UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EDUARDO MAYER TERROSO

An Image Organizer with Content-Based Image Retrieval

Undergraduate Thesis presented in partial
fulfillment of the requirements for the degree of
Bachelor of Computer Science

Prof. Dr. Marcelo Johann
Advisor

Porto Alegre, June 2008

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
CBIR	Content-Based Image Retrieval
FV	Feature Vector
GUI	Graphical User Interface
MPEG-7	Moving Pictures Expert Group Multimedia Content Description Interface
QPE	Query by Pictorial Example
RF	Relevance Feedback
SOM	Self-Organizing Map
TS-SOM	Tree-Structured Self-Organizing Map

LIST OF FIGURES

ABSTRACT

In the last years, digital cameras and image sharing sites became very popular. That allied to the constantly increasing capacity of storage devices made personal photo collections become huge. However, to manage these collections without an effective tool is almost impossible. Presently, there is a plenty of photo organizing software available, but almost all of them rely only on textual metadata manually added to the images. Many researches have been done addressing the problem of searching images in unannotated databases and many Content-Based Image Retrieval systems were developed. However, most of them are for online searches in existing databases or to find duplicated images in a hard disk. This work is about the implementation of LACAIO ("Lacaio is A Cbir Aided Image Organizer"), which is an image organizer that allows user to search images by visual similarity. LACAIO utilizes four Tree-Structured Self-Organizing Maps to index images by four different feature vectors. Even with a very simple GUI, it allows search by keywords, search by sketch and an iterative interactive search by similarity with relevance feedback.

Keywords: CBIR, query by pictorial example, query by sketch, TS-SOM, relevance feedback.

Um Organizador de Imagens com Busca Baseada em Conteúdo

RESUMO

Nos últimos anos, câmeras digitais e sites de compartilhamento de imagens se tornaram muito populares. Isso aliado ao constante aumento de capacidade dos dispositivos de armazenamento fez com que as coleções de fotos se tornassem imensas. Entretanto, gerenciar essas coleções sem uma ferramenta efetiva é quase impossível. Atualmente, existem muitos organizadores de imagens disponíveis, mas quase todos se baseiam somente em meta-dados textuais adicionados às imagens manualmente. Muitas pesquisas têm sido feitas a respeito do problema de procurar imagens em bancos não anotados e muitos sistemas de CBIR (*Content-Based Image Retrieval*) foram desenvolvidos, mas a maioria deles é para buscas *online* em bancos de dados já existentes ou para encontrar imagens duplicadas em um disco rígido. Este trabalho é sobre a implementação de LACAIO ("Lacaio is A Cbir Aided Image Organizer"), que é um organizador de imagens que permite que o usuário faça buscas baseadas em similaridade visual entre as imagens. LACAIO utiliza quatro TS-SOMs (*Tree-Structured Self-Organizing Maps*) para indexar imagens por quatro diferentes vetores de características. Mesmo tendo uma interface muito simples, ele permite busca por palavras-chave, busca por desenho e uma busca interativa e iterativa por similaridade com *feedback* de relevância.

Palavras-chave: busca baseada em similaridade visual, organizador de imagens.

1 INTRODUCTION

In last years, digital cameras and image sharing sites became very popular. Today, every user can easily have gigabytes of stored images, but manage such large collections is quite a challenge.

Presently, there is a plenty of photo organizing software available, but almost all of them rely only on textual metadata manually added to the images. Photo or image organizers usually provide means to organize images in overlapping categories, unlike file system hierarchy. Most of them also allow users to search images by keywords and add new keywords to images. These software are useful tools that provide an interface much richer than a general use file manager does, but they are not enough.

As the capacity of hard disks and flash cards increases, people become less selective about what to photograph and what to store. It is easy to have thousands of photos, but it is not easy to annotate thousands of photos. To easily retrieve images using a regular photo organizer, the user has to spend a long time classifying and adding keywords to each image and it is unlikely that most common users would do that. The consequence is that these software become of very little use for most people.

Many researches have been done addressing the problem of searching images in unannotated databases and many CBIR software were developed, but most of them are not intended for common users. Most CBIR systems available are for online searches in existing databases. There are software to find duplicated images that use CBIR techniques (VISUALPHOTOCOMPARE, 2004) and even a framework for CBIR (GIFT, 2000), but only two image organizers that use CBIR were found: Octagon (OCTAGON, 2005) and imgSeek (IMGSEEK, 2007).

1.1 Objectives

The aim of this work is to develop an image organizer to organize partially annotated personal photo collections. The target user is a common user, that is, no technical knowledge, talent, or special skill is assumed and thus the GUI must be as simple as possible.

1.2 Results

The LACAIO system was developed and some tests were made to evaluate it. The name “LACAIO” derives from the recursive acronym “Lacaio is A Cbir Aided Image Organizer”. The tests show that the system is promising and can be very helpful to manage unannotated or partially annotated collections, but to be ready to end users it would need further work.

1.3 Organization

Chapter 2 defines some concepts involved in Content-Based Image Retrieval and discusses design issues of CBIR systems. Chapter 3 gives an overview about Self-Organizing Maps and Tree-Structured Self-Organizing Maps, the structure used by LACAIO to index image features.

Chapter 4 is about the LACAIO system itself. The system is compared to existing systems and its implementation and user interface are explained in detail. Chapter 5 describes the four feature vectors used by LACAIO and Chapter 6 presents the results of the evaluation of the system.

2 CONTENT-BASED IMAGE RETRIEVAL

There are basically three ways to retrieve images from a database:

1. **Free Browsing**: The users navigate through the database, possibly categorized, until they find what they are looking for.

2. **Text Based**: The users enter words or phrases and the system shows images related to these words or phrases.

3. **Content-Based**: The search is based on information automatically extracted from the images, ranging from low level features to complex hierarchical descriptions of the items contained in the image. The query method may vary, but one commonly used is specifying an example image and the system looks for the images that are most similar to it.

Free browsing is adequate when the database is small and the user is eventual, but with large databases it is impractical and having to navigate through the whole database every time you look for an image is frustrating. To enable text based searches, the images in the database must be annotated (i.e., adding textual metadata to each image). This process is tedious and time-consuming, especially for large databases. The quality of the search is only as good as the metadata added in this process and it is often incomplete and inaccurate.

Neither of these two techniques is easily scalable to huge collections. This is why many researchers started to investigate ways to retrieve images based only on automatically extracted information. Content-Based Image Retrieval (CBIR) is using computer vision techniques to retrieve images from a database based on their visual content. Automatically extracting the semantic information from the images is currently an unsolved problem, since it would require a strong AI system capable of understanding abstract ideas and thinking like the human mind. Rather than trying to extract semantic information, most CBIR systems use low level features like statistical data about color, texture and shape.

2.1 Design Issues

In the last years the research field of CBIR has been very active as can be seen in (DATTA et al., 2008). Despite all the progress made, there are many open issues. When designing a CBIR system, some questions arise:

2.1.1 Which features to use?

This is a critical part of the system. The features extracted are all the information the system will have about the images. The chosen features must be meaningful enough to enable the system to measure similarity between images. A human user will probably consider two images similar based on high level semantic features. The lack of coincidence between the information contained in the low level features automatically extracted from visual content and the interpretation this same visual content would have to a user is referred to in the literature as *semantic gap* (SMEULDERS et al., 2000).

CBIR systems usually use color and texture features, some also extract information about frequency and shape. More specific systems might benefit from specialized features like face detection related information, for instance.

Choosing the right features to satisfactorily capture relevant information about images while tolerating noise that would be tolerated by a human is not an easy task. Besides that, the features must be represented in a way that allows fast similarity measurement to achieve satisfactory results in a reasonable time.

Each system may have its own features represented in its own way, which makes difficult to evaluate and compare results from different systems. To try to standardize the multimedia content description, the MPEG-7 (MPEG-7, 2004), or formally "Moving Pictures Expert Group Multimedia Content Description Interface", standard was defined. It defines general purpose content descriptors for audio, video and still images. Some CBIR systems already adopted standardized features defined by MPEG-7.

2.1.2 Which measure of similarity to use?

Without a measure of similarity, features are completely useless. The system must be able to somehow determine how similar two given images are. Typically, correlated features (e.g., the bins of a histogram) are organized in feature vectors. One possible measure is the Euclidean distance in the feature vector space. The main advantage of this approach is its simplicity, but it does not take human visual perception into account. A similarity measure based on human perception is proposed in (SAHA; DAS; CHANDA, 2007).

2.1.3 How to index the features?

To search over the whole database processing each feature vector of each image for each query is extremely inefficient, thus unacceptable for huge databases. It is necessary to index the feature vectors to reduce the complexity of the search, but to index points in a multidimensional vector space is not trivial.

There is a wide range of distinct techniques for indexing images based on their features. In (JORMA LAAKSONEN MARKUS KOSKELA, 2002), Scalar Quantization, Vector Quantization and Self-Organizing Maps techniques are compared and according with this work, SOMs yield better results.

2.1.4 How to weight the features for a given query?

The system needs to know which features are more relevant for each query. What is very important in one query may be completely irrelevant in the next one and the system must know how to deal with this. One possibility is to ask the user to explicitly weight the low level feature vectors, but this may be a difficult task even for technical users. Other possibility is to use machine learning techniques to infer these weights based on

user relevance feedback.

2.1.5 Which types of queries should be supported?

Defining the types of queries to support depends on the purpose of the system and the target users. To a technical user, having numerous types of queries with various options may be interesting, but for non-technical users it would likely be more confusing than helpful. These are some of the existing query types:

Free Browsing The user navigates freely through the collection. Useful when the user is not looking for something specific, but wants to have a general view of the entire collection. Clustering based on visual features can help automatically organizing the images somehow.

Category Browsing The user navigates through a hierarchy and can possibly perform new queries inside the selected subset of images. Categories can overlap if desired. Requires a time-consuming manual categorization of the images.

Query by Text Also requires a time-consuming manual annotation, but CBIR and machine learning techniques can be used to implement a semi-automatic annotation system like proposed in (WENYIN et al., 2001).

Query by Pictorial Example (QPE) Probably the most commonly used query method in CBIR. The user specifies an example image, contained or not in the database, and the system returns the images most similar to it in decreasing order of similarity. Requires an adequate sample image.

Query by Sketch Very similar to the previous, but instead of requiring an example image, the system provides an interface that allows user to draw the sample image. In order to be used by users with little or no designing skills, the drawing must be very simple, although if too simple it becomes useless.

All query types have limited use, but a good combination of methods allied with a good relevance feedback technique can be powerful.

2.1.6 How to make use of relevance feedback?

Relevance Feedback is the process of automatically adjusting a query using the user feedback about the relevance of its retrieved results. The goal is to adjust the query such that the results of the adjusted query will be more relevant than the previous ones.

Due to the problem of weighting feature vectors addressed in Sub-section 2.1.4, it is unlikely that a CBIR system will retrieve satisfactory results in its first attempt. Therefore, relevance feedback is commonly used by CBIR systems, turning the search into an interactive and iterative process.

To make use of RF, it is necessary to provide some tool that allows users to express their opinion about the relevance of the images presented so far. This feedback can be from a simple boolean value (i.e., relevant or not) to a numerical value specifying how good a retrieved image is. More critical than the GUI modifications is to adjust the query based on user feedback. This can be done, for instance, by adjusting the weights of the feature vectors. If relevant images are concentered in some feature vector space, that feature vector must be relevant to this particular query, thus its weight must be increased. On the other hand if relevant images are isolated in a feature space, maybe this feature vector must be ignored for this query.

3 SELF-ORGANIZING MAPS

A Self-Organizing Map is a type of unsupervised learning neural network. It is used in clustering and dimensionality reduction. A standard SOM has a fixed topology, usually a 2D grid, that adapts to the input space. Typically, the input space is high-dimensional and the SOM is two-dimensional. As the map adapts itself to the inputs, similar inputs are mapped to the same nodes or to neighboring nodes, reducing dimensionality and preserving the topological properties of the input space.

3.1 Network Structure

A SOM consists of a network of nodes or neurons. The most usual topology is a 2D rectangular grid, but it can be 3D, an hexagonal grid, toroidal, etc. In fact, it can be any connected graph, but a rectangular grid is very simple to implement and is useful for visualization purposes. Figure 3.1 shows representations of rectangular and hexagonal grids. Associated with each node is a feature vector of the same size of the input vectors. These vectors are the only thing that change during the training process. This way a SOM can be thought as an elastic structure, but with fixed topology in a multidimensional space. This can be seen in Figure 3.2.

Input vectors can be mapped to nodes of the map by finding the node which has the vector most similar to them. This node is called the Best Matching Unit (BMU) for that input. A SOM is useful as an index because input vectors that are close in the feature space are mapped to nodes close to each other in the map. This way, to find other vectors close to a given vector, instead of having to compare it to every single vector, one can just find the BMU for the given vector and take the nodes that were mapped to the same node.

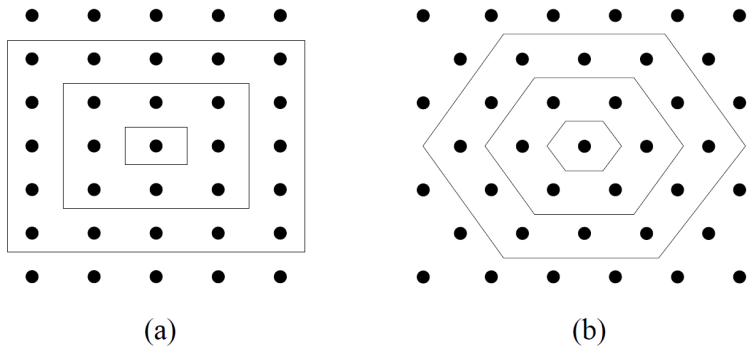

Figure 3.1: The two main SOM topologies, rectangular (a) and hexagonal (b) and its corresponding neighborhood shapes. (CHOPPIN, 1998)

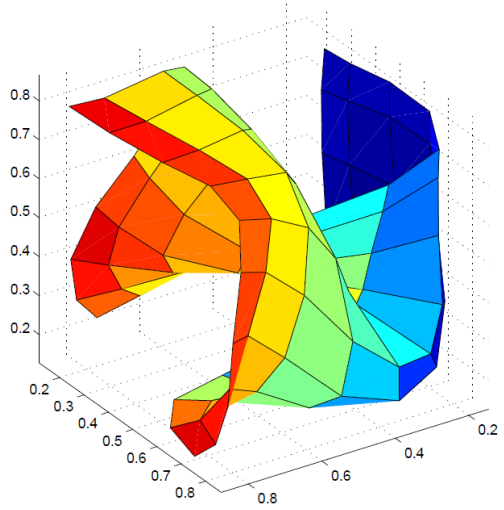

Figure 3.2: A 2D rectangular grid SOM fitted to 3D input vectors plotted in its feature vector space. (CHOPPIN, 1998)

If more vectors are needed, the vectors mapped to BMU's neighbors can be used then.

3.2 Training Algorithm

The goal of the training process is to adapt the map to the specific input vectors presented to the SOM. Sub-spaces containing more input vectors will have more map nodes and sub-spaces which don't have any input vectors most likely won't have any nodes either. The initial values in the vectors of the nodes can be random or, if there is any previous information about the inputs, they can be initialized to values close to them to increase training speed.

The training process consists of many iterations of the following steps:

1. Choose an input vector from the training set.

2. Find the closest node to the input vector (BMU).

3. Calculate the neighborhood of the BMU. This region is large when the training starts and gets smaller as the training proceeds.

4. Nodes within the BMU's neighborhood, including the BMU, are adjusted, such that they become closer to the input vector. The closer the node is to the BMU, the more it is adjusted.

Details about each step are very application dependent. The measure of distance between the input vector and the map nodes, for instance, can be any. The neighborhood depends on the size and topology of the map. The adjustment of the nodes can also be any as long as it makes the adjusted nodes become closer to the input node. One possible approach is to use a weighted mean where the weight of the input vector depends on the iteration and on the distance from the BMU. A Gaussian function can be used to determine how distance affects the weights and the neighborhood size can be determined by a threshold on the computed weight.

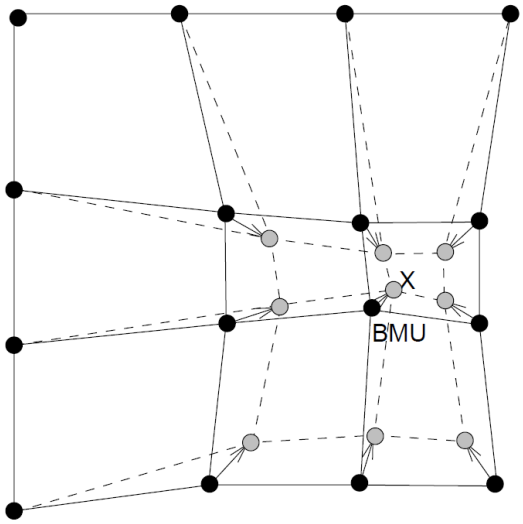

Figure 3.3: Adjustment of the BMU and its neighborhood to input vector *X*. (VESANTO et al., 1999)

Initially the adjustment and the neighborhood must be large, so that the map adapts quickly to the input data. Later these values must gradually decrease, making the map converge. There is not a general answer to how to do this. Figure 3.3 represents the adjustment of the BMU and its neighbors. The black nodes connected by solid edges are their initial positions and the gray nodes connected by dashed edges are their positions after moving toward the input vector position, marked by an *X*.

3.3 Tree-Structured Self-Organizing Maps

The standard SOM is powerful, but the training process can become very computationally expensive as the number of nodes increases. To properly use a SOM as an index, it must have a proper size. If too many vectors are mapped to the same nodes, it becomes useless. Hence, to map huge amounts of input vectors without degrading to lists, huge SOMs are necessary.

During the training phase, the most computationally expensive operation is finding the BMU. To find the closest node to a given input vector, the distance from each node to the input must be calculated. Thus, the cost of this operation increases linearly with the SOM size. The Tree-Structured Self-Organizing Map (TS-SOM) reduces the comparisons needed by using a hierarchical structure. It uses a multi-layered map to reduce complexity from O(n) to O(log(n)).

Figure 3.4 partially shows the structure of a TS-SOM in which, each layer is a rectangular grid with side twice as big as its upper layer. The dashed lines show the parent-child relation.

3.3.1 Training Algorithm

The training of a TS-SOM starts from the root node and proceeds, one layer at a time, until the layer of the leaves is reached. After a layer is trained, its nodes are frozen. Each layer is trained like a regular SOM, the only differences are the initialization and the way the BMU is found. Instead of searching over the whole layer, the previous layers are used as an index to limit the region in which the BMU will be searched.

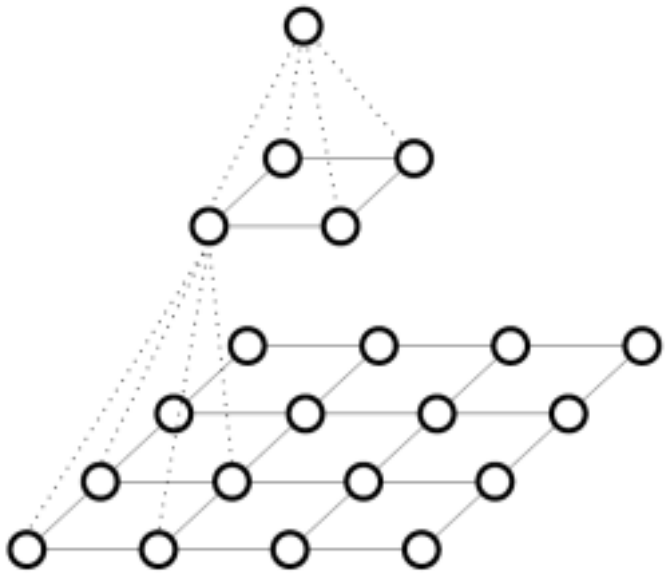

Figure 3.4: Example of a three-layer TS-SOM (PAKKANEN; IIVARINEN, 2002)

Following are the steps to train a TS-SOM:

1. Train the upper layer exactly like a standard SOM.
2. Copy the vectors of the nodes of the last trained layer to their children.
3. Move each of these children a little toward their neighbors.
4. Train the next layer.
5. Repeat the last three steps until all layers are trained.

Note that each layer after the first one is initialized based on the previous layer. This initialization considerably improves efficiency. The main difference is in the search for the BMU, which is made recursively. The region considered is only a small region around the children of the BMU of the upper layer.

4 THE LACAIO SYSTEM

In last years, many researches have been done addressing the problem of searching images in unannotated databases and many CBIR software were developed, but among them only two image organizers were found: Octagon and imgSeek. The aim of this work is to develop an image organizer to organize partially annotated personal photo collections. The target user is a common user, that is, no technical knowledge, talent, or special skill is assumed. The system must be as simple as possible to be easy to use, but must be able to handle the following situations:

- The user adds a set of images to the collection, adds a keyword to them and later, search for this keyword. The system returns the images previously associated with the keyword.
- The user wants to find a specific image, but doesn't know where it is located or which keywords are associated to it. The system must provide a way to find this image using CBIR techniques.
- Some images are associated with a keyword and the user wants to find images similar to them that should be associated to the same keyword. The system must provide a way to specify a set of images and return the images that are the most similar to them.

Additionally, it is desired that the user never be asked to perform parameter tuning like adjusting weights of feature vectors. Parameter values must be inferred from relevance feedback, if needed. From the two software cited above, neither satisfy these requirements. Both are still very immature software with very confusing GUI.

ImgSeek has too many options, too many menus and too many tabs, which makes its GUI incredibly confusing, yet it lacks some elementary features. imgSeek claims to support keyword search, but what it calls a keyword search, is actually a metadata search. One could emulate keyword search adding words to image descriptions, but it would be very inefficient and frustrating, as just one image's metadata can be edited at a time and the metadata search interface is too complex. It supports query by sketch, QPE using an external image and one round QPE with selected images, but apparently only color features are considered.

Octagon does not support pure keyword search, only a keyword filter with a very puzzling interface. It supports interactive iterative QPE with RF and also supports QPE using an external image. The relevance feedback is expressed by assigning to the returned images, one of the values: relevant, neutral or not relevant. It is not said how the RF is used, but it doesn't seem to have much effect and apparently the weight of the feature vectors is

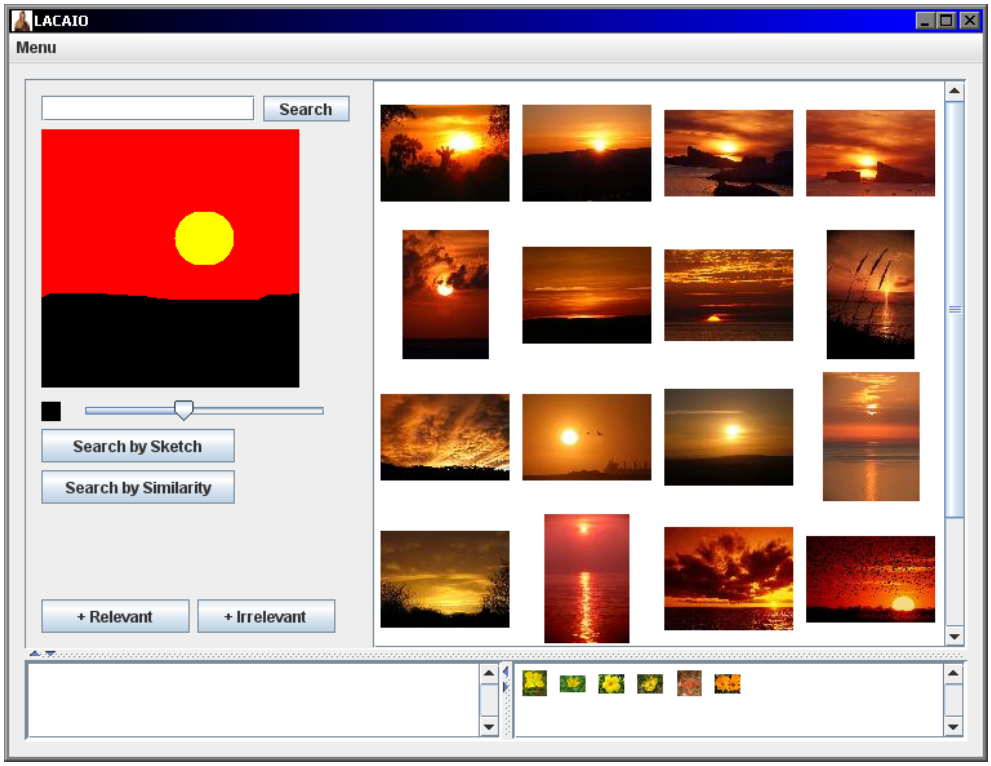

Figure 4.1: A screenshot of the main window.

explicitly defined by the user by moving a slider labeled “structure/color influence”. Octagon’s GUI is cleaner than that of imgSeek, but is also confusing and Octagon’s results seem to be less accurate.

In order to satisfy the established requirements, the following query methods were implemented:

- Search by keywords
- Search by sketch
- Interactive iterative search by examples with relevance feedback

4.1 Graphical User Interface

The GUI is as simple and intuitive as possible so that the user doesn’t need to spend time learning how to use it. The current version of LACAIO is a prototype focused on the search processes. It is not ready for end users. A final version, for real world use, would have to provide an easier way of adding new images to the collection, a way to remove missing files, etc.

Figure 4.1 shows a screenshot of the main window. The main window is composed by the *search panel* (upper left), the *result list* (upper right), the *relevant images list* (lower left) and the *irrelevant images list* (lower right).

Search By Keywords: The user types keywords separated by spaces in the text field on the upper part of the *search panel* and presses the button on its right. The system retrieves the images that are associated to each of the keywords in the query. Keywords can be associated to images, by clicking on them with the right button and selecting the proper option in the popup menu.

Search By Sketch: The user draws an sketch of the desired image in the area below the text field. Below the drawing area is a filled square indicating the selected color. Clicking on it, a dialog to change the selected color shows up. On the right of the selected color is a slider to select the diameter of the brush. After drawing, the user presses the *Search By Sketch* button, then the system retrieves the images most similar to the sketch based only on Average Colors feature vector (Section 5.1).

Search By Similarity: This is the main query method, which can be used in combination with the other two. Independently of the method used to retrieve images, the images in the *result list* can be defined as being relevant or not relevant, being then copied to the corresponding list. Images in these lists never appear as result of later queries, but the relevance feedback is only used in Search by Similarity. When the user presses the *Search by Similarity* button, the system retrieves images as close to the relevant images and as far from the irrelevant images as possible.

4.2 Implementation

The algorithm used to implement search by similarity was based on PicSOM (JORMA LAAKSONEN MARKUS KOSKELA, 1999a,b, 2002; MARKUS KOSKELA JORMA LAAKSONEN, 2003; PAKKANEN; IIVARINEN, 2002). Like PicSOM, LACAIO uses one TS-SOM for indexing each feature vector. The relevance feedback technique to "weight" the feature vectors is also based on PicSOM. The main differences between them is that PicSOM is used to query an existing online database and uses only Search by Similarity, while LACAIO combines three search methods to manage local image collections.

LACAIO was implemented in Java and uses Java DB as its DBMS. This provides portability and eliminates the need to install a DBMS, since Java DB can be embedded in Java programs. Figure 4.2 is a diagram of the structure of the database. To give a notion of the size and complexity of this implementation, it has around five thousand lines of code and thirty classes.

4.2.1 Feature Extraction and TS-SOM Training

Before start performing queries, two steps must be completed, the images must be analyzed and their features extracted and the SOMs must be trained. When images are added to LACAIO, it inserts their paths in the database, creates thumbnails to speed up the rendering and extracts their features. Feature extraction for each feature vector is explained in detail in Chapter 5. After adding the images, the SOMs are trained. When the training is finished, the images are then associated to their BMU in the bottommost layer of each SOM.

One problem with standard SOM and TS-SOM is that they do not adapt very well to large modifications in the input space. The consequence is that if a large amount of new images is added, the maps need to be trained again. In the implemented prototype, after the maps have been trained, new images cannot be added unless the maps are cleared and trained again. The first layer of the maps can have 4 or 9 nodes and the next layer always have 4 times more nodes than the previous one. The size of the first layer and the number of layers is calculated so that the bottommost layer have approximately 1 node for each 5 images. To train all four maps with the test data described in Chapter 6 takes a little less than three minutes.

The measure of similarity used when looking for the BMU was Euclidean distance. The neighborhood radius was 3 for all layers and all images were used as input 100 times

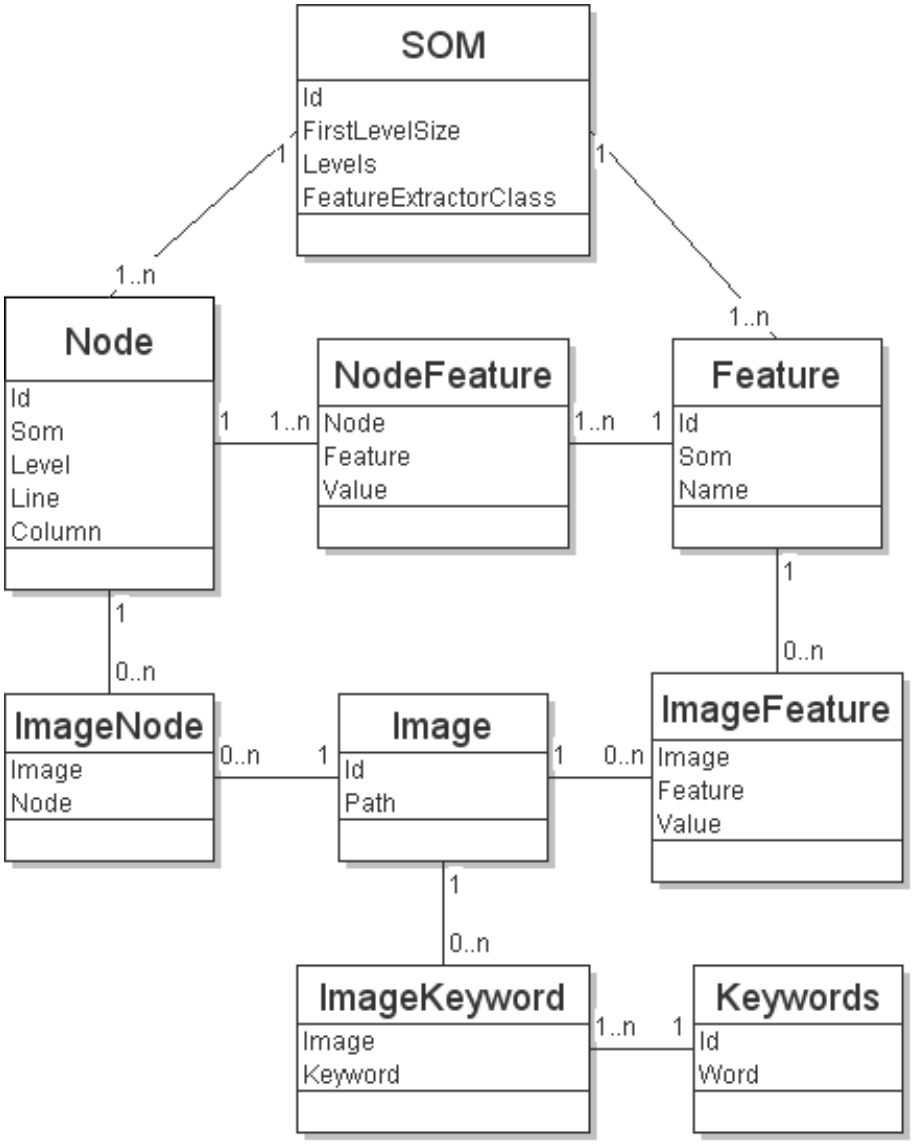

Figure 4.2: Database structure.

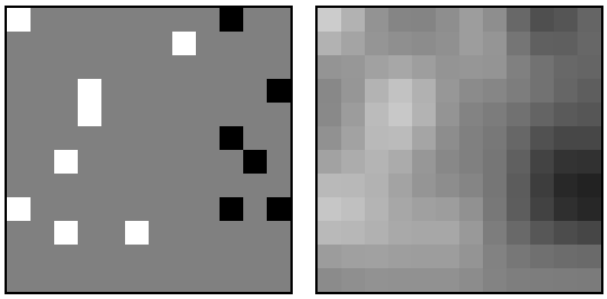

Figure 4.3: The bottommost layer of a TS-SOM after computing the scores of each node (left) and after applying a Gaussian filter (right). The darkest the node, the higher the score.

for each layer of each map, but no extensive test was made in attempt to tune these values.

4.2.2 Querying

In the following text, the term *query session* is used to refer to a sequence of interactions made by the user to achieve a goal. LACAIO is based on query sessions combining queries of different types. A query by similarity requires at least one relevant image, therefore the first query is never of this type.

Implementing the search by keywords is straightforward, first the words are separated and the database is queried for their ids. If at least one of the words is not in the database, the result is empty, otherwise a query for all images associated with these words do the work.

In the search by sketch, first the Average Colors feature vector is extracted from the sketch, then its BMU is found in the corresponding map. The resulting pictures are taken from the BMU and its neighbors. The number of returned images is fixed, so the search proceeds spirally until the desired amount is reached, and possibly passed, as all images of each visited node are taken. These images are then sorted by Euclidean distance to the vector extracted from the sketch and the first ones are presented to the user.

The search by similarity is the core of LACAIO. It is the only query method that uses all feature vectors and the only that uses relevance feedback. The process starts by finding the nodes associated to each relevant or irrelevant image in each map. Then, a score is computed for each node of the bottommost layer of each TS-SOM. The scores are calculated by adding one point to each BMU of a relevant image and subtracting one point from each BMU of an irrelevant image. After computing the scores, a Gaussian filter is used to spread these values over the neighboring nodes. Figure 4.3 represents the bottommost layer of a TS-SOM after computing the scores of each node (left) and after applying the Gaussian filter (right). The darkest the node, the higher the score. Finally, the scores of the images are calculated by summing the scores of the nodes associated to them in each map and the images with the highest scores are presented to the user.

Calculating the score of each image in a huge database would be too costly, computationally. To avoid all this processing, the nodes are first sorted by score and only the top ranked need to be visited. For more details on this, see Listing 4.1.

Using this algorithm eliminates the need to explicitly defining weights to feature vectors. Maps where the relevant images are closer, will have nodes with higher scores than maps where relevant images are spread and mixed with irrelevant ones. The relevance feedback is made by defining images from the result of one query as relevant or irrelevant

Listing 4.1: Pseudocode for search by similarity

```
1   integer scoreFromLast = 0
2   integer imagesToRetrieve // number of search results
3   sorted list images:= {}
4   list visitedImages:= {}
5   list nodeList:= {n | n is a node of the bottommost
6                   layer of one of the TS-SOMs}
7
8   sort(nodeList) // in decreasing order of score
9   foreach (node in nodeList) {
10    foreach (image associated to node) {
11      if (image is not in visitedImages) {
12        visitedImages.add(image)
13        if (images.length() < imagesToRetrieve
14            or image.getScore() > scoreFromLast){
15          images.add(image)
16          if (images.length() > imagesToRetrieve) {
17            images.removeLast()
18          }
19          scoreFromLast:= images.getLast().getScore()
20        }
21      }
22    }
23    if (getBestPossibleScore() < scoreFromLast) break
24  }
25  return images
```

and performing a new query.

5 FEATURE VECTORS

Four feature vectors were used, two for color (one global and one local), one for texture and one that can be classified as a texture feature or as shape feature. Scalable Color and Edge Histogram feature vectors are based on MPEG-7 standard descriptors.

5.1 Average Colors

Average Colors is a local color feature. The image is divided in nine non-overlapping sub-images and the average color for each sub-image is calculated. The HSV color space was chosen because it is more close to the way human perceive colors. The vector length is 27 (9 sub-images x 3 color channels).

5.2 Scalable Color

This feature vector is a global color feature and is based on a MPEG-7 standard descriptor with the same name. It consists in a 256-bin histogram in HSV color space. The colors are quantized to 16 values for Hue, 4 for Saturation and 4 to Value. The values are normalized so this feature is scale invariant and, as a histogram, it is also rotation invariant.

5.3 Texture

This is a simple local texture feature and is not based on any standard. The image is divided in 9 sub-images and for each sub-image, 4 values are calculated. These values are the average distances between the color of corresponding pixels (i.e., pixels labeled with the same number in Figure 5.1). For each pixel *P*, these distances are computed and the normalized averages of these values for each sub-image form this 36-dimension feature vector.

1	2	3
4	P	4
3	2	1

Figure 5.1: Matrix used in Texture feature extraction.

5.4 Edge Histogram

The Edge Histogram feature vector is based on a MPEG-7 standard texture descriptor, but can also be seen as a shape feature. It is also a local feature, since the image is divided in sub-images. The standard descriptor divides the image in 16 sub-images, but in this implementation the same 9 sub-images were used for every local feature. As the name suggests, this feature consists in a histogram of edge types, actually one for each sub-image. As illustrated in Figure 5.2, five edge types are considered (horizontal, vertical, 45°, 135° and non-directional).

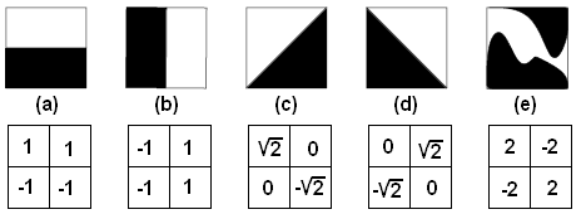

Figure 5.2: The five types of edges (horizontal (a), vertical (b), 45° (c), 135° (d) and non-directional (e)) and their corresponding edge detection filters.

The feature extraction is made by dividing each sub-image in blocks, each block is classified as being of one of the five edge types and then a normalized histogram is computed for each sub-image. To classify a block, it is divided in four sub-blocks and the average luminance of each of them is calculated. The luminance values of the four sub-blocks are convolved with filter coefficients in lower row of Figure 5.2 to obtain edge strengths. Among the five strengths, the maximum value determines the edge type. All image subdivisions can be seen in Figure 5.3.

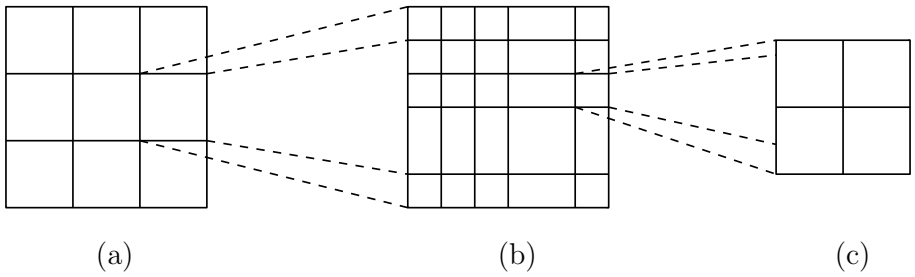

Figure 5.3: The subdivisions of an image - An image divided in 9 sub-images (a), a sub-image divided in blocks (b) and a block divided in 4 sub-blocks (c).

6 EVALUATION

Evaluating a CBIR system is difficult due to its subjectiveness, but CBIR is in essence an information retrieval problem. Therefore, some evaluation metrics were borrowed from information retrieval. According to (DATTA et al., 2008), two of the most popular evaluation measures for CBIR are *precision* and *recall*.

Given a query, a set of images relevant to this specific query and the set of retrieved images as the answer to the query, we have the following definitions:

Precision is the percentage of retrieved images that belongs to the set of relevant images.

Recall is the percentage of relevant images covered by the retrieved images.

Precision and recall are inversely proportional. As the number of retrieved images increases, recall also increases, but precision decreases. Traditionally, these measures are presented as *precision-recall* curves.

In order to make the measurements, one or more classes of semantically related images must be manually defined. Having these classes defined, the precision-recall pairs can be achieved by making sequences of queries, trying to obtain all images of a given class.

The tests were made with 350 images of 7 non-overlapping classes with 50 images in each class. All images used were taken from Wikimedia Commons (WIKIMEDIA, 2004) using Mayflower (MAYFLOWER, 2006) search tool to search by the following words: building, car, drawing, flower, marine life, plane and sunset.

First, one image is selected and defined as relevant to make the first query and get the first set of images. Each image retrieved in this way is classified as relevant if it belongs to the same class or irrelevant otherwise and a new query is made. These steps are repeated until all images of the given class are retrieved. All 350 images were used as the first image one time and the average precision and recall were computed for each class.

According to (JORMA LAAKSONEN MARKUS KOSKELA, 2002), the FVs can be tested individually, but this kind of test have severe limitations in particular because they do not take any relevance feedback into account. To overcome this problem, the tests described above were repeated five times, one using all four feature vectors and one for each combination of three of them. Comparing the results of the tests using four vectors and the tests using only three, one can tell how the missing vector affects the results.

Figure 6.1 shows the five *precision-recall* curves for each of the seven classes. Note that, as recall approaches 100%, precision falls approaching the precision of a random search ($\approx$ 14.3%). Figure 6.2 shows the distributions of the images of each class over the maps. Darker areas represent regions to which more images were mapped. A map in which the images of a given class are highly concentrated in a few regions indicates that

the corresponding feature is effective to cluster images of that class, but not necessarily to distinguish them from images from the other classes. The ideal situation to identify images of a class is when its images are clustered in one map and isolated from other images in another, or in the same map.

Clearly, drawings and sunsets are the easiest classes and buildings and planes the hardest. Drawings can easily be differentiated because they have a white background, while sunsets have smooth gradients of yellow, orange and red. Note in Figure 6.2 that images of buildings are far more concentrated than images of flowers in all maps. Nevertheless, flowers class reaches a higher precision. The initial precision for both classes is roughly the same, but the relevance feedback highly improves precision for flowers while doesn't seem to make any good to buildings class. This happens because although buildings images are more clustered, flowers images are more isolated from the others.

Benchmarking visual information retrieval solutions is an open problem. Comparing CBIR systems using different image collections with different sizes and different classes is inconclusive. In (HUIJSMANS; SEBE, 2005) a normalization is suggested to enable comparisons between precision-recall results independently of database or classes sizes. This is made by using the *a priori* probability of getting the relevant images to normalize the precision. However, no considerations are made about the content of irrelevant images. Adding images easily distinguishable from all classes to the collection would lower the *a priori* probability without significantly affecting the precision-recall curves, which would end in higher normalized results. Therefore, only using the exact same data to test different systems, conclusive results can be achieved.

All tests were made in a AMD Turion 64 X2 Mobile 1.60 GHz processor and each query takes about 500 milliseconds. More than 90% of this time is spent on database queries.

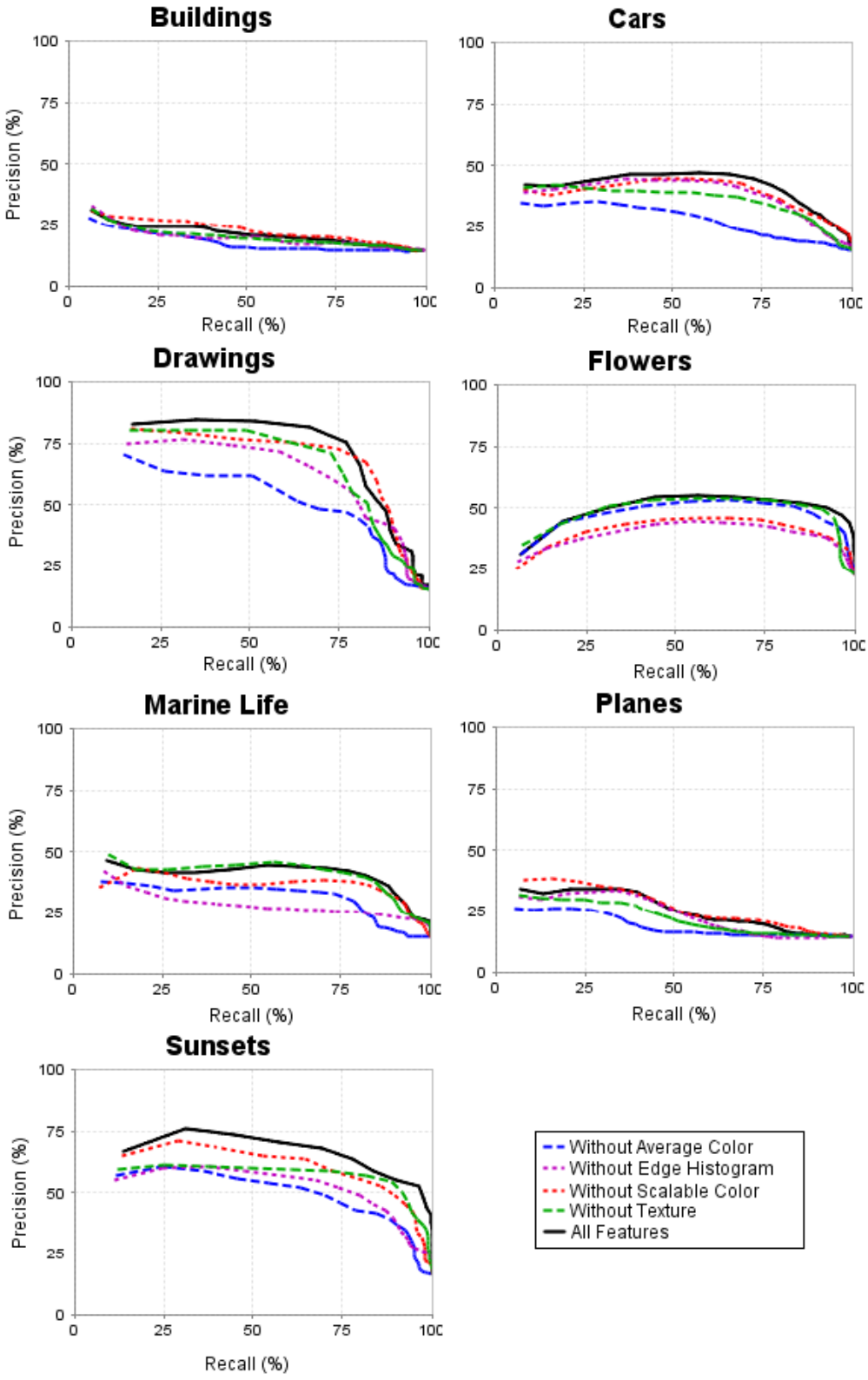

Figure 6.1: Precision-recall curves for each of the seven classes.

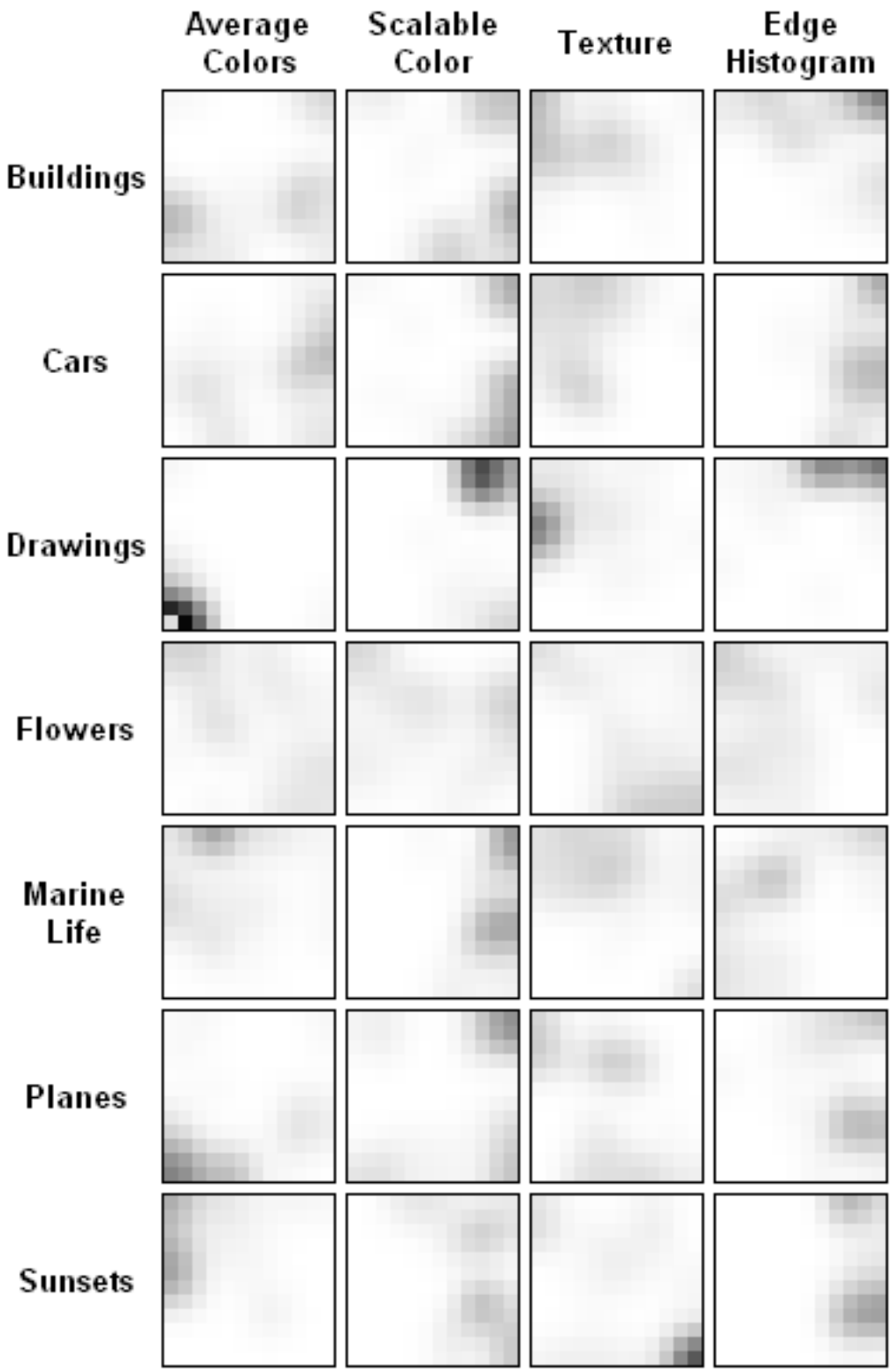

Figure 6.2: Distributions of the image classes over the bottommost layer of the maps.

7 CONCLUSIONS

Content-Based Image Retrieval is relatively new as a research field and thus has many open problems yet. Most image organizers available today make no use of CBIR techniques. The few image organizers who use CBIR are still very immature and are not ready for use by the general public.

LACAIO combines CBIR and RF techniques to ease image retrieval in a partially annotated image collection. LACAIO's GUI is very simple, but it enables search by keywords, search by sketch and search by similarity with relevance feedback.

Testing and comparing CBIR systems is difficult and subjective, but the tests suggest that the implemented system successfully achieved its objective. For all tested image classes the results were better than random results and the two better results passed 75% of precision at some point.

7.1 Discussion and Further Work

CBIR systems like LACAIO are composed of parts, to a certain level, independent from each other. Some of these parts have a wide range of possible alternative implementations. Some of these implementations have a large amount of parameters that can be tuned to improve the quality of the results. All these possibilities make necessary further investigation and comprehensive tests to determine the best alternatives. From the design issues discussed in Section 2.1, two deserve special attention: the features to use and the structure to index the features.

As can be seen in Figure 6.1, the addition of a new feature vector doesn't seem to negatively affect the results. Thus implementing other feature vectors would probably improve the results. However, redundant features are not likely to be useful and would decrease performance and increase the size of the database. Therefore, it is important to determine an ideal set of features such that the compromise between precision and performance be the best possible.

Tree-Structured Self-Organizing maps have many parameters that need further tuning to achieve its best results. However, other alternative models of Self-Organizing Maps could be used instead of TS-SOM. Some of them, like Growing Hierarchical Self-Organizing Map (DITTENBACH; MERKL; RAUBER, 2000) or Adaptive Hierarchical Incremental Grid Growing (MERKL et al., 2003) are more adaptable to changes. This adaptability can be useful to handle the addition of new images to already trained maps.

Another point that needs further work is the evaluation. More extensive and comprehensive tests should be made with a large set of images. With a few thousand images, more conclusive results about precision and recall could be achieved. Performance tests should also be conducted to see how the system handles queries in large databases.

REFERENCES

CHOPPIN, A. **Unsupervised Classification of High Dimensional Data by Means of Self-Organizing Neural Networks**. 1998.

DATTA, R.; DHIRAJ, J.; JIA, L.; WANG, J. Z. Image Retrieval: ideas, influences, and trends of the new age. **ACM Computing Surveys**, [S.l.], v.40, n.2, 2008.

DITTENBACH, M.; MERKL, D.; RAUBER, A. The Growing Hierarchical Self-Organizing Map. In: NEURAL NETWORKS, 2000. IJCNN 2000, PROCEEDINGS OF THE IEEE-INNS-ENNS INTERNATIONAL JOINT CONFERENCE ON, 2000. **Anais...** [S.l.: s.n.], 2000. v.6, p.15–19 vol.6.

GIFT. GIFT - The GNU Image-Finding Tool. Available at: <http://www.gnu.org/software/gift>.

HUIJSMANS, D.; SEBE, N. How to complete performance graphs in content-based image retrieval: add generality and normalize scope. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, [S.l.], v.27, n.2, p.245–251, 2005.

IMGSEEK. imgSeek - photo collection manager and viewer with content-based search. Available at: <http://www.imgseek.net>.

JORMA LAAKSONEN MARKUS KOSKELA, E. O. PicSOM - A Framework for Content-Based Image Database Retrieval using Self-Organizing Maps. In: SCANDINAVIAN CONFERENCE ON IMAGE ANALYSIS (SCIA'99), KANGERLUSSUAQ, GREENLAND, JUNE 7–11, 11., 1999. **Proceedings...** [S.l.: s.n.], 1999. p.151–156.

JORMA LAAKSONEN MARKUS KOSKELA, E. O. PicSOM: self-organizing maps for content-based image retrieval. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN'99), WASHINGTON, D.C., USA, JULY 10–16, 1999. **Proceedings...** [S.l.: s.n.], 1999.

JORMA LAAKSONEN MARKUS KOSKELA, E. O. PicSOM - Self-Organizing Image Retrieval with MPEG-7 Content Descriptors. **Neural Networks, IEEE Transactions on**, [S.l.], v.13, n.4, p.841–853, 2002.

MARKUS KOSKELA JORMA LAAKSONEN, E. O. Inter-Query Relevance Learning in PicSOM for Content-Based Image Retrieval. In: SUPPLEMENTARY PROC. OF INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETWORKS (ICANN'03), 2003, Istanbul, Turkey. **Anais...** [S.l.: s.n.], 2003. p.520–523.

MAYFLOWER. Mayflower search engine. Available at: <http://toolserver.org/ tangotango/mayflower>.

MERKL, D.; HE, S. H.; DITTENBACH, M.; RAUBER, A. Adaptive Hierarchical Incremental Grid Growing: an architecture for high-dimensional data visualization. In: WORKSHOP ON SELF-ORGANIZING MAPS (WSOM 2003), 4., 2003, Hibikino, Kitakyushu, Japan. **Proceedings...** [S.l.: s.n.], 2003.

MPEG-7. MPEG-7 Overview (version 10). Available at: <http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm>.

OCTAGON. Octagon - content based image retrieval software. Available at: <http://octagon.viitala.eu>.

PAKKANEN, J.; IIVARINEN, J. Evaluating SOM as an Index in Content-Based Image Retrieval. In: FINNISH ARTIFICIAL INTELLIGENCE CONFERENCE, 10., 2002, Oulu, Finland. **Proceedings...** [S.l.: s.n.], 2002. p.182–188.

SAHA, S. K.; DAS, A. K.; CHANDA, B. Image retrieval based on indexing and relevance feedback. **Pattern Recognition Letters**, [S.l.], v.28, n.3, p.357–366, February 2007.

SMEULDERS, A.; WORRING, M.; SANTINI, S.; GUPTA, A.; JAIN, R. Content-Based Image Retrieval at the End of the Early Years. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, [S.l.], v.22, n.12, p.1349–1380, 2000.

VESANTO, J.; HIMBERG, J.; ALHONIEMI, E.; PARHANKANGAS, J. Self-organizing map in matlab: the SOM toolbox. In: MATLAB DSP CONFERENCE, 1999, Espoo, Finland. **Proceedings...** [S.l.: s.n.], 1999. p.35–40.

VISUALPHOTOCOMPARE. Available at: <http://home.planet.nl/ edejong/visualphotocompare>.

WENYIN, L.; DUMAIS, S.; SUN, Y.; ZHANG, H.; CZERWINSKI, M.; FIELD, B. **Semi-automatic image annotation**. 2001.

WIKIMEDIA. Wikimedia Commons. Available at: <http://commons.wikimedia.org/wiki>.