value, each kernel element in a functional template is a scoring function that encodes a relationship between input image values and returned scores. High scores are returned whenever the input image value falls within the fuzzy limits of expected values. Low scores are returned whenever the input value falls outside these limits. The set of scores returned for each element of the kernel $K$ are averaged and clipped to the continuous range of (0.0,1.0). (In the clipping process, those averaged scores below zero are assigned a value of zero, while those averaged scores which are greater than one are assigned a value of one.)

In our implementation of FTC, a functional template consists of a kernel of indices; each index is associated with a scoring function that has been precom-

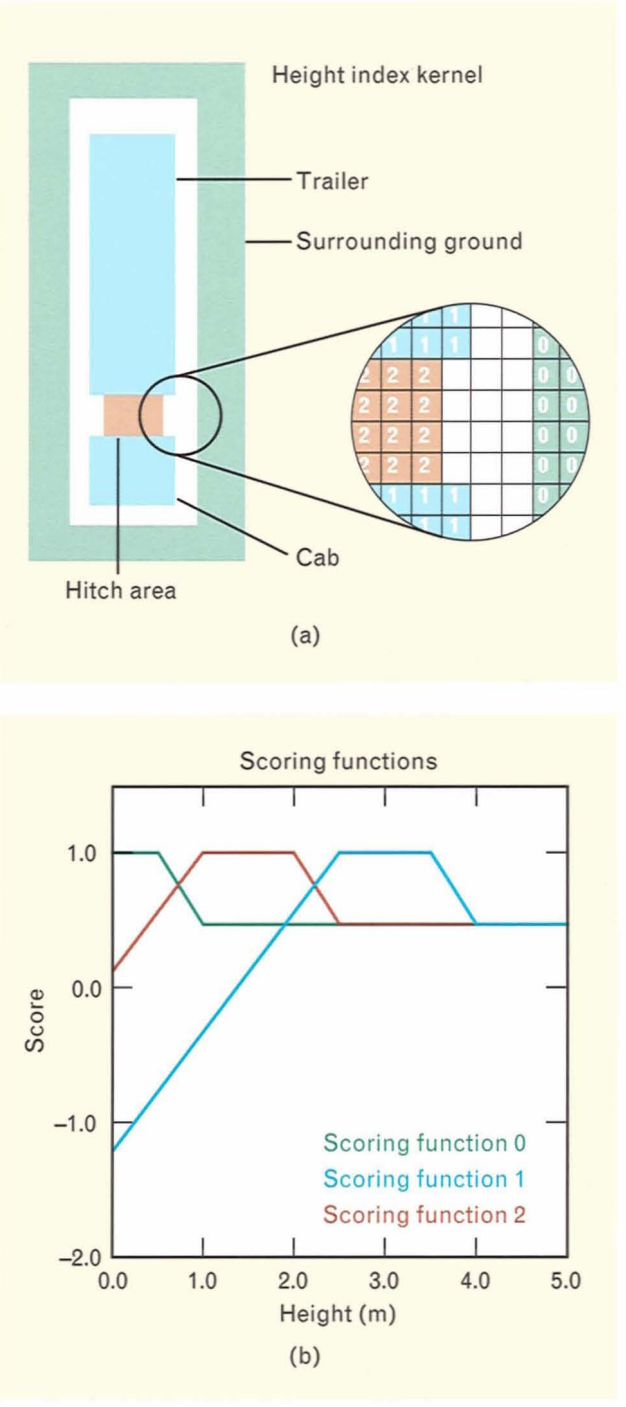


**FIGURE 3.** (a) Example of a functional template for the top view of a tank truck. Input images, derived from laser radar data, contain values representing height above the local ground level. These values are probed with the height index kernel. (b) The indices in the height index kernel determine which scoring functions are applied to the superimposed input image pixels. The scores returned from the scoring functions are averaged to generate a match score.

puted as a lookup table. As in autocorrelation, if the feature being sought can vary in orientation, then the match score is the maximum average score computed across multiple orientations of the kernel.

Consider the functional template designed to detect tank trucks in downlooking laser radar images. In this example, range values have been converted to heights above ground. Figure 3(a) shows the template kernel for a tank truck, consisting of integers that correspond to the three scoring functions 0, 1, and 2. Elements of the kernel that do not correspond to the scoring functions form guard regions in which image values are ignored and have no effect on match scores.

As shown in Figure 3(b), scoring function 1, corresponding to the top of the cab and trailer, returns a maximum score of 1.0 for heights from 2.5 to 3.5 m, and strongly inhibits scores for heights below 1.0 m. Note that scores returned for heights above 4.0 m in scoring function 1 drop only to a minimum score of 0.5, instead of dropping to a large negative score, as is seen for heights near 0.0 m.

The negative scores account for the fact that tank trucks are opaque to laser illumination, which means that the presence of ground-level heights where the cab or trailer is expected gives strong evidence that a target is not at that location. Heights greater than the expected interval of 2.5 to 3.5 m result in scores no less than 0.5—the level of ambiguity—because such heights could potentially indicate the presence of an occluding surface. In the real world, the cab of a tank truck may or may not be present underneath an occluding surface at least 4.0 m high.

The other scoring functions work in the same manner, except that the expected interval of heights for the background in scoring function 0 is from 0.0 to 0.5 m, and the expected interval for the hitch area in scoring function 2 is from 1.0 to 2.0 m. These scoring functions are tuned such that, when the template is applied to a patch of bare ground (zero height), the negative scores from scoring function 1 balance the positive scores from scoring functions 0 and 2, resulting in an overall score near 0.0. An unobscured target should generate a score near 1.0.

In general, by increasing or decreasing the interval of image values over which affirming scores are returned (i.e., scores greater than 0.5), the user can encode which image values are expected with varying degrees of uncertainty. In addition, knowledge of how a feature or object appears in sensor imagery can be encoded in scoring functions. Various design strategies can minimize the interfering effects of occlusion, distortion, noise, and clutter [17]. Consequently, functional templates customized for specific applications are generally more robust than standard generic signal-processing operations. FTC has been used as a direct one-step means of detecting 3-D objects and can be used to implement fuzzy knowledge-based versions of edge detection, thin-line filtering, thin-line smoothing, shape matching, skeletonizing of shapes, and shape erosion.

Data-fusion capabilities have been implemented in FTC, allowing multiple kernels to be stacked as a single functional template. Each kernel has a set of scoring functions and probes a different input image, to produce a single output interest image. This technique assumes that each input image is pixel registered to the others so that spatial scale and geographical area match and that all kernels share a common center of rotation. A match score for pixel $O_{xy}$ is computed by averaging the set of scores computed for all scoring functions of all kernels. If the target can vary in orientation, then the kernels of all functional templates are rotated as a unit.

### Functional-Template Learning

In functional-template learning, multispectral images are probed with a kernel constructed for each spectral band. Each kernel consists of a single pixel and an associated scoring function. Transformations of the input data based on local texture or shape analysis are also assigned a kernel and scoring function. Permutations of multiple bands and/or transformations into a single value, such as the ratio of two bands, are also considered as inputs and likewise assigned a scoring function. Given this structure for encoding spectral and textural patterns as functional templates, learning is implemented by constructing each scoring function from a set of corresponding attribute values. The attribute values are extracted from the user's set of examples and counterexamples.

We based our approach to functional-template learning on the construction of distribution histo-

grams. For each attribute (spectral band, transformation, or permutation), two histograms are generated. One histogram is constructed from the attribute value for all examples indicated by the user. The second histogram is likewise constructed from counterexamples. The scoring function is constructed by comparing how much examples and counterexamples overlap at each input image value. Once the scoring functions have been generated, those which produce the best discrimination between indicated examples and counterexamples are selected for the construction of a functional template.

To construct histograms, the system adjusts the scales of band, transformation, and permutation values. The mean and standard deviations of the example (but not the counterexample) values are computed for each attribute value. These statistics are then used to scale the attribute values to a range of 0 to 255. The mean value is scaled to 127, while 0 and 255 are mapped to the attribute values that are $n$ standard deviations (where $n$ is typically 3) below and above the mean, respectively. Attribute values more than $n$ standard deviations from the mean are clipped to 0 and 255. This scaling is applied to all example and counterexample values for that attribute.

Next, the example histogram is constructed from the set of scaled example values. Each value is added to the histogram by increasing the count of the corresponding bin. For example, if a scaled value is 201, then the histogram bin at index 201 is increased by one. If the total number of examples is small, then some number of bins to either side (e.g., bins 199, 200, 202, and 203) are also increased to approximate a more fully populated distribution. The number of adjacent bins to be adjusted is computed in the current system as follows:

$$\text{width} = \text{int}\left(\frac{p}{\sqrt{n}}\right),$$

where $p$ is a user-tunable parameter (typically set to 16), $n$ is the number of input example and counterexample values, and int ( ) is the truncation operator. A counterexample histogram is likewise constructed by using the same width computed for examples. As the width value changes with increasing $n$, the histograms are reconstructed.

Scoring functions are computed by comparing example and counterexample histograms, as shown in Figure 4. Each scoring function is implemented as a lookup table of 256 scores, one for each possible scaled image value. The score $S_k$ to be returned for each of the 256 possible scaled input attribute values (where the index $k$ varies from 0 to 255) is computed as follows:

$$S_k = \begin{cases} 0.0 \quad \text{if } H_k^{ex} \text{and } H_k^{cex} = 0 \\ -w \times R\left(H_k^{cex}, \sum_{i=0}^{255} H_i^{cex}, l_1, h_1\right) \quad \text{if } H_k^{ex} = 0 \\ w \times R\left(H_k^{ex}, \sum_{i=0}^{255} H_i^{ex}, l_2, h_2\right) \quad \text{if } H_k^{cex} = 0 \\ w \times R\left(H_k^{ex}, H_k^{cex}, l_3, h_3\right) \quad \text{if } H_k^{ex} > H_k^{cex} \\ -w \times R\left(H_k^{cex}, H_k^{ex}, l_4, h_4\right) \quad \text{otherwise,} \end{cases} \tag{1}$$

where $w$ is a scaling factor (typically 16), $i$ is the bin number from 0 to 255, $H^{ex}$ and $H^{cex}$ are the example and counterexample histograms, $l$ and $h$ are ramp end points (currently 0.0 for all $l$, 0.05 for $h_1$ and $h_2$, and 40.0 for $h_3$ and $h_4$). The ramp function $R(a, b, c, d)$ is shown below, returning a score from 0 to 1:

$$R(a, b, c, d) = \min\left\{1.0, \left[\frac{\max\left(0.0, \frac{a}{b} - c\right)}{d - c}\right]\right\}.$$

Once all scoring functions have been generated, the algorithm automatically selects a subset of the most discriminating scoring functions by using the technique of forward selection [19]. In this technique, the single scoring function that best partitions the examples and counterexamples is first selected. Next, the remaining scoring functions are each tested in combination with the first selected scoring function. Of these pairs, the algorithm again selects the one that best partitions the inputs. The remaining scoring functions are then each tested in combination with the first two selected scoring functions. Scoring

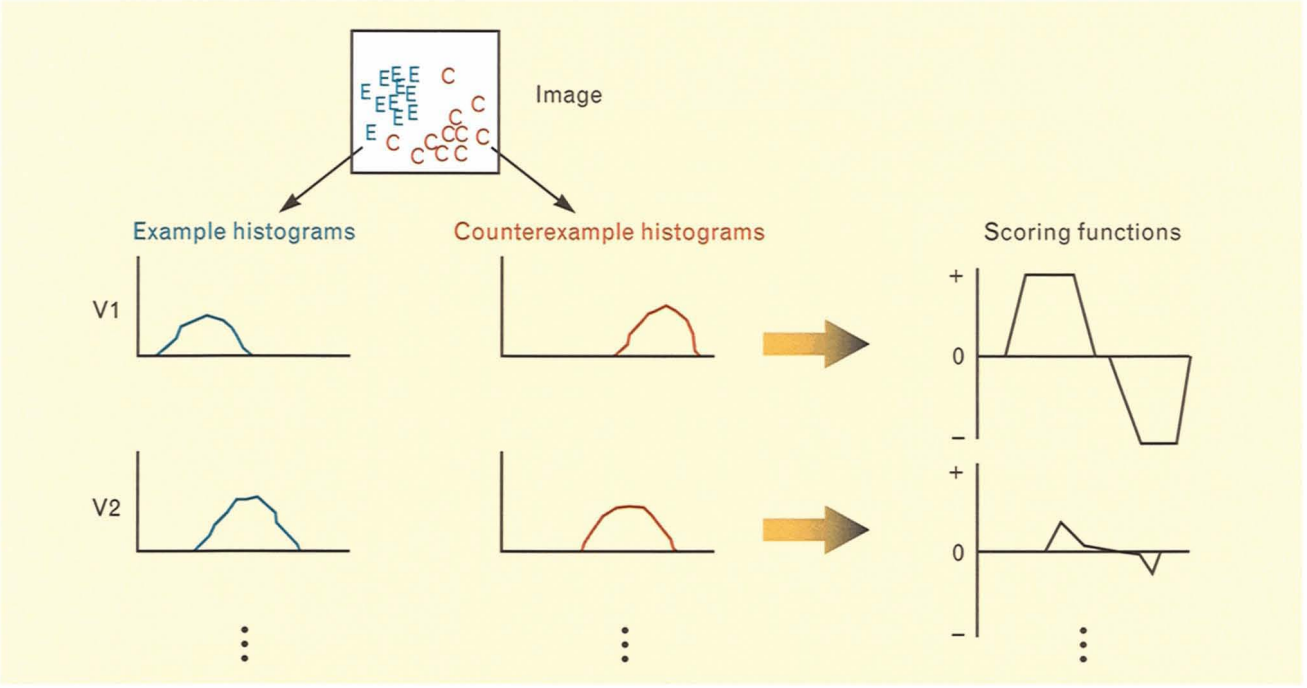


**FIGURE 4.** Construction of scoring functions from example and counterexample histograms. Each example (E) and counterexample (C) selected by the user in an image is characterized by some number of spectral bands or transformed attributes (variables), represented as V1, V2, .... F or each bin of a pair of histograms, a score in the corresponding scoring function is computed as in Equation 1. In bins where there are many example values and few counterexample values, the returned score is high. In bins where there are many counterexample values with few examples, the returned score is negative. In bins where there are equal numbers of example and counterexample values, the returned score is zero. The set of scoring functions that are determined to be most discriminating by the method of forward selection are combined to form a functional template.

functions are added in this manner until a maximum partitioning is encountered or a maximum number of scoring functions has been included. Backward selection—finding the scoring function that when removed from the previously accumulated set increases the partition accuracy—is also an option.

## Evaluation

We evaluate search tool functionality with two examples. First, we continue our previous discussion of using a functional template to search an image database for images of water. Then in the second example we discuss using TIM to classify vegetation types in Landsat data.

To search a database for images of water, the search tool must be assigned some selection criterion, which may be a threshold applied to some simple statistic computed from the interest values generated by the search tool. One selection-threshold mechanism would be to choose those images with an average interest value above a threshold. An alternative would be to select images in which a number of interest pixels have values above 0.5.

Figure 5 illustrates the selection results. At the left side of the figure is the full-size image of Figure 1. At the right side is the interest image generated by applying the trained search tool to the whole image. The yellow boxes enclose tiles that have average interest values of 0.06 or higher in the range (0.0,1.0). The same mechanism could be applied to the task of looking for high-interest images within a large image database. In addition to the functional template created for water, we have also generated templates for concrete, vegetation, and asphalt for images created by this sensor, as shown in Figure 6.

The second example in our evaluation of search-tool functionality examines the more difficult and realistic task of classifying vegetation types in Landsat

Thematic Mapper images. T.P. Huber and K.E. Casler have reported how they evaluated a variety of simple classification metrics and formulas developed by wildlife biologists [20]. In an image of a 190 × 168-km region in Colorado, each pixel was assigned one of fourteen vegetation types by analysts. In comparing classification results with the human-identified "truth," the percentage of correct classifications did not exceed 50%. Even when the researchers included digital-elevation, ground-slope, and aspect data, the performance was still less than 70% correct classification. Huber and Casler concluded that using remote sensing information in complex terrains could be a difficult and relatively inaccurate process.

Using the same Thematic Mapper data, M. Augusteijn et al. [21] applied a cascade-correlation neural net (an enhanced version of back propagation) [22] to classify spectral patterns. For testing and training, they chose data consisting of homogenous 8 × 8-pixel boxes—all 64 pixels were the same vegetation

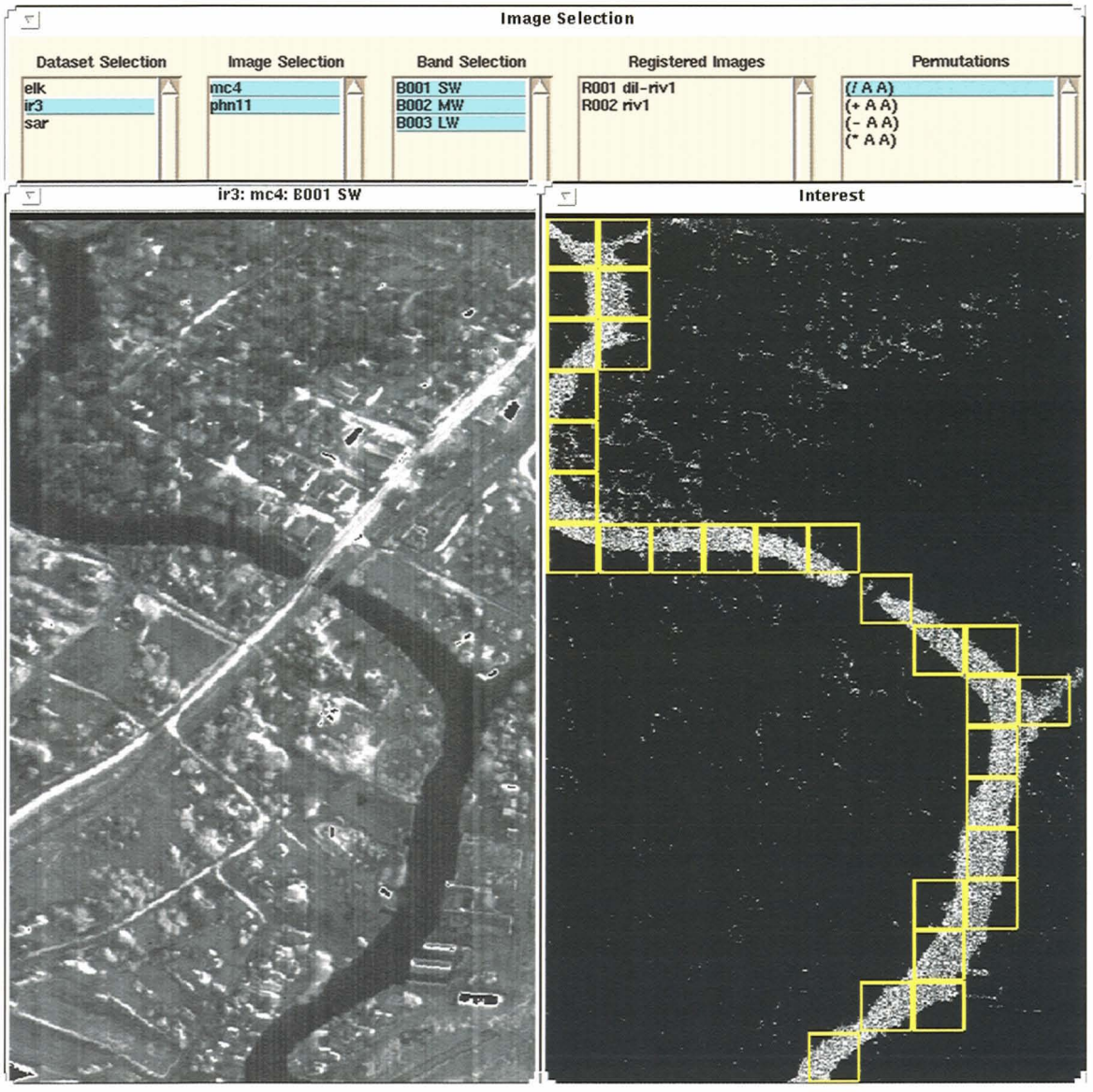


**FIGURE 5.** Test image (left) and interest image (right) resulting from the application of the functional template constructed in Figure 2. The yellow boxes overlaid on the interest image are those which contain more than a threshold average interest value. If the test image represents part of an image dataset, these boxes would indicate images that would be retrieved for the user to see.

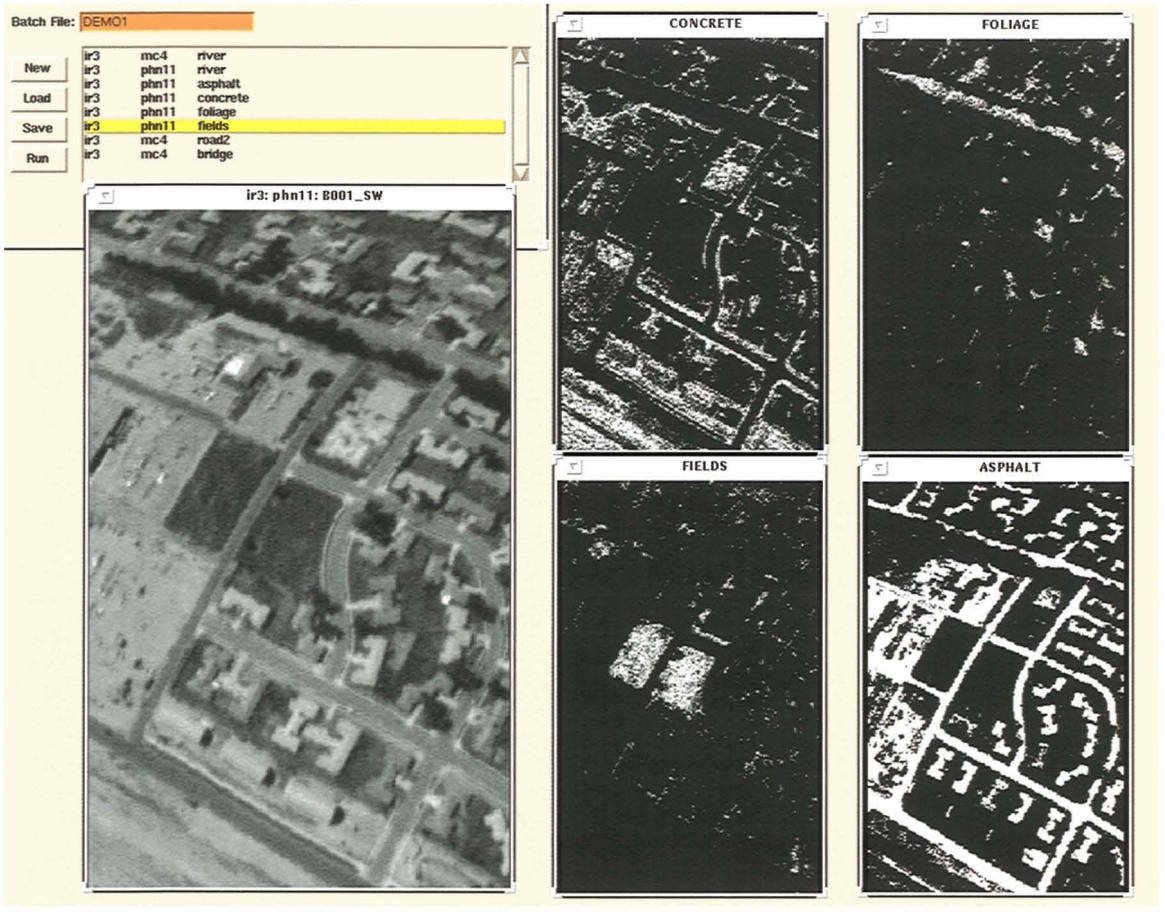


**FIGURE 6.** A test image (left) and four functional templates (right) trained in the TIM environment to recognize features such as concrete, vegetation (foliage and fields), and asphalt in three-band infrared data.

type. Of the fourteen vegetation types, only nine were available in boxes of this size. The task was to train the neural net on half of these boxes and then attempt to identify the types of vegetation in the remaining boxes. With this approach, they achieved approximately 98.8% correct classifications for the nine available classes.

We repeated the experiment done by Augusteijn et al., replacing the cascade-correlation neural net with TIM and functional-template learning. In addition, rather than use only $8 \times 8$-pixel boxes containing a single vegetation type, we conducted a series of tests with pixels that were at the center of homogeneous boxes of varying sizes ($9 \times 9$, $7 \times 7$, $5 \times 5$, and $3 \times 3$).

For each box size, half of the center pixels were randomly assigned to a training set while the other half were assigned to a test set. Functional templates for each of the fourteen vegetation types were trained separately. Classification was based on a simple winner-take-all strategy; that is, the template generating the highest match score for a pixel was the one that assigned the class. More robust decision strategies that look for patterns in match scores probably could have achieved better results.

Training was done in an automated mode of the TIM environment. With truth data available, automatic scoring and feedback were possible. A simple rule-based example selection procedure was implemented to locate errors of omission and commission and automatically add these pixels to the sets of examples and counterexamples, respectively. With these new inputs, the functional template was modified and applied to all the training pixels, and the results were scored to decide what new examples and counterexamples should be added next. For each vegetation type, the template that generated the highest

**Table 1. Vegetation Classification Results**

| | | | *Training Data* | | *Test Data* | |
|---|---|---|---|---|---|---|
| *Box Size* | *Number of Classes* | *Training Inputs* | *N* | *Percent Correct* | *N* | *Percent Correct* |
| 9 × 9 | 10 | 128 | 1055 | 100.0 | 1101 | 98.1 |
| 7 × 7 | 12 | 494 | 2766 | 100.0 | 2599 | 98.7 |
| 5 × 5 | 14 | 2614 | 9113 | 99.9 | 9305 | 98.1 |
| 3 × 3 | 14 | 13,218 | 59,527 | 98.8 | 59,846 | 97.5 |

score during training was the one chosen for testing. Training ended if one of three conditions was met: (1) TIM learned to discriminate perfectly the target vegetation pixels from all other pixels, (2) all instances of the particular vegetation type had been used for training, or (3) the total number of examples and counterexamples exceeded an arbitrary limit.

Table 1 shows the classification performance of the fourteen templates on initial training boxes and on separate test boxes. The header *Number of Classes* refers to the number of classes represented by at least one example. *Training Inputs* is the total number of input examples and counterexamples from the *Training Data* used to train all templates. *N* refers to the number of available pixels in the training and testing datasets. According to the table, trained functional templates applied to test data performed at a level comparable to the results of Augusteijn et al. (98.8% correct classification for 8 × 8 boxes) using the cascade-correlation neural net.

An additional experiment was done by using the 3 × 3-pixel-box trained templates on all 3.4 million individual pixels. A correct classification percentage of 83.0% was observed. The lower classification performance was due in large part to the high percentage of pixels lying on the border between two or more vegetation types. More than 96% of all pixels in the image were in contact with another pixel having a different vegetation type (only 119,373—the sum of 59,527 and 59,846—out of 3.4 million pixels were determined to be the center of a 3 × 3-pixel homogeneous patch of vegetation). Indeed, the 83.0% correct classification rate seems robust, given the large number of pixels likely to have mixtures of vegetation types and correspondingly confused spectral patterns. For comparison, Huber and Casler achieved less than 50% correct classification for all pixels by using only spectral information [20].

Table 2 shows the fourteen vegetation classes allocated to the Landsat image. Table 3 shows the confu-

**Table 2. Vegetation Classes**

| *Class* | *Vegetation Type* |
|---|---|
| 1 | Ponderosa pine |
| 2 | Douglas fir |
| 3 | Spruce fir |
| 4 | Mixed conifer |
| 5 | Limber/bristlecone pine |
| 6 | Aspen/conifer mix |
| 7 | Non-vegetated |
| 8 | Aspen |
| 9 | Water |
| 10 | Wet meadow |
| 11 | Riparian deciduous shrub |
| 12 | Mesic grassland |
| 13 | Dry meadow |
| 14 | Alpine |

**Table 3. Confusion Matrix for All Templates Trained with 3 x 3-Pixel Boxes***

| | *N* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *14* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *1* | 10,453 | 98.2 | 0.4 | 0.0 | 0.0 | 0.8 | 0.0 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.0 |
| *2* | 5506 | 1.1 | 98.1 | 0.2 | 0.3 | 0.1 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 |
| *3* | 5440 | 0.0 | 0.5 | 97.3 | 1.9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| *4* | 4144 | 0.0 | 0.3 | 1.3 | 97.5 | 0.7 | 0.1 | 0.1 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| *5* | 1290 | 1.4 | 0.0 | 0.0 | 0.0 | 92.4 | 0.0 | 4.1 | 0.1 | 0.0 | 0.0 | 0.0 | 0.1 | 1.3 | 0.5 |
| *6* | 565 | 0.9 | 1.2 | 0.0 | 5.1 | 0.0 | 80.5 | 0.0 | 12.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| *7* | 747 | 0.4 | 0.0 | 0.0 | 0.0 | 0.7 | 0.0 | 83.5 | 0.3 | 0.0 | 0.4 | 0.0 | 1.7 | 5.6 | 7.3 |
| *8* | 1953 | 0.5 | 0.0 | 0.0 | 3.5 | 0.0 | 3.3 | 0.3 | 90.7 | 0.1 | 0.1 | 1.1 | 0.4 | 0.0 | 0.0 |
| *9* | 641 | 0.0 | 0.3 | 3.0 | 1.6 | 0.0 | 0.0 | 0.0 | 0.2 | 94.9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 |
| *10* | 539 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.5 | 0.0 | 0.0 | 95.4 | 1.1 | 0.4 | 1.5 | 0.2 |
| *11* | 2257 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 4.0 | 0.0 | 0.2 | 95.6 | 0.0 | 0.0 | 0.0 |
| *12* | 1047 | 1.8 | 0.0 | 0.0 | 0.0 | 0.2 | 0.0 | 2.6 | 1.1 | 0.0 | 0.2 | 0.0 | 91.0 | 3.1 | 0.0 |
| *13* | 24,163 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 99.5 | 0.0 |
| *14* | 1101 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 6.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.8 | 93.2 |

* Integers down the left margin and across the top of the table are the actual and estimated vegetation types, respectively. *N* refers to the number of test boxes classified. Numbers in the body of the table are the percent of test boxes classified as each vegetation type.

sion matrix for test pixels from the 3 × 3-pixel-box experiment. Integers down the left margin and across the top are the actual and estimated vegetation types, respectively. *N* refers to the number of test boxes classified. Numbers in the body of the table are the percent of test boxes classified as each vegetation type.

Twelve of the fourteen templates achieved more than 90% correct classification. Many of the errors can be attributed to the fact that the two templates with the largest numbers of errors had the second and fourth smallest numbers of training inputs. In general, we have observed that the larger the number of training inputs relative to the size of the sample population, the better the performance.

In addition, the template trained to identify the aspen/conifer mix (class 6) mislabeled 12.2% of these boxes as aspen (class 8) and 5.1% of them as mixed conifer (class 4). These classifications would seem to be reasonable mistakes, given that the aspen/conifer mix has a spectral signature that is a hybrid of the aspen and mixed-conifer classes.

Finally, the template trained to detect non-vegetated locations also had a score below 90%. In this case most of the errors were associated with naming the non-vegetated locations (class 7) as dry meadow (class 13) and alpine (class 14). All three groundcover types were characterized by high intensity values in all bands, making the spectral signatures similar and more difficult to distinguish.

Besides accuracy, the efficiency of converging to a satisfactory solution is crucial to the usefulness of an interactive learning environment. Learning must occur reasonably quickly. As Table 1 shows, the number of examples and counterexamples selected for training was typically a small subset of all training data. For the sake of comparison, Augusteijn et al. used all available training data five times to train their neural net classifier.

## Discussion

Data-handling capabilities of computers and communications networks are rapidly outpacing those of humans. Databases will soon contain more data than humans can scan in any useful block of time. This data explosion is fueling the development of computer tools that can facilitate database searches.

We have designed a way for a computer algorithm to infer the search intentions of a user on the basis of the selections made by the user. The process occurs as a two-way dialogue of feedback. TIM is relatively unconstrained by predefined query languages or menus of samples, allowing it to be adapted to search tasks and image data that were not specifically anticipated in its design.

More traditional approaches to machine learning might have been used in the TIM framework in place of functional-template learning. Functional-template learning, however, has several advantages that make it particularly well suited to the TIM environment:

1. Preliminary comparisons suggest that functional-template learning can compete with neural nets in classification performance.
2. Search-tool generation is rapid—typically from about a second to a minute on a Sun SPARCstation 10, depending on the number of input spectral bands and registered images. This speed enables an interactive dialogue between user and computer.
3. Because functional templates consist of scoring functions implemented as lookup tables, and because functional-template learning uses only the most discriminating attributes, search tools constructed in TIM are a computationally efficient means of searching images pixel by pixel.
4. Functional templates are easily interpreted and edited because they are composed of a series of scoring functions. While we cannot accurately predict how changing the weight at a particular node of a neural net will affect classification performance, scoring functions can be edited with readily predictable results. The ability to visualize and edit scoring functions is a built-in part of the TIM environment.
5. Functional-template learning is capable of rapid convergence, which generates reasonable performance with even a handful of input examples and counterexamples.
6. Functional-template learning does not assume any kind of normal distribution of the data.

For several reasons, the search tools generated by TIM ought to be successful as independent agents exploring image databases on the behalf of users. First, the search tools can be constructed to match a user's intentions precisely, increasing the likelihood that the agent will retrieve what the user really wants. Second, the functional template and the subroutines used to generate the various data transformations are conceptually simple and could be exported not as code but as data. This simplifies security issues related to allowing the remote installation of a computer program by an outside user who might have malicious intentions. Finally, as the agent makes mistakes, the user can continue to modify search performance and not be forced to choose between accepting inadequate results or starting over and building a new search tool.

An application that highlights the usefulness of learning from mistakes is that of prospecting. Suppose a geologist finds an unusual mineral deposit in the field. Using hyperspectral image data, the geologist might indicate the location of the deposit, introducing it as an example to TIM. From that single example, the constructed search tool would create an interest image highlighting the known deposit and any other locations with similar characteristics. The geologist could then visit some of these locations, either confirming or denying the search tool's assessment. Feedback in the form of new examples or counterexamples would uncover the best discriminants and lead to a reliable means of finding other mineral deposits.

An interesting feature of TIM that is worth further study relates to the selection of training data. Most learning algorithms are trained with a selection of inputs arbitrarily chosen and organized prior to learning. Selection of a next input is unrelated to how well the discrimination task has been learned. In contrast, the learning environment in TIM promotes a more directed selection of inputs. The use of interest images as a means of visual feedback allows the user to select new inputs that will most rapidly correct errors

in discrimination. If a solution exists, convergence should be rapid. Moreover, a more biased selection of training data, with inputs chosen by the user to focus on the most difficult discrimination boundaries in an $n$-dimensional variable space, may be able to generate search tools that are better classifiers than the tools that would be obtained by random choices of inputs.

## Summary

We have presented a prototype workstation environment that enables users to create customized image-database search tools capable of learning from experience. The environment is relatively unconstrained by a predefined query language or menus of samples, making it more flexible than other existing approaches to content-based image retrieval. The search tools generated are based not on a single example, but are derived from a user-identified set of examples and counterexamples, allowing TIM to discover complex solution spaces. As a result, search tools created in this manner rival the performance of one-of-a-kind, custom-built classification algorithms and yet can be built by users with no programming skills.

In addition to the examples presented, TIM is being used to learn signatures of vegetation in Airborne Visual and Infrared Imaging Spectrometer (AVIRIS) data, to identify sensor artifacts by texture in Doppler weather radar data, and to discriminate vehicular targets from clutter in synthetic-aperture radar data. Aside from content-based image retrieval, potential TIM applications include quantitatively analyzing images and trends; generating metadata for annotating images; prioritizing or reducing data in bandwidth-limited situations; and building components of larger, more complex computer-vision algorithms.

## Acknowledgments

This research is funded by the U.S. Air Force and the Remote Sensing and Geographic Information Systems Center of the U.S. Army Corps of Engineers.

# REFERENCES

1. "Broader Involvement of the EOSDIS Community Is Needed," U.S. General Accounting Office Report GAO/IMTEC-92-40, May 1992.
2. "NASA's EOSDIS Development Approach Is Risky," U.S. General Accounting Office Report GAO/IMTEC-92-24, Feb. 1992.
3. *SPIE: Storage and Retrieval of Still Image and Video Databases IV*, **2670**, *San Jose, 1–2 Feb. 1966.*
4. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by Image and Video Content: The QBIC System," *Computer* **28**, 23 (Sept. 1995).
5. D. Landgrebe and L. Biehl, *An Introduction to MultiSpec* (School of Electrical Engineering, Purdue University, W. Lafayette, IN, 1994).
6. R.L. Kettig and D.A. Landgrebe, "Classification of Multispectral Image Data by Extraction and Classification of Homogeneous Objects. *IEEE Trans. Geosci. Electron.* **GE-14**, 19 (1976).
7. S.I. Gallant and D.M. Fram, "Image Retrieval Using Image Context Vectors," *SPIE* **2368**, 2 (1994).
8. S.I. Gallant and M.F. Johnston, "Image Retrieval Using Image Context Vectors: First Results," *SPIE* **2420**, 82 (1995).
9. R.W. Picard and T. Kabir, "Finding Similar Patterns in Large Image Databases," *Proc. IEEE Conf. on Acoustics, Speech, and Signal Processing* **5**, *Minneapolis, 27–30 Apr. 1993,* p. V161.
10. V.E. Ogle and M. Stonebraker, "Chabot: Retrieval from a Relational Database of Images," *Computer* **28**, 40 (Sept. 1995).
11. R.L. Delanoy, "Supervised Learning of Tools for Content-Based Search of Image Databases," *SPIE* **2670**, 194 (1996).
12. R.L. Delanoy, J.G. Verly, and D.E. Dudgeon, "Machine Intelligent Automatic Recognition of Critical Mobile Targets in Laser Radar Imagery," *Linc. Lab. J.* **6**, 161 (1993).
13. R.L. Delanoy and S.W. Troxel, "Machine Intelligent Gust Front Detection," *Linc. Lab. J.* **6**, 187 (1993).
14. R.L. Delanoy and S.W. Troxel, "Automated Gust Front Detection Using Knowledge-Based Signal Processing," *IEEE 1993 Natl. Radar Conf., Boston, 20–22 Apr. 1993,* p. 150.
15. M.M. Wolfson, R.L. Delanoy, B.E. Forman, R.G. Hallowell, M.L. Pawlak, and P.D. Smith, "Automated Microburst Wind-Shear Prediction," *Linc. Lab. J.* **7**, 399 (1994.)
16. R.L. Delanoy, J.G. Verly, and D.E. Dudgeon, "Pixel-Level Fusion Using Interest Images," Technical Report TR-979, MIT Lincoln Laboratory, Lexington, MA, May 1993, DTIC #AD-A266640.
17. R.L. Delanoy, J.G. Verly, and D.E. Dudgeon, "Functional Templates and Their Application to 3-D Object Recognition," *Proc. 1992 Int. Conf. of Acoustics, Speech, and Signal Processing* **3**, *San Francisco, 23–26 Mar. 1992,* p. 141.
18. R.L. Delanoy and J.G. Verly, "Computer Apparatus and Method for Fuzzy Template Matching Using a Scoring Function," U.S. Patent No. 5,222,155, June 1993.
19. N. Draper and H. Smith, *Applied Regression Analysis* (John Wiley, New York, 1966), pp. 167–195.
20. T.P. Huber and K.E. Casler, "Initial Analysis of Landsat TM Data for Elk Habitat Mapping," *Int. J. Remote Sensing* **11**, 907 (1990).
21. M.F. Augusteijn, L.W. Clemens, and K.A. Shaw, "Performance Evaluation of Texture Measurements for Ground Cover Identification in Satellite Images by Means of a Neural Network Classifier," Technical Report EAS-CS-93-8, University of Colorado, Colorado Springs, CO, Oct. 1993.
22. S.E. Fahlman, "The Cascade-Correlation Learning Architecture," in *Advances in Neural Information Processing Systems* **3**, eds. R.P. Lippmann, J.C. Moody, and D.S. Touretzky (Morgan Kaufmann, San Mateo, CA, 1990), pp. 190–196.

**RICHARD L. DELANOY**
is a staff member in the Machine Intelligence Technology Group. His work has focused on various aspects of computer vision and machine learning, including automatic target recognition systems and automatic computer algorithms for detecting and tracking hazardous weather conditions. From 1980 to 1983, he was a research scientist in the psychology department at the University of Virginia, where he investigated the biochemical correlates of learning and the effects of stress-related hormones on electrophysiological models of memory. Before joining Lincoln Laboratory in 1987, he worked for GE Fanuc Automation N.A., Inc., as a software engineer developing numerical and programmable controllers for manufacturing automation. Dick received a B.A. degree in biology from Wake Forest University, a Ph.D. degree in neuroscience from the University of Florida College of Medicine, and an M.S. degree in computer science from the University of Virginia. He was a National Science Foundation Predoctoral Fellow and a National Institute of Mental Health Postdoctoral Fellow.