Chapter 9

The RaBI retrieval system

In the previous chapters, we introduced a decision-theoretic formulation for the retrieval problem and shown that it has interesting properties as a unified solution to the visual recognition and learning problems. In this chapter, we discuss the implementation of a CBIR system based on this decision-theoretic formulation. The "Retrieval as Bayesian Inference" (RaBI) system is an image search engine designed to operate under the query by example search paradigm. It implements most of the retrieval functionalities discussed so far, including support for multimodal queries (visual and text), short- and long-term learning, local queries, and both positive and negative examples. We next discuss the implementation of each of these functionalities in some detail.

9.1 System overview

Figure 9.1 presents a generic block diagram of RaBI. The system consists of three software modules: a Java client, a SQL server, and an image server. The Java client implements the user interface and communicates with the image server through a simple custom protocol. It is a light-weight applet that can run on any platform either in stand-alone mode or from an Internet browser. Once connected to the image server, the client allows the user to select a database and perform text-based searches, visual searches, or a combination of both.

The SQL server implements two functionalities. First, it allows one level of indirection

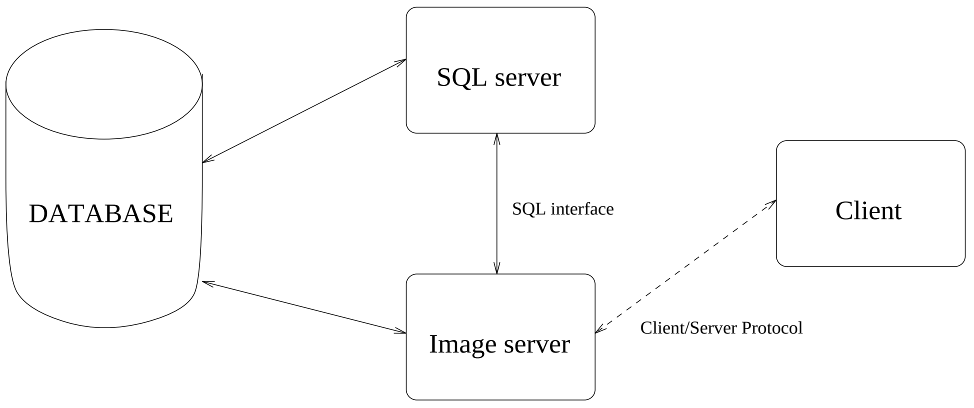

Figure 9.1: Block diagram of the RaBI retrieval system.

with respect to the physical location of all the files on the database. This simplifies the process of relocating all the data if that becomes necessary. Second, it supports simple text-based queries that can be performed very efficiently using standard SQL indexing mechanisms. The image server accepts queries from the client, performs the searches, and transmits back the search results. Some of the text-based searches are dispatched to the SQL server. We next provide a more detailed description of each of the modules.

9.1.1 Client

The client consists of two major modules: a graphical user interface (GUI) and the client/server protocol (CSP) interface. The GUI is presented in Figure 9.2. The larger image displayed in the top left corner is the *query canvas*. By dragging the mouse over a given region of the image, the user can select that region as a positive (by pressing the left button) or negative (right button) query example. Boundaries of the image blocks in the positive examples are displayed in green, while those in negative examples are shown in red.

Three text boxes are shown below the query canvas. The one on the left displays the text keywords known to the system, the one in the middle shows which vision modules are available, and the one on the left displays the visual concepts on which the system has been trained. By selecting the “Show me” or ”Not interested in” buttons the user can select any of the entries on the text boxes as positive or negative text examples, respectively. The

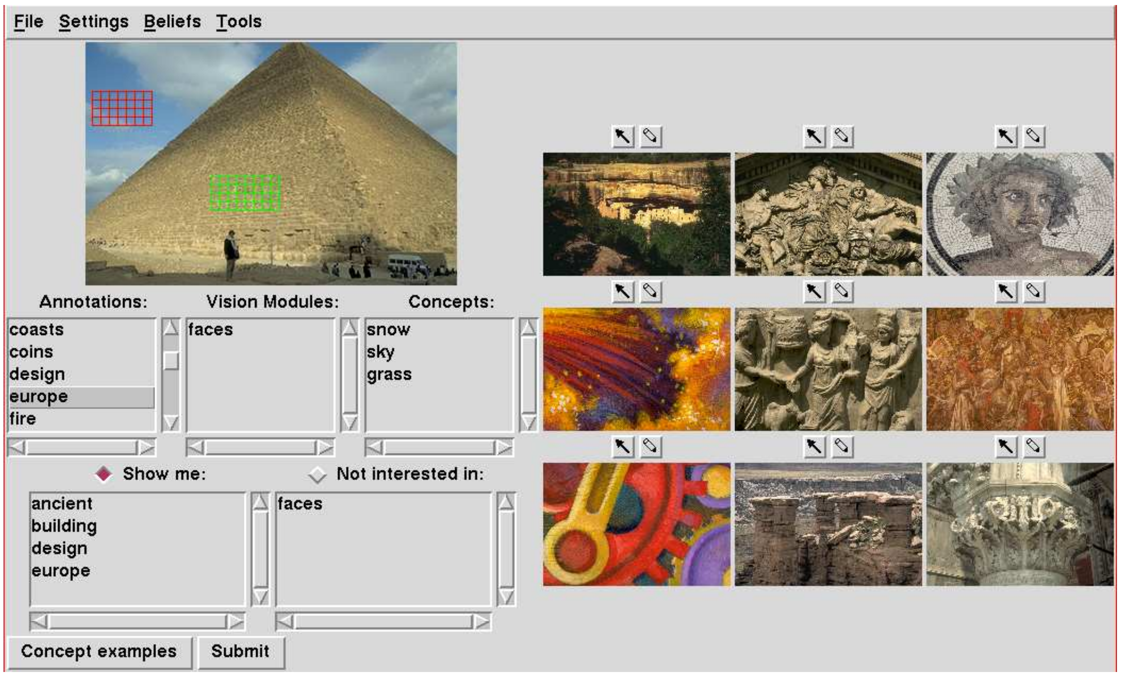

Figure 9.2: The GUI of the RaBI system.

selected keywords then appear in the two boxes in the bottom-left.

When the user presses the "Submit" button, the query (consisting of all the keywords, the lists of coordinates of the selected image blocks, and the identifier of the query image) is passed to the CSP interface that encodes all the information and dispatches it to the image server. The server replies with a list of URLs that point to the images that best satisfy the query. This response is decoded by the CSP interface and the top $M \times N$ images are displayed on the right half of the GUI, as illustrated by Figure 9.2. The variables M and N can be defined by the user. Two buttons are displayed above each image. By pressing the button containing an arrow, the user can make the image appear in the query canvas and use it as a basis for subsequent queries. By pressing the other button (pencil icon), the user can change the text annotations associated with the image (e.g. give less weight to a particular keyword).

Whenever visual examples are selected in the query canvas, it is possible to associate a concept keyword with them. For this, the user simply needs to press the "Concept examples" button. A dialog box then presents a list of the known concepts and the user has the option of either selecting one of these concepts or defining a new one. The concept examples are then sent to the image server and passed on to the concept learning algorithm.

Finally, the menus on the top-left corner of the interface allow a series of routine operations, such as clearing the short-term learning memory (when the user is starting a new query), opening and closing databases, and setting the values of the various parameters of the retrieval algorithm (e.g. the number of subspaces to be considered and the decay factor α for learning).

The CSP is a simple communications protocol consisting of 1) a control sequence that indicates what type of operation the user is carrying out (e.g. query, opening new database, providing concept examples, etc.), and 2) a collection of data fields that depends on the type of operation.

9.1.2 SQL server

The SQL database contains pointers to all the image information and implements a preliminary image classification according to database and image class. It associates three information components to each image: a database name, an image class, and two strings indicating the image URL and the physical image location on disk.

The database classification establishes a very coarse image grouping. Current databases are the ones discussed in the previous chapters: Corel, Columbia, Brodatz, Columbia mosaics, and Brodatz mosaics. Each database defines a set of keywords that allow finer image grouping. These appear in the "Annotations" box of the GUI of Figure 9.2.

Image classes allow the user to restrict the query to a subset of the database, and can be defined in several ways. For some databases, images are naturally grouped into obvious classes. For example, Brodatz (Columbia) contains various views of each texture (object), while Corel images are organized into thematic directories, each containing 100 images. In these cases, textures, objects, and themes constitute natural classes for image grouping. If a text model is available, classes can also be derived automatically, by associating each image to the keyword that explains it with highest probability. Finally, classes can always be defined manually.

Despite the support for database and image classes, there is no obligation on the part of the user to use this information at any point of the retrieval operation. Instead, the classification should be seen as a filtering mechanism to quickly eliminate many images that are a priori known to be irrelevant for the search, and direct the attention of the retrieval system to the images that matter. Since changing the database in the middle of a query may lead to inconsistencies in learning, this mechanism can only be used at the beginning of a visual retrieval session. If it is, this initial database/class query is based on standard SQL indexing structures and, therefore, very efficient.

9.1.3 Image server

The image server has two main operation modes. In the database selection mode, it 1) provides the client with a list of available databases and image classes, 2) receives a text-based query relative to these, and 3) dispatches this query to the SQL server. The SQL server replies with the list of internal file paths and URLs associated with the images to be considered in the subsequent visual queries. The image server then assembles all the data-structures required for these queries and enters the visual query mode.

Seven different data-structures are assembled. The first is a collection of matrices containing the mixture parameters that are required to evaluate the likelihood of the visual query features according to (2.14). Two others are a *text probability matrix*, and a *concept probability matrix*. In these matrices, each row corresponds to a textual or visual concept and each column corresponds to one image, the entry i, j containing the parameter $\log p_{i,j}$ required by (8.5). Textual concepts are keywords derived either from the text-model discussed in section 8.4 or from vision sensors. For example, a face detector can be applied to each image in the database, assigning the "face" keyword to each image with a certain probability. Visual concepts are learned over time as discussed in section 8.4.2.

The remaining data-structures are a *visual likelihood buffer*, a *concept likelihood buffer*, a *text likelihood buffer*, and a *short-term memory buffer*. The first three hold information relative to the current iteration of the retrieval process, while the fourth is reset only when the user starts a new query. Their role is illustrated by Figure 9.3, which provides a graphical depiction of the inner workings of the image server.

In response to a visual query, the image server uses the mixture parameters associated with each image in the database and (2.14) to evaluate the log-likelihood of the visual features. The resulting log-likelihoods are stored in the visual likelihood buffer. The textual component of the query is then analyzed. The rows of the text and concept likelihood matrices associated with the keywords and concepts selected in the query are added and stored in the text and concept likelihood buffers, respectively. The log-likelihoods associated with the current query are then evaluated by adding the three buffers. The following step is to combine these log-likelihoods with the log-posterior probabilities given the past interaction, that are stored in the short-term memory buffer. This is done according to (7.5), i.e. by weighting the entries in the two buffers according to the memory decay factor α. The result is a set of log posterior probabilities that are passed to an *image ranking* module.

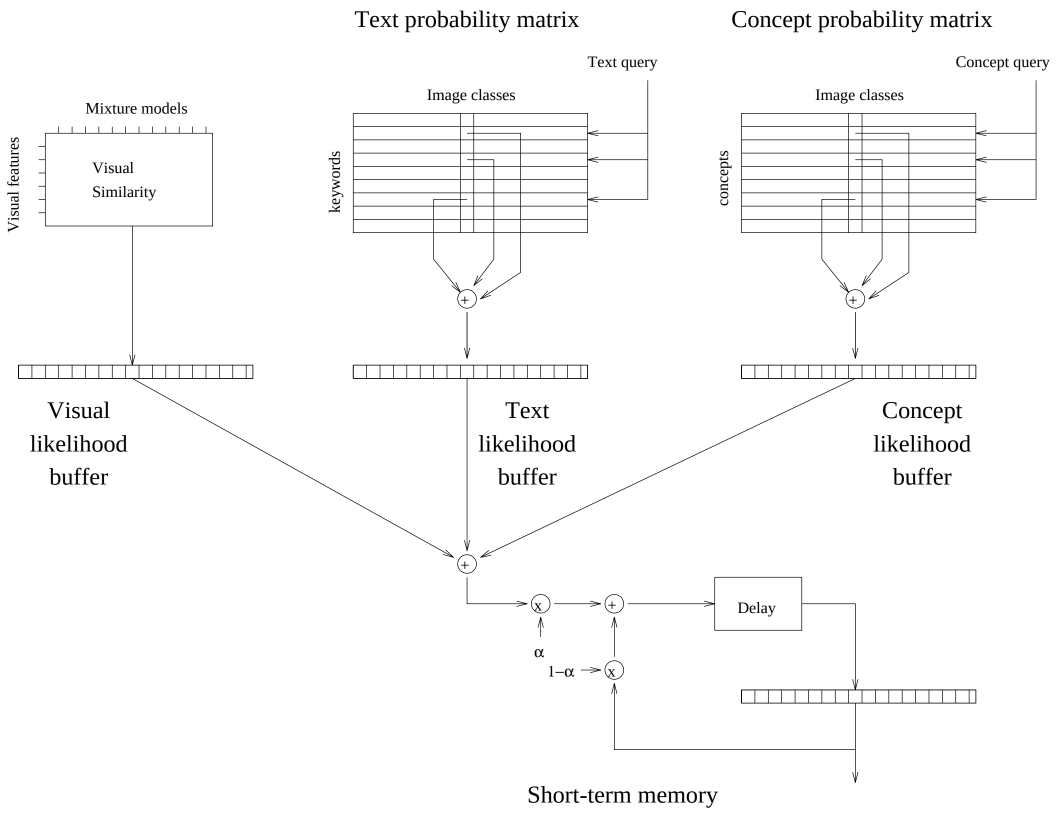

Figure 9.3: Block diagram of the image server.

As described, the process only accounts for positive examples. Extension to negative examples is straightforward, simply requiring the duplication of the four buffers and application of (7.7) instead of (7.5). As discussed in section 7.4, the main difference occurs in the image ranking module. When only positive examples are considered, this module simply orders the images by decreasing log-posterior probabilities. When negative examples are present, the top N matches according to the positive examples are then re-ordered by decreasing log posterior odds-ratio. The parameter N can be modified by the user, the default being $N = 50$.

Given the list of top matches, the image server requests the corresponding URLs from the SQL database and transmits them to the client using the CSP. The client simply displays the images and waits further instructions from the user, upon which the entire query process is repeated.

9.2 Implementation details

We conclude this chapter by discussing some implementation details of the RaBI system. This discussion is intended mostly for those interested in replicating portions or the entirety of the work described in the thesis. Although many of the details have already been mentioned in the previous chapters, we believe that it is useful to present a cohesive summary of the most important points. We concentrate on the three components of the retrieval architecture: feature transformation, feature representation, and evaluation of image similarity.

Before proceeding to describe the details of each component we note that, even though the theoretical formulation developed in the thesis is valid for any grouping of the database images into image classes, in RaBI each image is currently considered as a class by itself for the purposes of visual retrieval. In the future, we plan to investigate the impact of more sophisticated hierarchical database organization strategies in the retrieval efficiency. A preliminary step in this direction has, indeed, already been presented in [188].

9.2.1 Feature transformation

Given a database image, feature transformation currently consists of the following steps.

- Extract all 8×8 image blocks separated by d pixels in each dimension. d can be defined by the user, the default value being $d = 4$.
- Compute the DCT for each block according to (4.11).
- Vectorize the 2-D DCT array into a row vector. While this can be done in several ways, RaBI relies on the coefficient scanning mechanism defined by the MPEG [63] standard.
- If there are various color channels, interleave the corresponding vectors. For example, if there are three color channels, the first three coefficients of the interleaved vector contain the first coefficient from each of the three color channels.
- RaBI currently uses the YBR color-space defined by MPEG, but this selection has not been subject to detailed scrutiny. Better results may be possible with a perceptual color space, e.g. HSV [163].

9.2.2 Feature representation

In RaBI, feature representation is based on Gaussian mixtures with a fixed number C of classes. The default value is $C = 8$, but can be modified when the database is initialized. Several methods have been proposed in the literature to automatically determine the number of mixture components in each model. These include *minimum description length* [146], *Akaike's information criteria* [81] and the *Bayesian information criteria* [23] among others. Because these methods increase the complexity of density estimation by a significant amount, we decided to use a fixed number of mixture components. Automatic determination of the number of components may, however, be included in future versions of RaBI.

All Gaussian mixture parameters are estimated using the EM algorithm [36, 143, 13]. The implementation is fairly standard, the only details worth mentioning are the following.

- All Gaussians have diagonal covariances.
- In order to avoid singularities, a variance limiting constraint is applied. This constraint

places a minimum value of 0.1 on the variance along each dimension of the feature space.

- Initialization is performed with a vector quantizer designed by the LBG algorithm, using a variation of the cell splitting method described in [58].
- In order to split a cell, 1) we compute the (diagonal) covariance matrix of all the points that land inside it, and 2) replicate its centroid, adding to the replica a perturbation in the direction of largest variance. The perturbation is the square root of this largest variance.
- Given the codebook obtained with the LBG algorithm, we compute covariances and mixing probabilities for each of the quantization cells. These, together with the centroids contained in the codebook, provide an initial estimate for the mixture parameters.
- This initial estimate is refined with EM. Since significant changes usually only occur in the first iterations, we limit these to eight.

9.2.3 Evaluation of image similarity

The evaluation of image similarity consists of the following steps.

- Given the list of query block coordinates, extract these blocks from the query image.
- Compute the DCT, vectorize, and interleave the color channels of each block as described in section 9.2.1.
- Evaluate the likelihood under each database mixture model according to (2.16).
- To prevent numerical underflow, the likelihood of each feature vector is replaced by a fixed threshold T before taking the logarithm if its value is smaller than T. The default value is $T = 10^{-300}$.

9.2.4 Vision modules

Currently, only a face detection module is implemented in RaBI. This face detector consists of a set of libraries that were provided by Henry Rowley at CMU. It is described in [148]. Since vision sensors can be applied to the images in the database off-line, RaBI is highly extensible. In fact, the system does not even need to know the implementation details of the sensors, since all that is required is the table of probabilities generated by these. We therefore expect to incorporate other vision sensors in RaBI in the future.