46)

Image Database Retrieval With Multiple-Instance Learning Techniques

by

Cheng Yang

B.S. Computer Science
Yale University (1997)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements
for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1998

Author ....
Department of Electrical Engineering and Computer Science
August 27, 1998

Certified by ....
Tomás Lozano-Pérez
Cecil H. Green Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Image Database Retrieval With Multiple-Instance Learning Techniques

by

Cheng Yang

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 1998, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

In this thesis, we develop and test an approach to retrieving images from an image database based on content similarity. First, each picture is divided into many overlapping regions. For each region, the sub-picture is filtered and converted into a feature vector. In this way, each picture is represented by a number of different feature vectors. The user selects positive and negative image examples to train the system. During the training, a multiple-instance learning method known as the Diverse Density algorithm is employed to determine which feature vector in each image best represents the user's concept, and which dimensions of the feature vectors are important. The system tries to retrieve images with similar feature vectors from the remainder of the database. A variation of the weighted correlation statistic is used to determine image similarity.

The approach is tested on a large database of natural scenes as well as single- and multiple-object images. Comparisons are made against a previous approach, and the effects of tuning various training parameters, as well as that of adjusting algorithmic details, are also studied.

Thesis Supervisor: Tomás Lozano-Pérez
Title: Cecil H. Green Professor of Computer Science and Engineering

to my parents
Zhuping Wu and Pengxin Yang

Acknowledgments

I would like to express my deepest gratitude to my thesis supervisor, Professor Tomás Lozano-Pérez, for giving me the opportunity and support to work on this thesis, for providing me with crucial guidance and encouragement throughout the year, and for serving as a model of an enthusiastic teacher and a dedicated scientist. This work would not have been possible without his help and advice.

I am grateful to Professor Eric Grimson and Dr. Tom Knight for offering me the opportunity to study and work at MIT AI lab and for providing me with Research Assistantships. I also thank Professor James Duncan at Yale University for introducing me to the fields of image processing and computer vision.

Thanks go to Oded for teaching me the Diverse Density concept and implementation details, to Aparna, Kinh, Jeremy and Tianyun for the technical discussions and inspirations, to Jacqui for the administrative help, and to all members of the AI lab for making a fun and enjoyable environment to work in. I would also like to thank Bikui, Gary, Haicheng, Huijing, Xiaojun, Yibing and all my friends at MIT, at Yale and elsewhere for the great moments we have shared in the past years.

I am forever indebted to my parents, for their unconditional love, constant support and confidence in me, and for making me the way I am. No words are adequate to express my gratitude to them.

Finally, I would like to thank my dear friend Jie, for her love, encouragement and commitment to our relationship which conquered the 3,000-mile distance between us.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview and Previous Work

While searching for textual data on the World Wide Web and in other databases has become common practice, search engines for pictorial data are still rare. This comes as no surprise, since it is a much more difficult task to index, categorize and analyze images automatically, compared with similar operations on text.

An easy way to make a searchable image database is to label each image with a text description, and to perform the actual search on those text labels. However, a huge amount of work is required in manually labelling every picture, and the system would not be able to deal with any new pictures not labelled before. Furthermore, it is difficult to give complete descriptions for most pictures. Consider the picture in Figure 1-1. One might be tempted to describe it as "river, trees and stones", but it would not be able to respond to user queries for "water", "waterfall", "clouds" or "white blobs in background". To make a real content-based image retrieval system, we need some mechanism to search on the images directly.

Early approaches to the content-based image retrieval problem include the IBM QBIC (Query-By-Image-Content) System [3], where users can query an image database by average color, histogram, texture, shape, sketch, etc. The image database is preprocessed with some human assistance to facilitate the search. However, image queries along these lines are not powerful enough, and more complex queries (such as "all

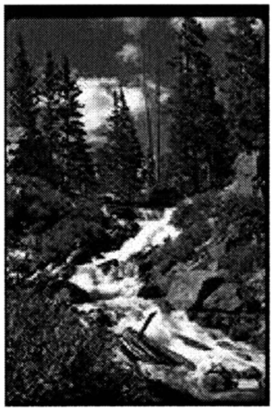


Figure 1-1: A Sample Picture

pictures that contain waterfalls") are hard to formulate. Lipson *et al.* [8] used handcrafted templates to classify natural scene images. While it has been successful in this domain, the process is difficult to automate. Recent research has paid more attention to query by examples [1, 11, 15]. In these systems, user queries are given in terms of positive and negative examples, and sometimes salient regions are also manually indicated. The system then proceeds to retrieve images "similar" to the positive examples and "dissimilar" to the negative ones.

For images, however, "similarity" is not well-defined. Many algorithms have been proposed to compute image similarities. They typically do so by converting images into feature vectors and using feature vector distances as a similarity measure. Grosky and Mehrotra [4] experimented with a representation using object boundaries' local structural features, and they used string edit-distance as a distance measure. Mehrotra and Gary [12] used relative positions of "interest points" along object boundaries to represent shape, and used Euclidean distance as a distance measure. These methods are based on object recognition techniques. However, they are quite sensitive to noise in the images, and cannot handle images where there are no distinct objects, as in natural scenes. De Bonet and Viola [1] proposed an algorithm where images are passed through a tree of nonlinear filters to obtain feature vectors that represent "texture-of-texture" of the original images. It works well with natural scenes and single-object test sets. Maron and Lakshmi Ratan [11] used simple features like a row's mean color, color differences and color distributions among neighbors, etc, and it works well for color images of natural scenes.

More detailed reviews of previous literature in image classification and retrieval can be found in [7, 11].

1.2 The Multiple-Instance Learning Approach

Since the picture in Figure 1-1 can be viewed differently as “river”, “waterfall”, “trees”, “clouds”, etc, and multiple-object images are more common than single-object images, it is natural to have one image correspond to more than one feature vector, each one describing one particular view (or object). In this way, each positive or negative example translates into multiple feature vectors. After Maron [9], we call each of these feature vectors an *instance*, and we call the collection of instances for the same example image a *bag*.

For a positive example, at least one of the instances in the bag is a close match to the concept the user had in mind when he or she chose the examples, but we do not know which one. The rest of the instances in the bag are irrelevant and should be regarded as noise. For a negative example, we know for sure that none of the instances in the bag corresponds to the user's concept. Given the large collection of instances from positive and negative examples, our task is to find the “ideal” feature vector that accounts for the user's concept.

This kind of problem is known as a *Multiple-Instance Learning* problem [2, 9, 10]. One way to solve this type of problem is to examine the distribution of these instance vectors, and to look for a feature vector that is close to a lot of instances from different positive bags and far from all the instances from negative bags. Such a vector is likely to represent the concept we are trying to learn. This is the basic idea behind the *Diverse Density* algorithm, proposed by Maron and Lozano-Pérez [9, 10, 11], and is illustrated in Figure 1-2. In Figure 1-2, there are five positive examples (bags) labelled 1 to 5 and three negative examples (bags) labelled 6 to 8. Each bag has several instances. The feature vector space is 2-dimensional. The “ideal” feature vector is where there is a high concentration of positive instances from different bags.

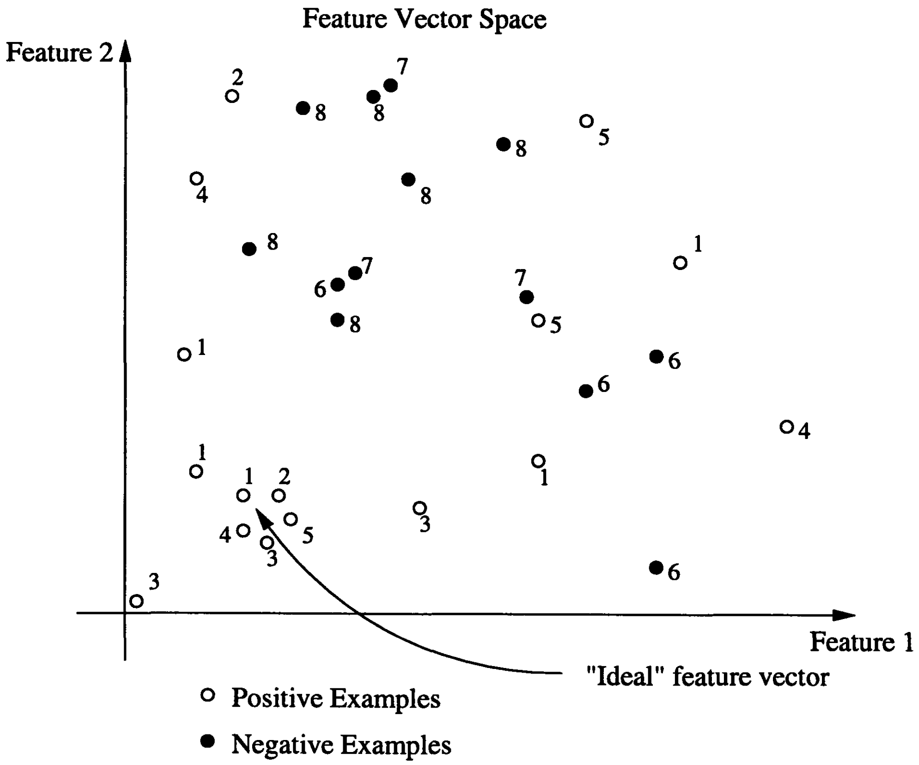


Figure 1-2: A Multiple-Instance Learning Algorithm: Diverse Density

Maron and Lakshmi Ratan [11] have applied the Diverse Density technique to image retrieval problems by using image features such as color statistics and color distribution patterns. In this thesis, we develop a new approach to solving the problem of content-based image retrieval by using Diverse Density learning techniques. We define an image similarity measure as the correlation coefficient of corresponding regions after smoothing and sampling, and further refine it by allowing different weight factors for different locations when comparing for similarity. Based on this, we develop a feature vector representation for images where we can use weighted Euclidean distance to reflect the distance defined by our weighted similarity measure. Multiple instances for each example image are obtained by choosing different subregions of the image and generating a feature vector for each region. We show that this approach is effective in content-based retrieval of images.

Specifically, each image in our database is preprocessed as follows:

1. If it is a color image, convert it into a gray-scale image.

2. Select some regions from the image. This will be discussed in Section 3.2.

3. Extract two sub-pictures from each region: one as the image itself in the region, and the other as its left-right mirror image. For each sub-picture, perform smoothing and sampling to get an $h \times h$ matrix, and treat this as an h^2-dimensional feature vector. This will be discussed in Section 3.1.2.

4. Transform each feature vector into a new one by subtracting its mean from it and dividing it by its standard deviation. This transformation enables us to use weighted Euclidean distance rather than weighted correlation coefficient to determine image similarity. This will be discussed in Section 3.4.

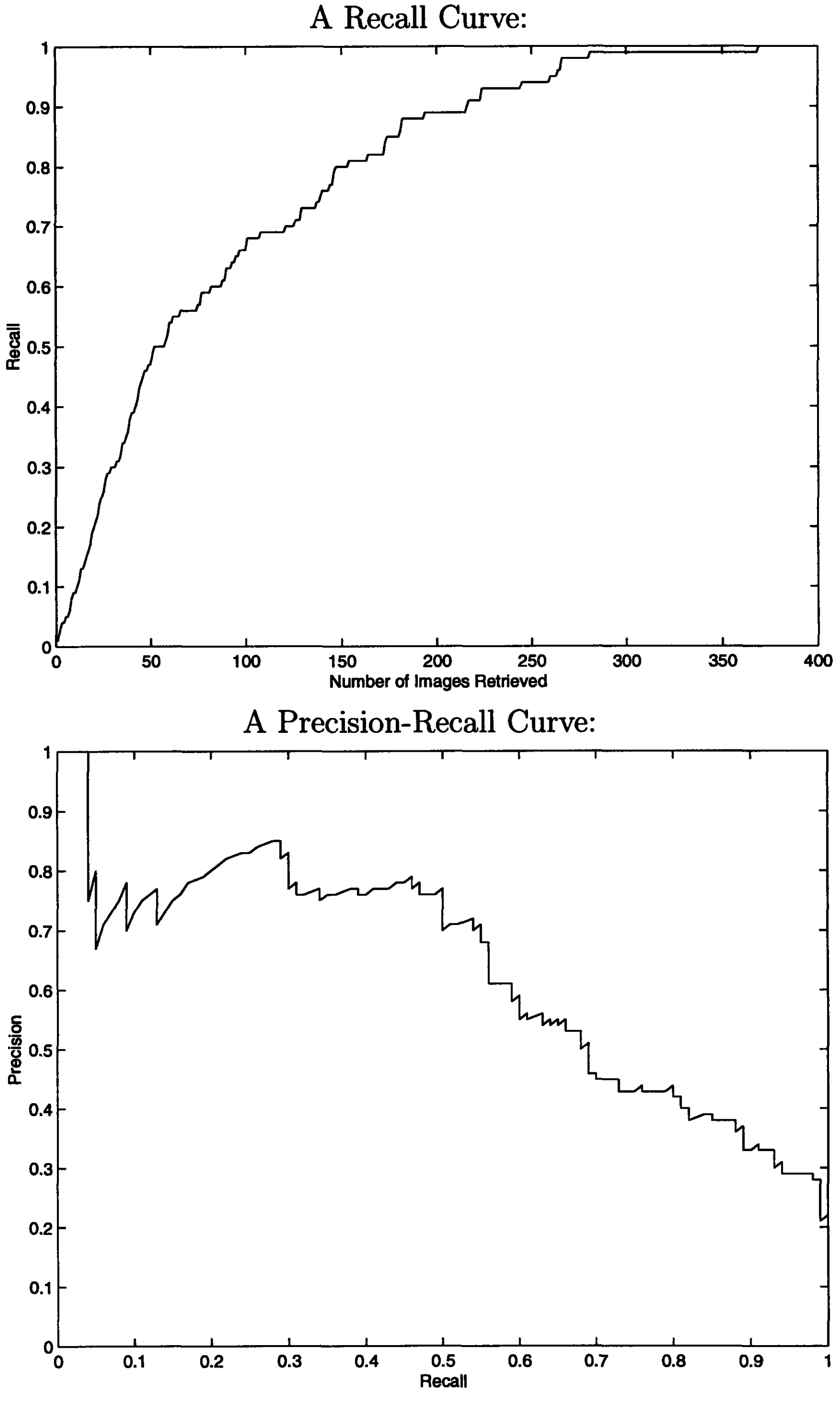
5. For each image in our database, we have obtained a number of feature vectors (after the transformation). Treat each one as an instance and put them together to form a bag for the image.

In response to user selections of positive and negative example images, our system puts together the corresponding image bags and feeds them into the Diverse Density (DD) algorithm. The DD algorithm returns an "ideal" point in the feature space as well as a corresponding set of feature weight values. Then the system goes to the image database and ranks all images based on their weighted Euclidean distances to the ideal point. (To find the distance from an image to the ideal point, it computes the distances of all of its instances to the point, and then picks the smallest one.) It then retrieves images in the ranked order. If the retrieval results are not satisfactory, the user may obtain better performance by picking out false positives and/or false negatives, adding them to the examples and training the system again. During evaluation, this part of user feedback can also be simulated by the computer automatically.

The treatment of feature weight values in the DD algorithm affects the system performance significantly. The original DD algorithm described in [9] tends to push most of weight values towards zero, leaving only a few large values. This is not ideal for the image retrieval tasks. In Section 3.6, we will discuss some modifications of the DD algorithm aimed at controlling feature weight values.

The algorithm is tested on a large image database, and performance is evaluated using precision-recall curves and recall curves such as the ones shown in Figure 1-3. Precision is the ratio of the number of correctly retrieved images to the number of all images retrieved so far. Recall is the number of correctly retrieved images to the total number of correct images in the test database. In Section 4.2, we will make comparisons among different schemes of controlling feature weight factors, and will study the effects of choosing different number of instances per bag, as well as changing resolutions for smoothing and sampling. Comparisons will also be made with a previous approach to using DD in example-based image retrieval by Maron and Lakshmi Ratan [11].

Chapter 2 of this thesis introduces the Diverse Density algorithm in greater detail. Chapter 3 discusses our correlation similarity measure, its corresponding feature representation and weight factor controlling methods. Chapter 4 gives experimental details, shows the effects of modifying various parts of the algorithm, and discusses a way to reduce training time to speed up the system.



A Recall Curve:

A Precision-Recall Curve:

Figure 1-3: Sample Precision-Recall Curve and Recall Curve

Chapter 2

The Diverse Density Algorithm

2.1 Background

2.1.1 Machine Learning

Machine learning algorithms provide ways for computer programs to improve automatically with experience [13]. This is a step further from traditional programming approaches where the programmer has to give step-by-step instructions to carry out computation. Sometimes the instructions for a computation can be very complex and not easily defined. For example, what kind of credit card transactions are likely to be fraudulent? It is hard to give a formula to determine "good" versus "bad" transaction patterns; even if we found one, it might change over time. However, we do have a large database of historical data of good and bad transaction records, from which the computers can learn. Sometimes we just do not know how to give explicit instructions to computers. For example, how do we recognize human speech and handwriting? There must exist functions which map sound signals or handwriting image pixels to words or letters, but we are not yet able to write them down. However, as humans, we know how to do these things without being aware of the exact rules. And children learn them easily not from rules, but from examples. So why not let computers learn from us, also from examples?

In a typical machine learning problem, the task is to learn a function

$$y = f(x_1, x_2, ..., x_n)$$

In our previous examples, input values $x_1, x_2, ..., x_n$ can be the fields in credit card transaction records, digitized sound signals or handwriting image pixel values, while the output y can be a boolean value (indicating "good" or "bad"), a word or a letter. f is the function which we would like the computer to learn, so that it can be applied to new input values. A lot of examples need to be provided to "train" the machine learning algorithm.

In *Supervised Learning*, the examples are given in terms of $(y_i, x_{i1}, x_{i2}, ..., x_{in})$ tuples, where i is the index of examples: $i = 1, 2, 3, ...$ That is, each set of input values $(x_{i1}, x_{i2}, ..., x_{in})$ is tagged with the correct label y_i. In *Unsupervised Learning*, however, only the tuples $(x_{i1}, x_{i2}, ..., x_{in})$ are given in the training examples, without any information on y_i. Between these two extremes, there are many other possible scenarios. For example, only part of the inputs are labelled with y_i values; the labels y_i may be noisy, unreliable or not immediately available, etc. We are going to focus on one of these scenarios: *Multiple-Instance Learning* [2, 9, 10, 11].

2.1.2 The Multiple-Instance Learning Problem

In the Multiple-Instance Learning framework that we are going to consider, all input values are n-dimensional vectors in the form of $(x_{i1}, x_{i2}, ..., x_{in})$ and the output value y_i is a real number between 0 and 1 to indicate possibility (0 for FALSE, 1 for TRUE). In the training examples, input vectors (called *instances*) are not individually labelled with its corresponding y_i value; rather, one or more instances are grouped together to form a *bag*, and they are collectively labelled with a y value TRUE or FALSE. If the label is TRUE, it means that at least one of the instances in the bag must correspond to y_i=TRUE, while others may correspond to either TRUE or FALSE. If the label is FALSE, it means that all of the instances in the bag must correspond to FALSE.

In terms of the image retrieval problem, each positive example selected by the user corresponds to a bag labelled TRUE, and each negative example selected by the user corresponds to a bag labelled FALSE. A feature vector consists of n numbers (features), each of which partially describes the image in some way, for example, pixel values, color statistics, edge locations, etc. Redundant or irrelevant features are allowed. Since the pictures are inherently ambiguous, we generate more than one feature vector (instance) to describe each picture. We expect that one of these feature vectors for each positive example would account for the concept the user had in mind when picking the examples, and that none of them in the negative examples would coincide with the user's concept.

We would like to train the system so that it can make predictions for new examples: given a new example image (a bag of instance vectors), it should determine whether it correspond to TRUE or FALSE. To allow for uncertainty, the system may give a real value between 0 (FALSE) and 1 (TRUE).

We make a simplifying assumption that the user's concept can be represented by a single "ideal" point in the n-dimensional feature space. A bag is labelled TRUE if one of its instances is close to the ideal point. A bag is labelled FALSE if none of its instances is close to the ideal point. This is illustrated in Figure 2-1. In Figure 2-1, there are five positive examples (bags) labelled 1 to 5 and three negative examples (bags) labelled 6 to 8. Each bag has several instances. The feature vector space is 2-dimensional. The "ideal" feature vector is where there is a high concentration of positive instances from different bags. The probability of a bag being TRUE can be measured by the distance from the ideal point to the closest instance vector in the bag. [9, 10] developed an algorithm called *Diverse Density*, which is able to find such a point. Not all dimensions of feature vectors are equally important, so the distance here is not restricted to normal Euclidean distance, but may be defined as a weighted Euclidean distance where important dimensions have larger weights. The Diverse Density algorithm is capable of determining these weight factors as well.

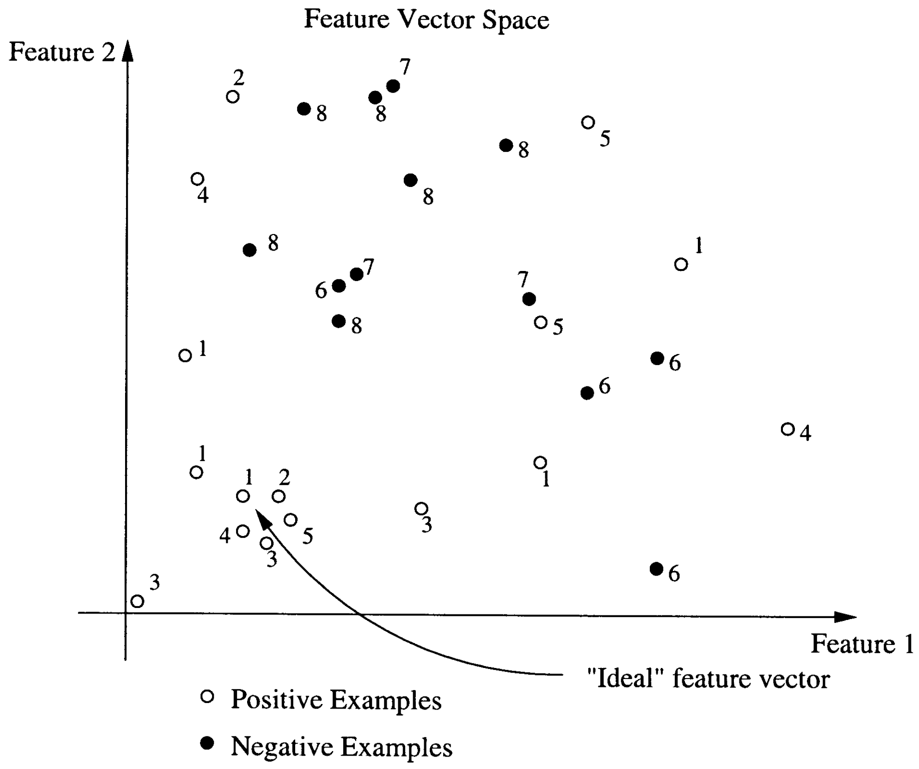


Figure 2-1: A 2-D Feature Vector Space in Multiple-Instance Learning

2.2 Diverse Density

2.2.1 Framework

In this section, we give a brief introduction to the theory behind the Diverse Density algorithm. A more elaborate treatment can be found in [9, 10, 11].

Following the same notations as in [9, 10, 11], we denote the positive bags as

$$B_1^+, B_2^+, ..., B_n^+$$

and the negative bags as

$$B_1^-, B_2^-, ..., B_m^-$$

The j^{th} instance of bag B_i^+ is written as B_{ij}^+, while the j^{th} instance of bag B_i^- is written as B_{ij}^-. Each bag may contain any number of instances, but every instance must be a k-dimensional vector where k is a constant.

Ideally, the point we are looking for should be the intersection of all positive bags minus the union of all negative bags. This is not true in reality, because of the presence of noise and inaccuracies. Therefore, our task is to look for a point in the k-dimensional space near which there is a high concentration of positive instances from different bags. It is important that they are from *different* bags, since a high concentration of instances from the same bag is effectively the same as one instance at that point. In other words, we are looking for a point where there is a high *Diverse Density* of positive instances.

For any point t in the feature space, the probability of it being our target point, given all the positive and negative bags, is: $\Pr(t|B_1^+, ..., B_n^+, B_1^-, ..., B_m^-)$. So the point we are looking for is the one that maximizes this probability, that is

$$\arg\max_t \Pr(t|B_1^+, ..., B_n^+, B_1^-, ..., B_m^-)$$

According to Bayes' rule, this is equal to

$$\arg\max_t \frac{\Pr(B_1^+, ..., B_n^+, B_1^-, ..., B_m^-|t)\Pr(t)}{\Pr(B_1^+, ..., B_n^+, B_1^-, ..., B_m^-)}$$

Assuming a uniform prior over the concept location $\Pr(t)$, and since the denominator is a constant (probability of data), the above is equivalent to:

$$\arg\max_t \Pr(B_1^+, ..., B_n^+, B_1^-, ..., B_m^-|t)$$

Assuming conditional independence of the bags given the target concept t, this can be decomposed as

$$\arg\max_t \prod_i \Pr(B_i^+|t) \prod_i \Pr(B_i^-|t)$$

Again, with Bayes' rule and the assumption of a uniform prior over the concept location, and factoring constant data probability, this becomes

$$\arg\max_t \prod_i \Pr(t|B_i^+) \prod_i \Pr(t|B_i^-)$$

This is a formal definition of maximizing Diverse Density. We use the "noisy-or" assumption (see Maron [9] for motivation and discussions) that

$$\begin{aligned} \Pr(t|B_i^+) &= 1 - \prod_j (1 - \Pr(B_{ij}^+ = t)) \\ \Pr(t|B_i^-) &= \prod_j (1 - \Pr(B_{ij}^- = t)) \end{aligned}$$

and make the following assumption:

$$\Pr(B_{ij} = t) = \exp(-||B_{ij} - t||^2)$$

where $||B_{ij} - t||$ is the distance between the two vectors. This is a Gaussian bump centered on the feature vector. As we mentioned before, not all dimensions are equally important, so we define the distance to be a weighted Euclidean distance:

$$||B_{ij} - t||^2 = \sum_k w_k^2 (B_{ijk} - t_k)^2$$

where B_{ijk} is the k^{th} dimension in the vector B_{ij}. w_k^2 is a non-negative weight. (We use w_k^2 rather than w_k in order to force the weights to be non-negative.) Now we need to maximize Diverse Density over both t and w. By introducing weights, we have actually doubled the number of dimensions over which we are trying to maximize Diverse Density.

2.2.2 Finding the Maximum

The problem of finding the global maximum Diverse Density (DD) is difficult, especially when the number of dimensions is large. The original DD algorithm makes use of a gradient ascent method with multiple starting points. It starts from every instance from every positive bag and performs gradient ascent from each one to find the maximum. The idea is that, at least one of the positive instances is likely to be close to the maximum. So if we do hill-climbing from every positive instance, it is very likely that we will hit the maximum DD point.

We are trying to maximize over both t (feature values) and w (weight values). In the presence of few negative instances, the original DD algorithm tends to push most of the weights towards zero, leaving only a few large weight values. This is a form of overfitting, and is not desirable in our problem domain. In chapters 3 and 4 we will discuss alternate approaches, including adding constraints on the weight factors.

Chapter 3

Adapting Diverse Density

3.1 The Correlation Similarity Measure

3.1.1 Definitions

Given two series of sampled signals $f_1(t)$ and $f_2(t)$, $t = 1, 2, ..., n$, there is a standard way to find out how correlated they are with respect to each other: we can compute their *correlation coefficient* [16]. In its simplest form, the correlation coefficient r is defined by

$$r = \frac{\frac{1}{n}\sum_{t=1}^{n}(f_1(t) - \overline{f_1})(f_2(t) - \overline{f_2})}{\sigma_{f_1}\sigma_{f_2}}$$

where $\overline{f_1}$, $\overline{f_2}$ are the average values of $f_1(t)$ and $f_2(t)$, and σ_{f_1}, σ_{f_2} are the standard deviations of $f_1(t)$ and $f_2(t)$, respectively:

$$\begin{aligned}
\overline{f_1} &= \frac{\sum_{t=1}^{n} f_1(t)}{n} \\
\overline{f_2} &= \frac{\sum_{t=1}^{n} f_2(t)}{n} \\
\sigma_{f_1} &= \sqrt{\frac{1}{n}\sum_{t=1}^{n}(f_1(t) - \overline{f_1})^2} \\
\sigma_{f_2} &= \sqrt{\frac{1}{n}\sum_{t=1}^{n}(f_2(t) - \overline{f_2})^2}
\end{aligned}$$

Strictly speaking [14], in the definitions for σ_{f_1}, σ_{f_2} and r above, $\frac{1}{n}$ should be replaced by $\frac{1}{n-1}$. But it does not matter to us. Both definitions work the same way in the derivations of this chapter, and we choose to use $\frac{1}{n}$ which is more convenient.

When $r = 1$, the two signals are perfectly correlated (Figure 3-1(a)). When $r \approx 0$, there is little or no correlation between the two (Figure 3-1(b)). When $r = -1$, the two signals are perfectly inversely correlated (Figure 3-1(c)). If we only count positive correlations as "similar", then r can be used as a direct measurement of similarity: as r increases, similarity increases.

The same idea applies to two-dimensional signals, such as images, as well. The correlation coefficient r of $f_1(x,y)$ and $f_2(x,y)$ $(x = 1,2,...,n, y = 1,2,...,m)$ is defined by:

$$r = \frac{\frac{1}{nm}\sum_{x=1}^{n}\sum_{y=1}^{m}(f_1(x,y) - \overline{f_1})(f_2(x,y) - \overline{f_2})}{\sigma_{f_1}\sigma_{f_2}}$$

where

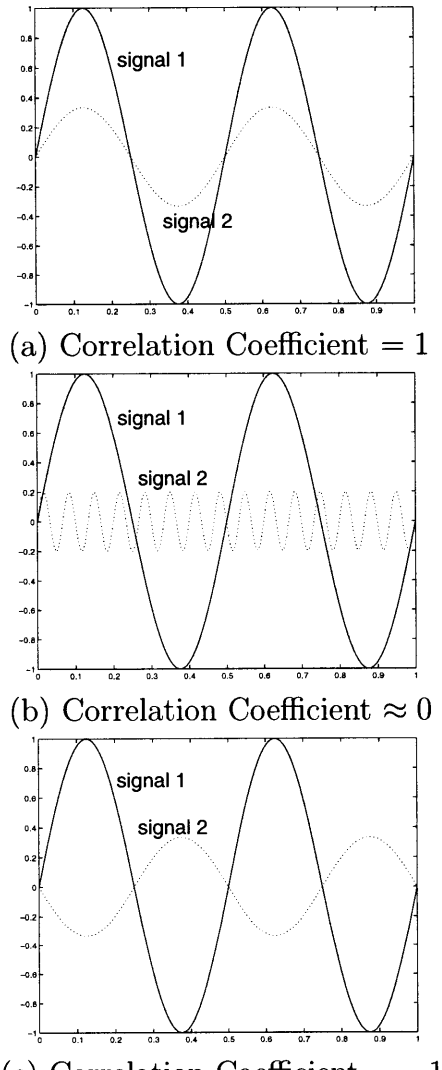
$$\begin{aligned}
\overline{f_1} &= \frac{\sum_{x=1}^{n}\sum_{y=1}^{m} f_1(x,y)}{nm} \\
\overline{f_2} &= \frac{\sum_{x=1}^{n}\sum_{y=1}^{m} f_2(x,y)}{nm} \\
\sigma_{f_1} &= \sqrt{\frac{1}{nm}\sum_{x=1}^{n}\sum_{y=1}^{m}(f_1(x,y) - \overline{f_1})^2} \\
\sigma_{f_2} &= \sqrt{\frac{1}{nm}\sum_{x=1}^{n}\sum_{y=1}^{m}(f_2(x,y) - \overline{f_2})^2}
\end{aligned}$$

Basically, we are just treating the $m \times n$ matrix as one big mn-dimensional vector. For images, this is often used to measure similarities between two regions [5].

3.1.2 Image Smoothing and Sampling

In this thesis, we deal with gray-scale information only. All color images are converted into gray-scale images first.

If we apply the correlation formula to the original images directly, on a pixel-by-pixel basis, a shift in the image by one pixel would cause a relatively big change in the correlation value, which is not desirable. To avoid this effect, we smooth and sample the $m \times n$ image down to a low-resolution $h \times h$ matrix. In most of the experiments



(a) Correlation Coefficient = 1

(b) Correlation Coefficient ≈ 0

(c) Correlation Coefficient $= -1$

Figure 3-1: Correlation Coefficient for 1-D signals

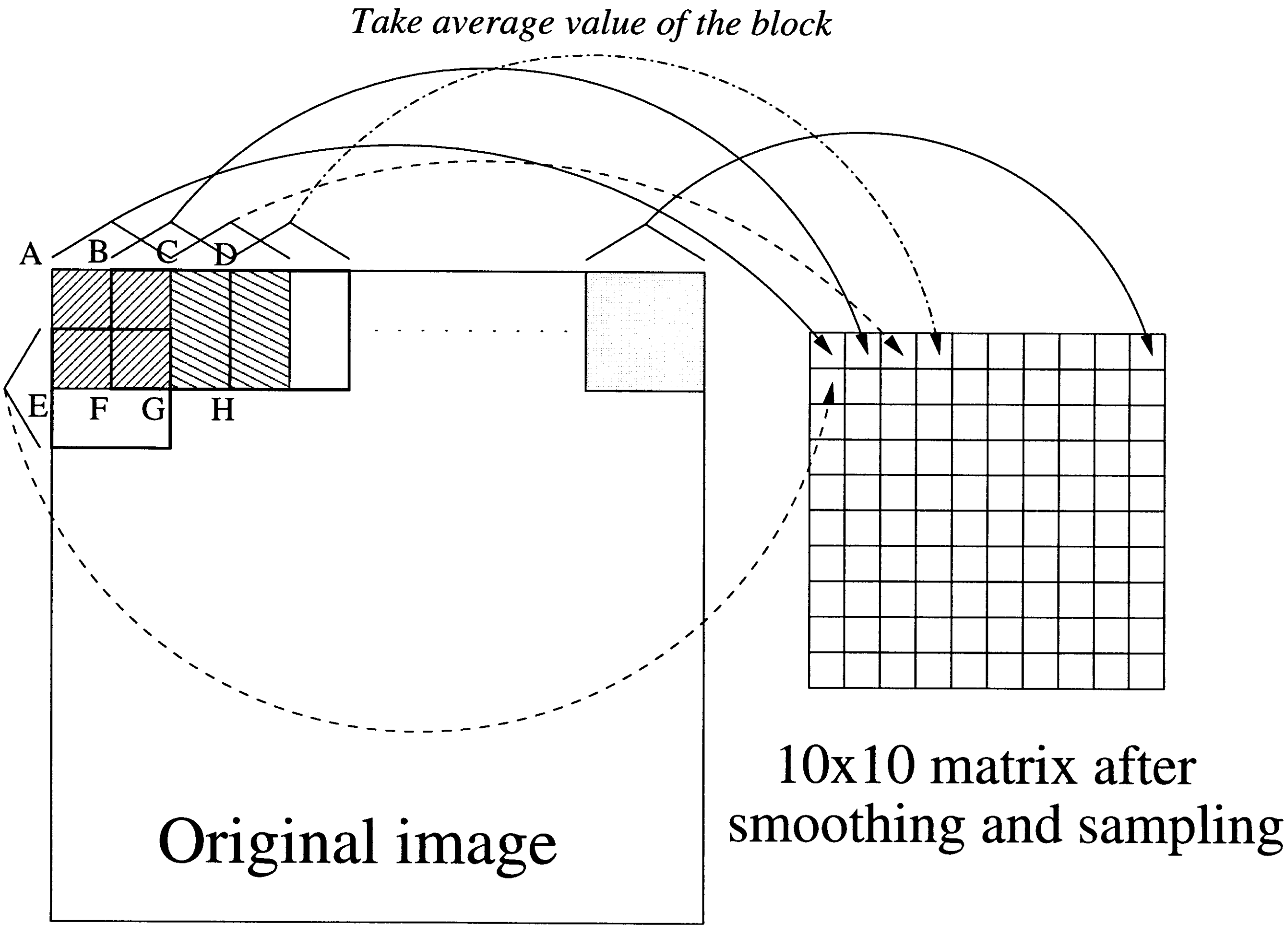

Take average value of the block
A
B
C
D
E
F
G
H
10x10 matrix after smoothing and sampling
Original image

Figure 3-2: Illustration of the Smoothing and Sampling Process

in this thesis, we choose $h = 10$. Specifically, we smooth the $m \times n$ image with a $\frac{2m}{h+1} \times \frac{2n}{h+1}$ averaging kernel and then sub-sample it to get an $h \times h$ matrix. In other words, each entry in the resulting $h \times h$ matrix is the average gray-scale value of a corresponding block region in the original image, as illustrated in Figure 3-2. In Figure 3-2, the average value of block $AEGC$ goes into the 1st entry of the 10×10 matrix, the average value of block $BFHD$ goes into the 2nd entry (1st row, 2nd column) of the matrix, and so on. Each block has a 50% overlap with any of its neighbors. The large overlap is intended to reduce sensitivity to the choice of block border locations.

With the above smoothing and sampling scheme and $h = 10$, Table 3.1 shows the correlation coefficients of some sample object images. It can be seen that, correlation coefficients are quite effective in measuring image similarities.

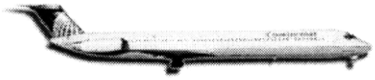
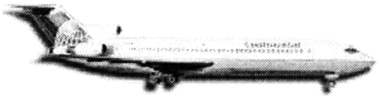
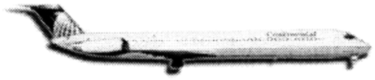
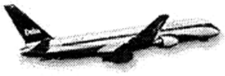
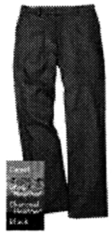
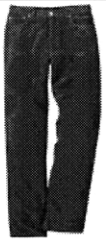
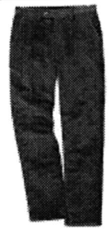
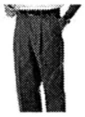
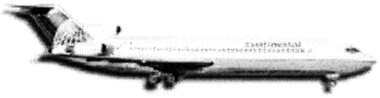
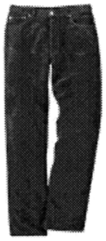
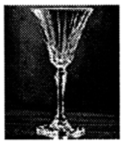
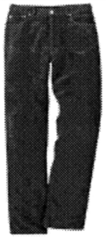
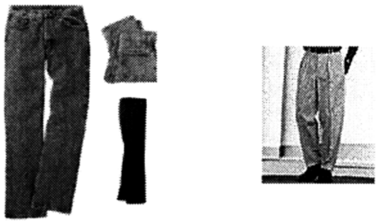
Picture 1	Picture 2	Correlation Coefficient
		0.838
		0.670
		0.783
		0.652
		0.110
		0.224

Table 3.1: Correlation Coefficients of Sample Image Pairs



Picture 1 Picture 2

The correlation coefficient of these two images is 0.118.

Figure 3-3: More Complex Images

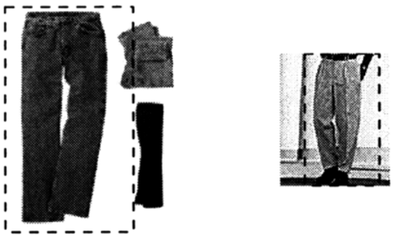
3.2 Region Selection

So far we have only been concerned with single-object images, where the correlation coefficient works well to determine similarity. This would not generalize to more complex cases such as multiple-object images, where the object (or feature) of interest may not be at the same position in all pictures.

In a more complex image, the object (or feature) of interest does not occupy the whole image, but only a sub-region of the image. We would not be able to get satisfactory results if we compared the two entire images in Figure 3-3 using the correlation similarity measure, but we may have better luck if we compare a region in one image against a region in the other. For example, the correlation coefficient of the two entire images in Figure 3-3 is 0.118, while the correlation coefficient of the two marked regions in Figure 3-4 is 0.674, indicating similarity.

Now the question is, how do we choose the regions? In fact, we do not know which regions we should pick, since the pictures are inherently ambiguous, and any region might become the region of interest, depending on the user's concept. This is exactly where multiple-instance learning can help us: we can simply pick all possible regions and let the learning algorithm take care of finding the "right" region for us.

Figure 3-5 shows 20 possible regions (as shaded areas). Conceptually, there is an unlimited number of possible regions. When deciding the actual number of regions to consider, there is a trade-off between the chance of hitting the "right" region and



Picture 1 Picture 2

The correlation coefficient of the two marked regions is 0.674.

Figure 3-4: Images with Regions Marked

the amount of noise introduced. This will be discussed further in Section 4.2.2.

In most of this thesis, we only consider the 20 possible regions shown in Figure 3-5. For each region, we consider both the original image in that region and the left-right mirror image of that region. This is because left-right mirror images occur very frequently in image databases and we would like to regard them as the same. Therefore, there are a total of 40 sub-pictures to consider. This translates into 40 instances per bag in the multiple-instance learning framework. Here, we do a little optimization to throw out regions whose variances are below a certain threshold, since low-variance regions are not likely to be interesting. Therefore, there may be fewer than 40 instances per bag, depending on the image. For each sub-picture, we process it with smoothing and sampling as described in Section 3.1.2, to get an $h \times h$ matrix which we treat as an h^2-dimensional feature vector.

3.3 Weighted Correlation Coefficient

Not all dimensions in the feature vector are equally important. For example, some of them may correspond to the background in the image, and we do not want them to carry the same weights as other dimensions. Therefore, we extend our correlation similarity measure to allow different dimensions to have different weight factors. We define a weighted correlation coefficient for two n-dimensional feature vectors f_1 and f_2 as:

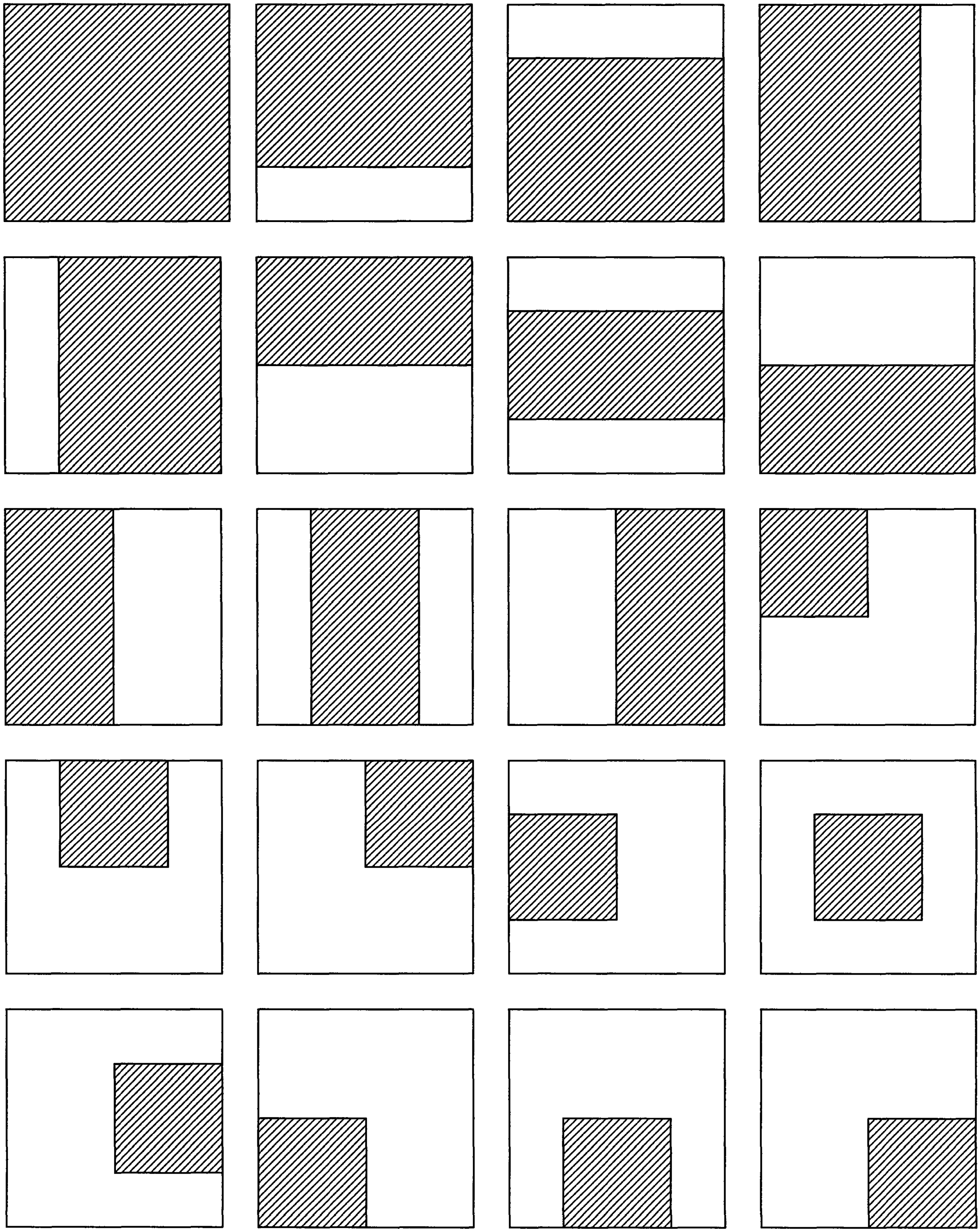


Figure 3-5: 20 Possible Regions to Consider

$$r = \frac{\frac{1}{n}\sum_{k=1}^{n} w_k^2 (f_1(k) - \overline{f_1})(f_2(k) - \overline{f_2})}{\sigma'_{f_1}\sigma'_{f_2}}$$

where w_k^2 is the non-negative weight for the k^{th} dimension, $\overline{f_1}$, $\overline{f_2}$ are the average values of $f_1(k)$ and $f_2(k)$, and σ'_{f_1}, σ'_{f_2} are the "weighted" standard deviations of $f_1(k)$ and $f_2(k)$, respectively:

$$\begin{aligned}
\overline{f_1} &= \frac{\sum_{k=1}^{n} f_1(k)}{n} \\
\overline{f_2} &= \frac{\sum_{k=1}^{n} f_2(k)}{n} \\
\sigma'_{f_1} &= \sqrt{\frac{1}{n}\sum_{k=1}^{n} w_k^2 (f_1(k) - \overline{f_1})^2} \\
\sigma'_{f_2} &= \sqrt{\frac{1}{n}\sum_{k=1}^{n} w_k^2 (f_2(k) - \overline{f_2})^2}
\end{aligned}$$

3.4 Fitting the Similarity Measure into Euclidean Space

Our similarity measure is defined as the weighted correlation coefficient on feature vectors, rather than Euclidean distance. This is not very convenient. However, there is a simple way to transform the vectors, so that we can use weighted Euclidean distance directly to reflect the weighted correlation coefficients of the original feature vectors.

Suppose that A_{ij} is the n-dimensional feature vector we have obtained for the i^{th} bag, j^{th} instance. w_k^2 is the weight factor for the k^{th} dimension. Define

$$B_{ij} = \frac{A_{ij} - \overline{A_{ij}}}{\sigma'_{A_{ij}}}$$

where $\overline{A_{ij}}$ is the average of A_{ij} entries, and $\sigma'_{A_{ij}}$ is the "weighted" standard deviation of A_{ij} entries:

$$\overline{A_{ij}} = \frac{\sum_{k=1}^{n} A_{ijk}}{n}$$

$$\sigma'_{A_{ij}} = \sqrt{\frac{1}{n}\sum_{k=1}^{n} w_k^2 (A_{ijk} - \overline{A_{ij}})^2}$$

With this definition, we are going to show that, comparing or ranking A_{ij} vectors based on weighted correlation coefficients is the same as comparing or ranking B_{ij} vectors based on weighted Euclidean distances in reverse order. The smaller the weighted Euclidean distance is between B_{ij} vectors, the higher the weighted correlation coefficient is between the corresponding A_{ij} vectors. This is formally stated as follows:

Claim For any i, j, l, m, p, q, u, v and weight factors $\{w_k^2\}$,

1. $Corr(A_{ij}, A_{lm}) > Corr(A_{pq}, A_{uv})$ if and only if $||B_{ij} - B_{lm}|| < ||B_{pq} - B_{uv}||$

2. $Corr(A_{ij}, A_{lm}) = Corr(A_{pq}, A_{uv})$ if and only if $||B_{ij} - B_{lm}|| = ||B_{pq} - B_{uv}||$

3. $Corr(A_{ij}, A_{lm}) < Corr(A_{pq}, A_{uv})$ if and only if $||B_{ij} - B_{lm}|| > ||B_{pq} - B_{uv}||$

where $Corr(\alpha, \beta)$ means the weighted correlation coefficient of α and β, and $||\alpha - \beta||$ means the weighted Euclidean distance between α and β.

Lemma For any i, j,

$$\sum_{k=1}^{n} w_k^2 B_{ijk}^2 = n$$

Proof of Lemma

$$\begin{aligned}
& \sum_{k=1}^{n} w_k^2 B_{ijk}^2 \\
= & \sum_{k=1}^{n} w_k^2 \left(\frac{A_{ijk} - \overline{A_{ij}}}{\sigma'_{A_{ij}}}\right)^2 \\
= & \frac{\sum_{k=1}^{n} w_k^2 (A_{ijk} - \overline{A_{ij}})^2}{\sigma'^2_{A_{ij}}} \\
= & \frac{n\sigma'^2_{A_{ij}}}{\sigma'^2_{A_{ij}}} \\
= & n
\end{aligned}$$

Proof of Claim

$$\begin{aligned}
& ||B_{ij} - B_{lm}|| \\
= & \sum_{k=1}^{n} w_k^2 (B_{ijk} - B_{lmk})^2 \\
= & \sum_{k=1}^{n} (w_k^2 B_{ijk}^2 + w_k^2 B_{lmk}^2 - 2w_k^2 B_{ijk} B_{lmk}) \\
= & n + n - 2\sum_{k=1}^{n} w_k^2 B_{ijk} B_{lmk} \\
= & 2n - 2\sum_{k=1}^{n} w_k^2 (\frac{A_{ijk} - \overline{A_{ij}}}{\sigma'_{A_{ij}}})(\frac{A_{lmk} - \overline{A_{lm}}}{\sigma'_{A_{lm}}}) \\
= & 2n - \frac{2\sum_{k=1}^{n} w_k^2 (A_{ijk} - \overline{A_{ij}})(A_{lmk} - \overline{A_{lm}})}{\sigma'_{A_{ij}} \sigma'_{A_{lm}}} \\
= & 2n - 2nCorr(A_{ij}, A_{lm})
\end{aligned}$$

Similarly,

$$||B_{pq} - B_{uv}|| = 2n - 2nCorr(A_{pq}, A_{uv})$$

and the Claim follows.

3.5 Bag Generation and Image Retrieval

Now we are ready to put everything together. For every image in our database, we do the following pre-processing:

1. If it is a color image, convert it into a gray-scale image.

2. Select some regions from the image, according to Section 3.2. Throw out regions whose variances are below a certain threshold.

3. Extract two sub-pictures from each region: one as the image itself in the region, and the other as its left-right mirror image. For each sub-picture, perform smoothing and sampling as described in Section 3.1.2 to get an $h \times h$ matrix. Treat this as an h^2-dimensional feature vector.

4. Transform each feature vector into a new one according to Section 3.4, i.e. subtract its mean from it and then divide it by its standard deviation. (All weights are 1 to start with.)

5. For each image in our database, we have obtained a number of feature vectors (after the transformation). Treat each one as an instance and put them together to form a bag for the image.

After these steps, our image database is ready to respond to user queries. The user is asked to select several positive and negative examples. The system puts together the corresponding image bags of multiple-instance data and feeds them into the Diverse Density (DD) algorithm. The DD algorithm returns an "ideal" point in the feature space as well as a set of feature weight values which maximize Diverse Density. Then the system goes to the image database and ranks all images based on their weighted Euclidean distances to the ideal point. (To find the distance from an image to the ideal point, it computes the distances of all of its instances to the point, and then picks the smallest one.) It then retrieves images in the ranked order. If the retrieval results are not satisfactory, the user may obtain better performance by picking out false positives and/or false negatives, adding them to the examples and training the system again.

Figure 3-6 shows a sample run of the system, to retrieve images that contain waterfalls.

3.6 Controlling Feature Weight Factors

The DD algorithm finds an "ideal" feature vector t and a set of weights w to maximize Diverse Density. For the task in Figure 3-6, we have shown the resulting t and w values (as 10×10 matrices) in Figure 3-7. It can be seen that, most of the weight factors are very close to zero, leaving only a few large weight values, which means that we are only using a small fraction of pixels to classify and retrieve images. Since we have very little training data, a too-simple concept based on a few pixels is likely

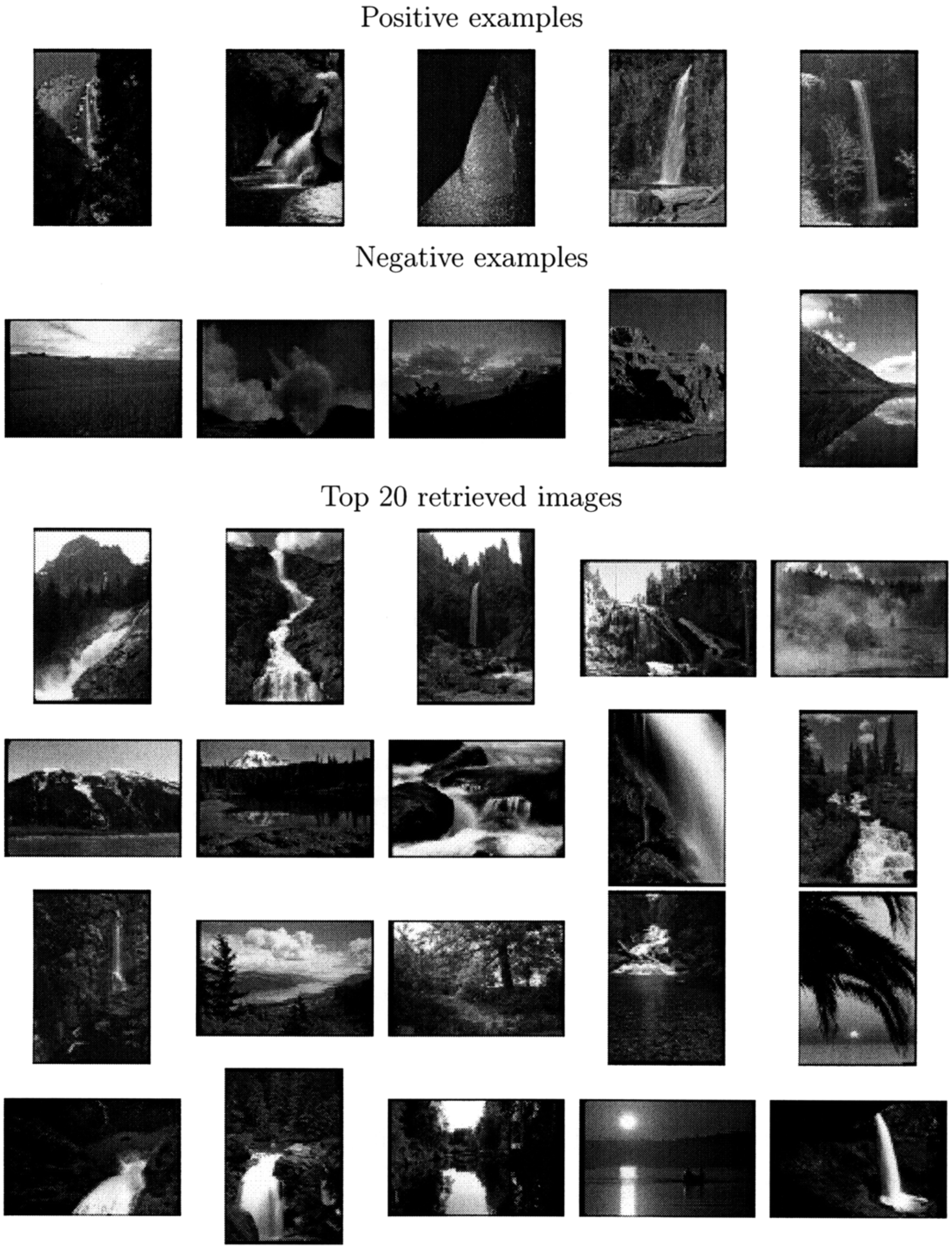


Figure 3-6: A Sample Run of the Image Retrieval System

to work well on the training set. However, it is not likely to generalize well, especially for complex image concepts. Although the performance in this example is reasonable, it is just a lucky case; the system does much worse on most other cases.

To address this issue, we have tried a number of alternate approaches, which will be discussed below.

3.6.1 Forcing All Weights to be the Same

This is the easiest modification to make. We simply force all weights to be 1 and maximize Diverse Density over the choice of feature point t only. Figure 3-8 shows its output on the same set of input data as used for Figure 3-7. As will be discussed in Section 4.2.1, this approach works well on the object image database, but not very well on the natural scene database.

3.6.2 "Hacking" with Partial Derivatives for Gradient Ascent

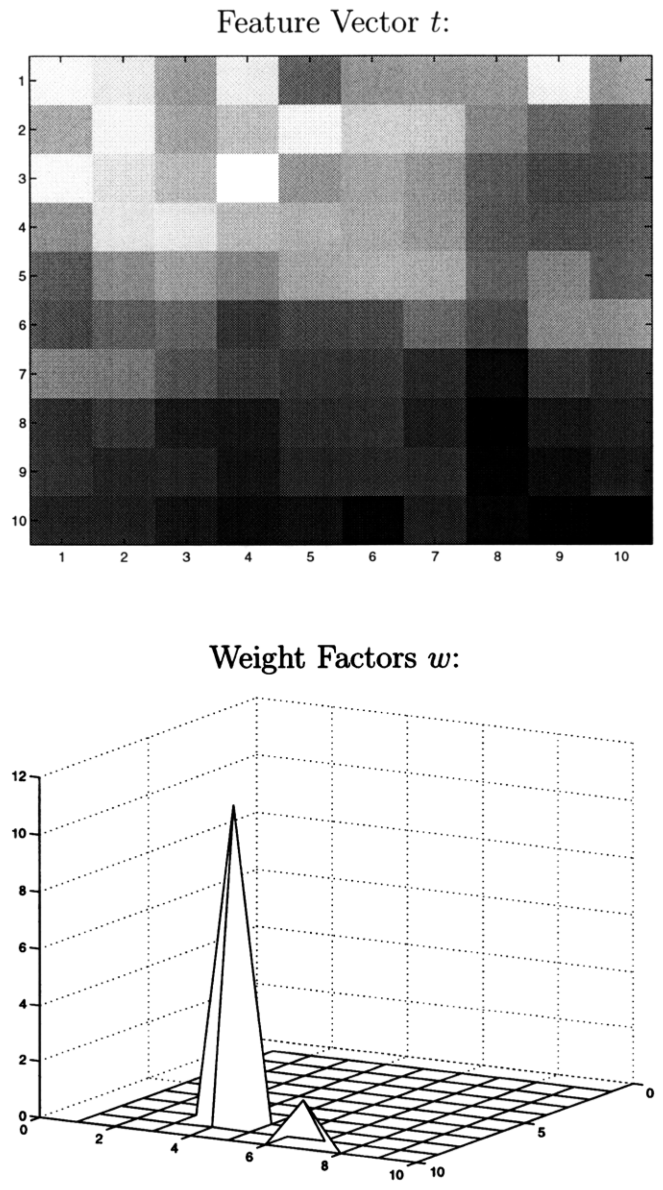
When the DD algorithm looks for maximum Diverse Density in a high-dimensional space, it uses a gradient ascent algorithm, which involves computing derivatives of the weighted Euclidean distance function $d(t, B_{ij}) = \sum_k w_k^2(t_k - B_{ijk})^2$ along all directions:

$$\begin{aligned}\frac{\partial d}{\partial t_k} &= 2w_k^2(t_k - B_{ijk}) \\ \frac{\partial d}{\partial w_k} &= 2w_k(t_k - B_{ijk})^2\end{aligned}$$

We experimented with a small "hack" to define $\frac{\partial d}{\partial w_k}$ as

$$\frac{\partial d}{\partial w_k} = 2w_k(t_k - B_{ijk})^2 \cdot \frac{1}{\alpha}$$

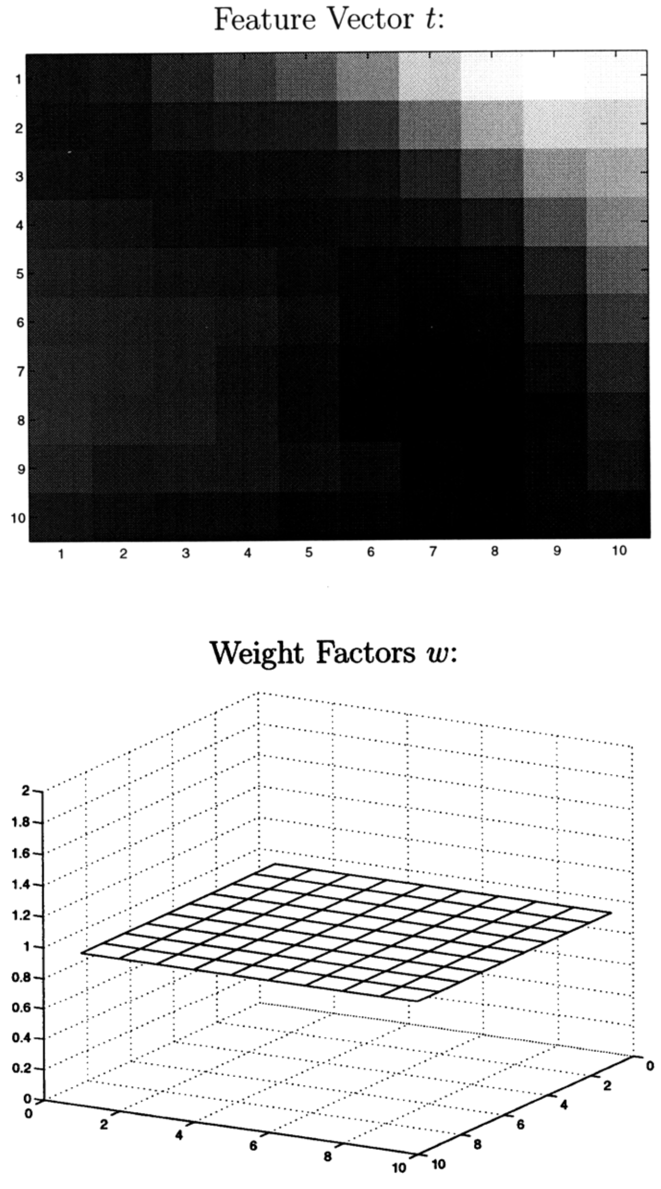
As we pick bigger and bigger α values, the gradient ascent process becomes more and more reluctant to move along w_k directions, effectively making fewer changes to the weights. The original DD algorithm corresponds to $\alpha = 1$; forcing all weights to be



Feature Vector t:

Weight Factors w:

Figure 3-7: DD Output for the Task in Figure 3-6


Feature Vector t:

Weight Factors w:

Figure 3-8: DD Output with Identical Weights

the same corresponds to $\alpha = \infty$. If we pick α somewhere in between, such as 50, the performance is occasionally better than both.

A problem with this method is that, it is hard to justify. We have modified $\frac{\partial d}{\partial w_k}$ but left $\frac{\partial d}{\partial t_k}$ intact, and now there is no simple target function that corresponds to these partial derivatives. Therefore, we do not know which function we are trying to maximize. Although it sometimes works better than the original DD algorithm, we can only say that it is just a hack, with little theoretical support.

3.6.3 Adding Inequality Constraints on Sum of Weights

Now we consider the option of constraining weight factors during the optimization process. Without loss of generality, we require that all weight factors be between 0 and 1: $0 \leq w_k \leq 1, k = 1, 2, ..., h^2$. ($h^2$ is the number of dimensions in the feature vectors.) We can limit the change in weight factors w_k by imposing the following constraint, which sets a lower bound for the sum of weights:

$$\sum_{k=1}^{h^2} w_k \geq \beta \cdot h^2$$

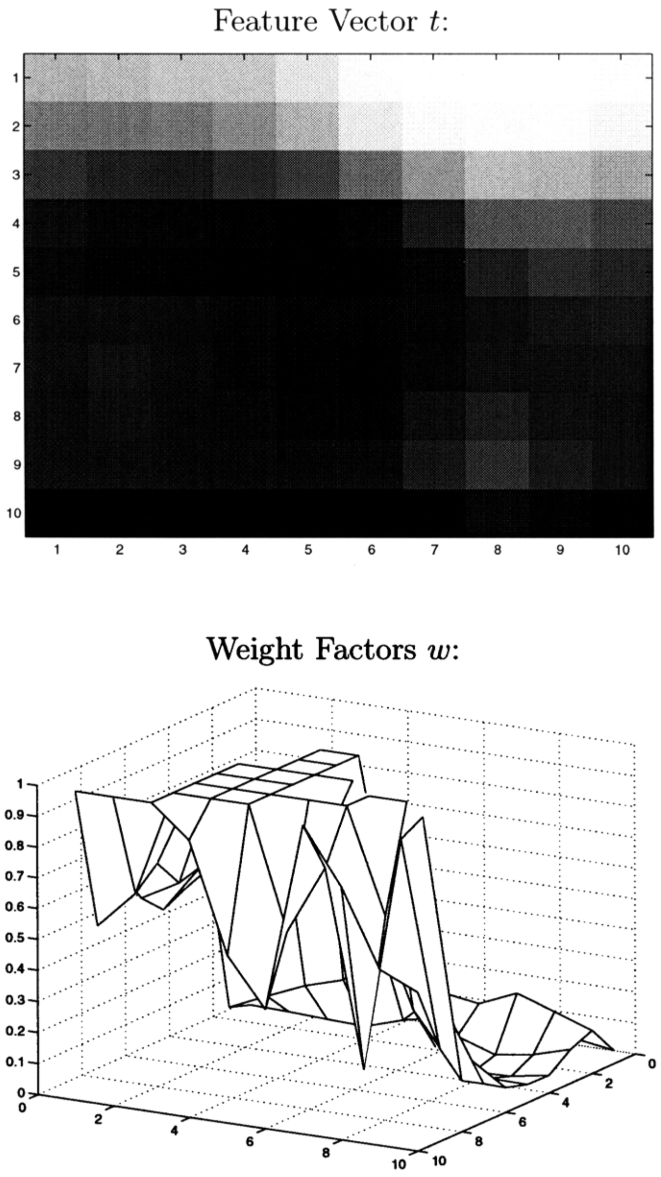
where β is a constant between 0 and 1. When $\beta = 0$, there is no restriction on the weights, and we are back to the original DD algorithm. When $\beta = 1$, we are forcing all weight factors w_k to be equal to 1. The restrictions on weight factors are easily controlled by changing β values. For example, when $\beta = 0.5$, the average of weight factors must be greater than 0.5, so no more than half of the weight factors can be close to zero.

The simple unconstrained minimization algorithm[1] used in the original DD method would no longer work to find the maximum with this new constraint. We switch to a more powerful algorithm called CFSQP (C code for Feasible Sequential Quadratic Programming) [6], which is capable of handling minimization problems with constraints. As will be shown in Section 4.2.1, this approach works well on a wide

[1] We often use the terms "maximization" and "minimization" interchangeably, since we maximize DD by minimizing -log(DD).

variety of situations.

Figure 3-9 shows its output on the same set of input data as used for Figure 3-7.


Feature Vector t:

Weight Factors w:

Figure 3-9: DD Output with Inequality Constraint and $\beta = 0.5$

Chapter 4

Results

4.1 Experimental Setup

We have tested our system on two different image databases. One is a natural scene image database, consisting of 500 pictures, 100 each for waterfalls, mountains, fields, lakes/rivers, and sunsets/sunrises. These are taken from the COREL library, the same database as used in [11]. Some examples are shown in Figure 4-1. The other one is an object image database, consisting of 228 pictures from 19 different categories, such as cars, airplanes, pants, hammers, cameras, etc. These are downloaded from the websites of AVIS Car Rental (www.avis.com), Bicycle Online (www.bicycle.com), Continental Airlines (www.flycontinental.com), Delta Airlines (www.delta-air.com), J. Crew (www.jcrew.com), JCPenney (www.jcpenney.com), Ritz Camera (www.ritzcamera.com), Sears (www.sears.com) and Sony (www.sony.com). Some examples are shown in Figure 4-2.

To simulate user feedback while minimizing user intervention, we followed the same experimental method as used in [11]:

The entire image database is split into a small *potential training set* and a large *test set*. The correct classifications for all images in the potential training set are known to the system. After the user selects positive and negative image examples, we generate corresponding bags and run the DD algorithm once, and then use the results to rank images from the potential training set. Since their correct classifications are already

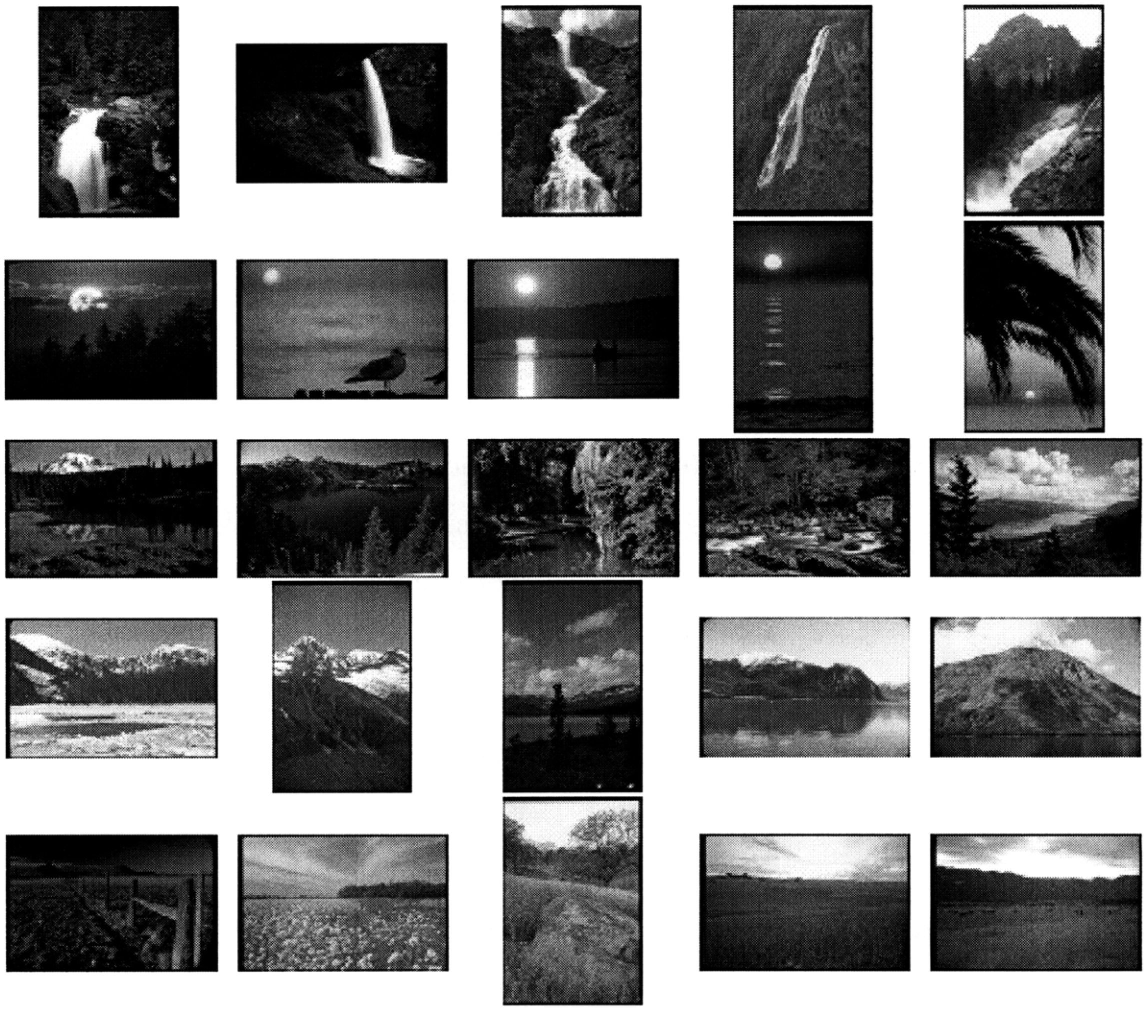


Figure 4-1: Sample Natural Scene Images

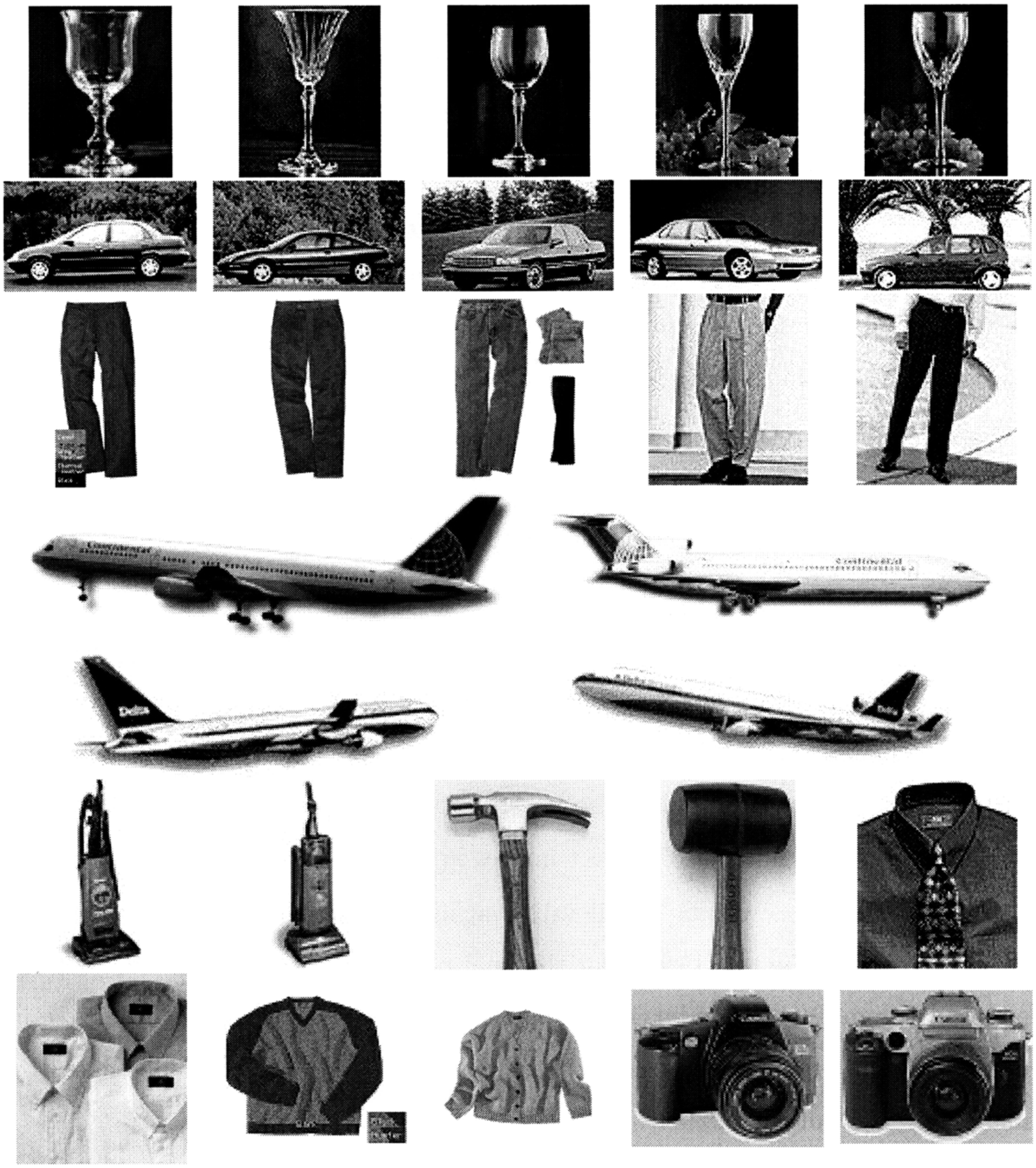


Figure 4-2: Sample Object Images

known, the system can evaluate its own performance on these images without asking the user. It can pick out some false positives and/or false negatives and add them to the examples to train itself again. This process can be repeated more than once, and it effectively simulates what a user might do to obtain better performance. In most experiments in this thesis, the system picks out top 5 false positives from the potential training set and adds them to the negative examples for a second round of training, and then picks out another top 5 false positives and trains for a third time. Finally it retrieves images from the larger test set.

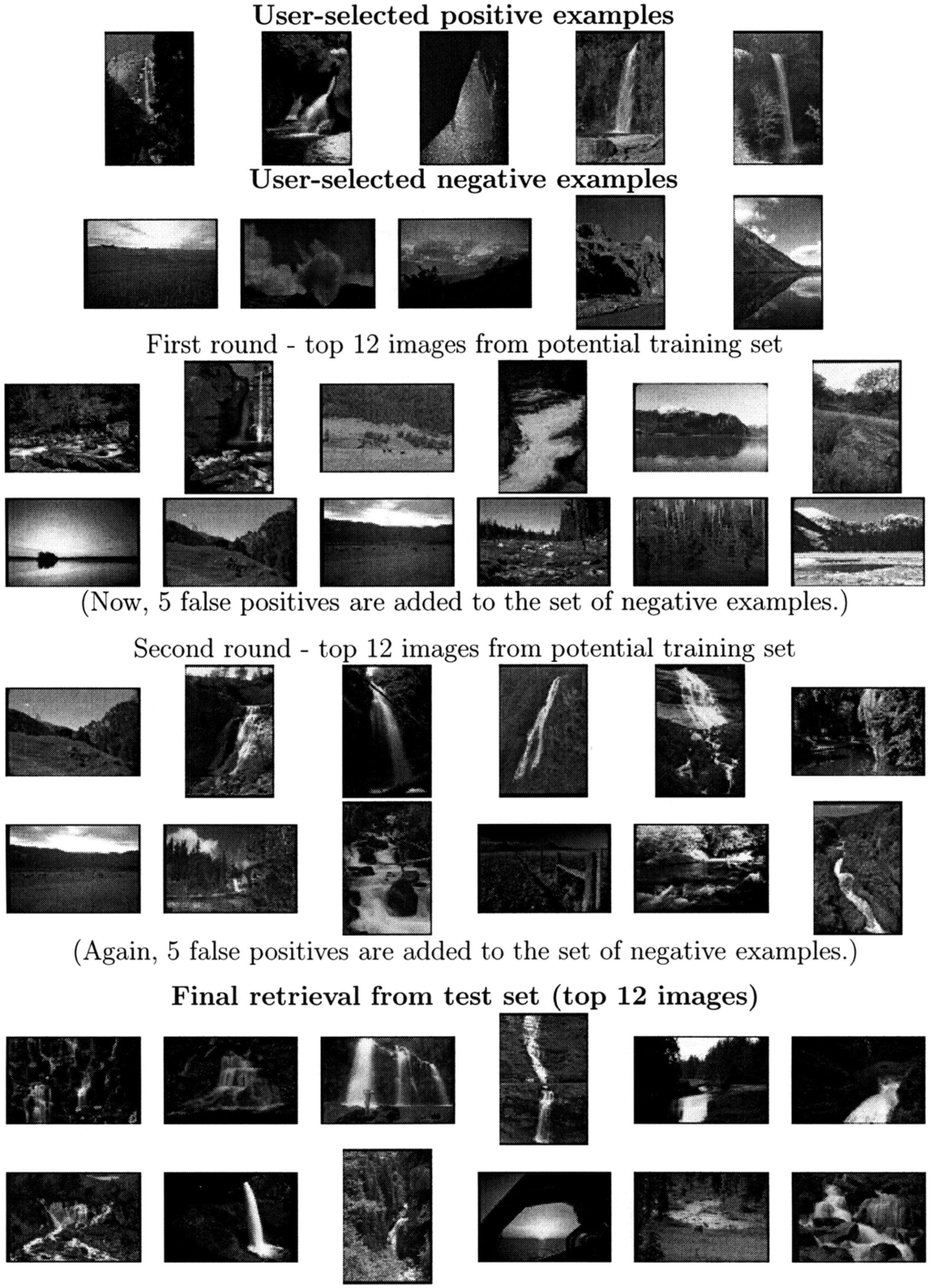
The selection of the potential training set can be either random or pre-defined. With random selection, a random seed allows the experiments to be repeatable. For most experiments in this chapter, 20% of images from each category are placed in the potential training set.

A sample run of the image retrieval system on the natural scene database is shown in Figure 4-3, where the user wants to retrieve images that contain waterfalls. A sample run on the object image database is shown in Figure 4-4, where the user wants to retrieve images that contain cars.

One way to evaluate image retrieval performance is to use precision-recall curves and recall curves. Precision is the ratio of the number of correctly retrieved images to the number of all images retrieved so far. Recall is the number of correctly retrieved images to the total number of correct images in the test database.

In a recall curve, we plot recall values against the number of images retrieved. Figure 4-5 shows the recall curve for the final retrieval result in Figure 4-3. A completely random retrieval of images would result in a recall curve as a 45-degree line from the lower-left corner to the upper-right corner. A better result is indicated by a more convex recall curve.

In a precision-recall curve, we plot precision values against recall values. Figure 4-6 shows the precision-recall curve for the retrieval result in Figure 4-3. In this graph, precision is 0.6 when recall is 0.15, which means: in order to obtain 15% of all waterfalls, 40% of the images retrieved are not waterfalls. A completely random retrieval of images would result in a precision-recall curve as a flat line at a level



User-selected positive examples

User-selected negative examples

First round - top 12 images from potential training set

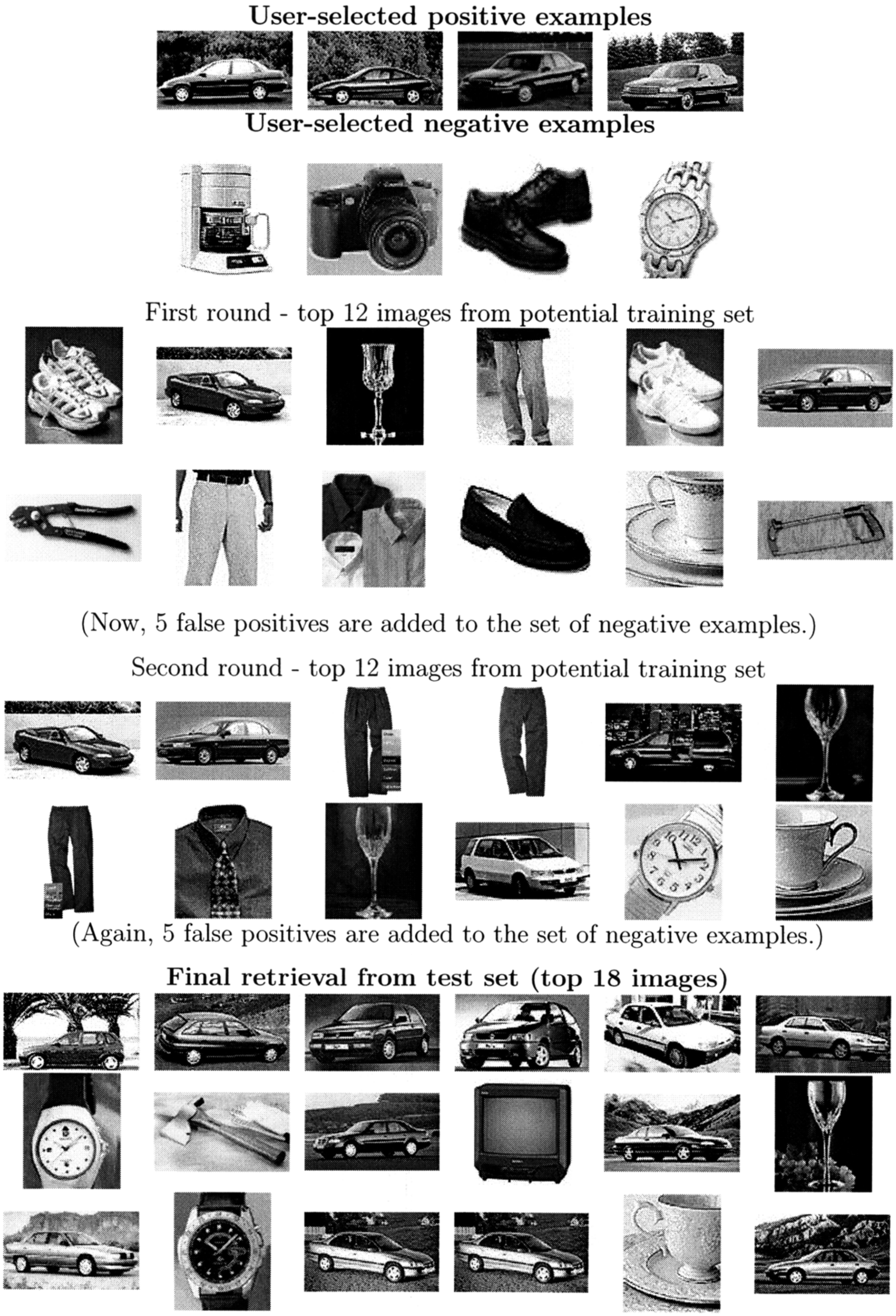
(Now, 5 false positives are added to the set of negative examples.)

Second round - top 12 images from potential training set

(Again, 5 false positives are added to the set of negative examples.)

Final retrieval from test set (top 12 images)

Figure 4-3: A Sample Run With 3 Rounds of Training: Retrieving Waterfalls



User-selected positive examples

User-selected negative examples

First round - top 12 images from potential training set

(Now, 5 false positives are added to the set of negative examples.)

Second round - top 12 images from potential training set

(Again, 5 false positives are added to the set of negative examples.)

Final retrieval from test set (top 18 images)

Figure 4-4: A Sample Run With 3 Rounds of Training: Retrieving Cars

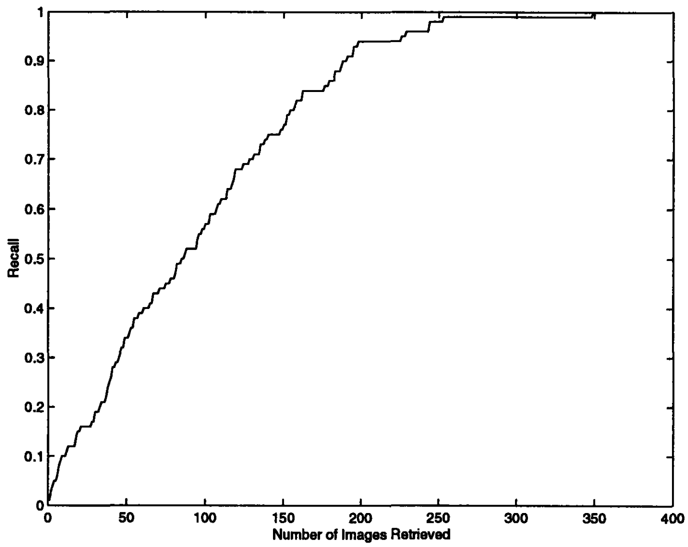


Figure 4-5: Recall Curve for Figure 4-3

indicating the percentage of correct images in the database. For our natural scene database, it would be a flat line at 0.2. A better result is indicated by a higher precision-recall curve. At the left end of a precision-recall curve, it is easy to see whether the first few retrieved images are correct. The curve is flat at 1.0 until the first incorrectly retrieved image brings it down.

If the first image retrieved is incorrect, followed by a few correct images, the precision-recall curve would look like Figure 4-7. This sometimes gives a misleading impression that the retrieval performance is bad, but it is not that bad after all. In Figure 4-7, the first retrieved image is not correct, but the following 7 images are all correct.

4.2 Comparisons

4.2.1 Using Different Weight Factor Controlling Methods

As we discussed in Section 3.6, there are different methods to control the feature vector weight factors to prevent them from becoming too imbalanced. Using precision-recall curves and recall curves, we compare the performance of 3 different approaches: the original DD algorithm, DD with identical weights and DD with the inequality

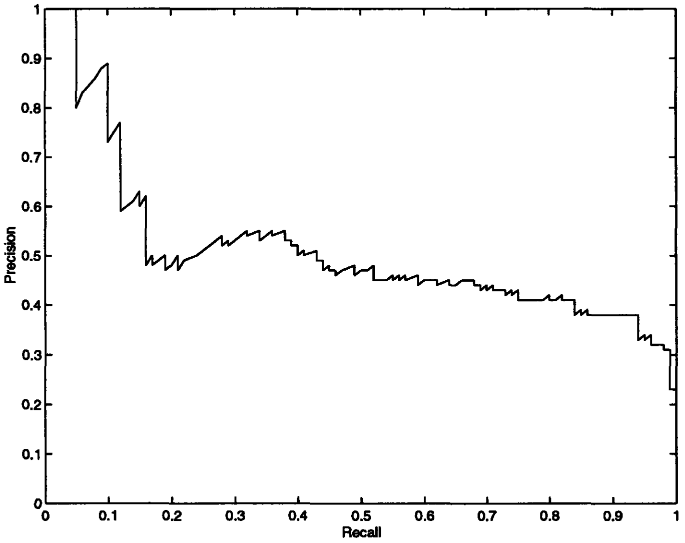


Figure 4-6: Precision-Recall Curve for Figure 4-3

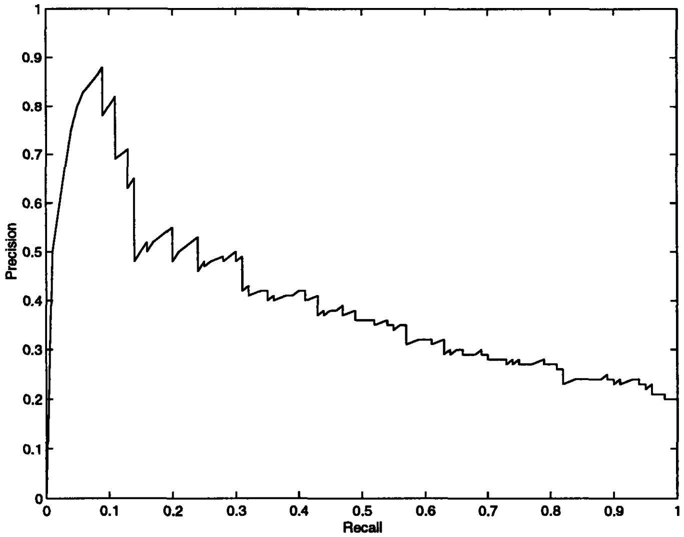


Figure 4-7: A Somewhat Misleading Precision-Recall Curve

constraint. Typical results for retrieving waterfalls, fields, sunsets/sunrises, cars, pants and airplanes are shown in Figures 4-8, 4-9, 4-10, 4-11, 4-12, 4-13, respectively. For the inequality constraint, we choose $\beta = 0.5$. In Figure 4-11, the performance of the inequality constraint method is not very good, but when we change β to 0.25, it works very well (see Figure 4-14).

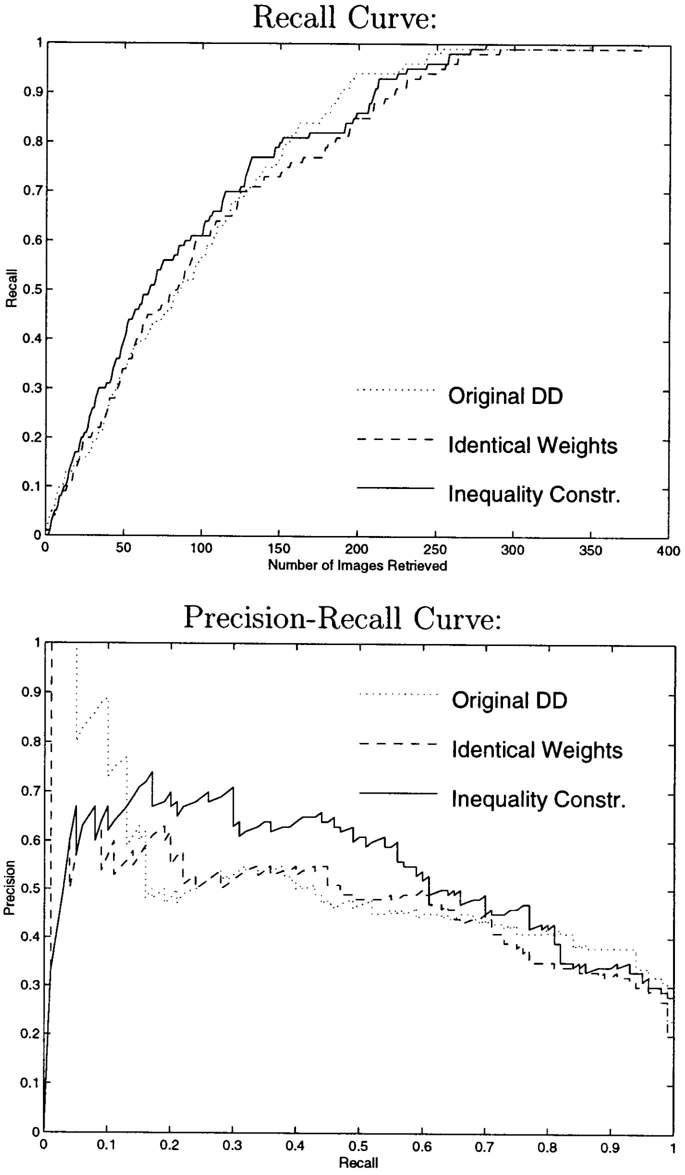
It can be seen that, although there is a lot of variation in the relative performance in different experiments, the inequality constraint method works very well (best or close to best) in a majority of test cases, especially for natural scenes. For the object image database, sometimes forcing all weights to be the same gives the best result. This is due to the fact that there is much less variation among objects in the object image database, and that most of our object images have uniform backgrounds, while the natural scenes have more variation and very noisy backgrounds.

The β value in the inequality constraint affects performance very much. For the experiment shown in Figure 4-10, we vary the β value and show the results in Figures 4-15, 4-16 and 4-17. As β moves towards 0, the precision-recall curve tends to move close to that of the original DD algorithm. As β moves towards 1, the precision-recall curve tends to move close to that of forcing all weights to be identical. This is consistent with our analysis in Section 3.6.3.[1]

4.2.2 Choosing Different Number of Instances Per Bag

Most of the experiments have been done with up to 40 instances per bag, by picking 20 different regions and taking mirror images. Figure 4-18 shows the effects of using fewer and more instances per bag. In general, having more instances per bag means a higher chance of hitting the "right" region. However, it also means introducing more noise which affects DD performance. Therefore, more instances per bag do not guarantee better performance. This is supported by Figure 4-18.

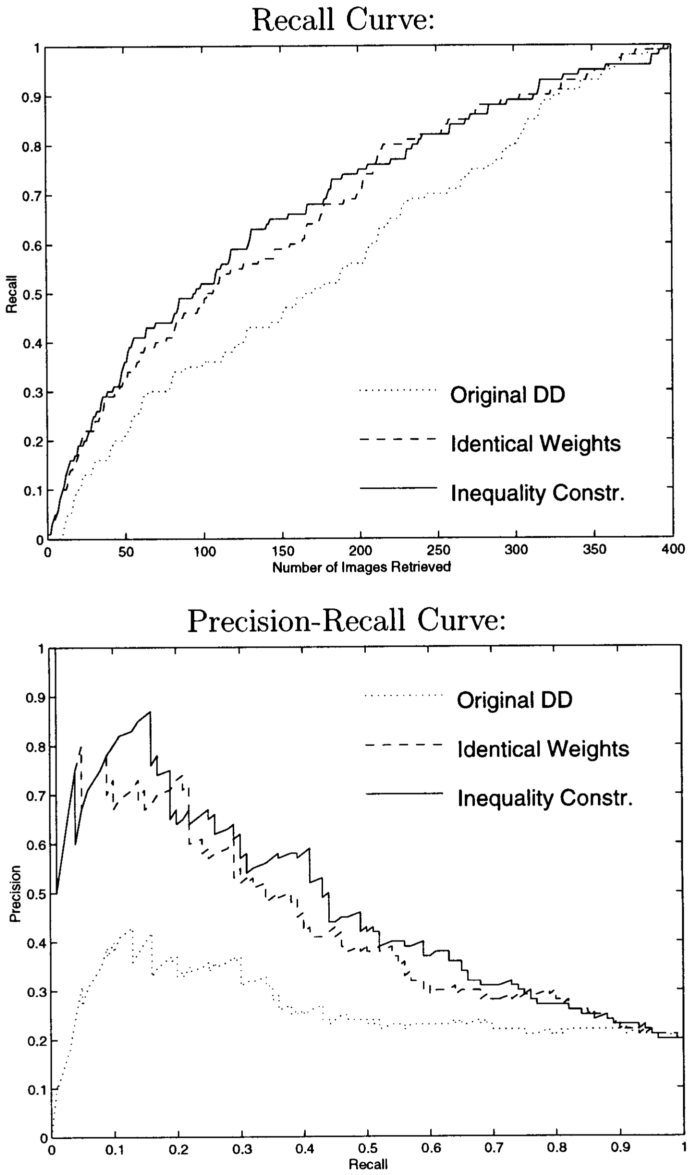
[1] When $\beta = 0$ or $\beta = 1$, the curve does not agree exactly with that of the original DD algorithm or that of identical weights. This is due to the difference in the minimization algorithms used.



Recall Curve:

Precision-Recall Curve:

Figure 4-8: Retrieving Waterfall Images


Recall Curve:

Precision-Recall Curve:

Figure 4-9: Retrieving Field Images

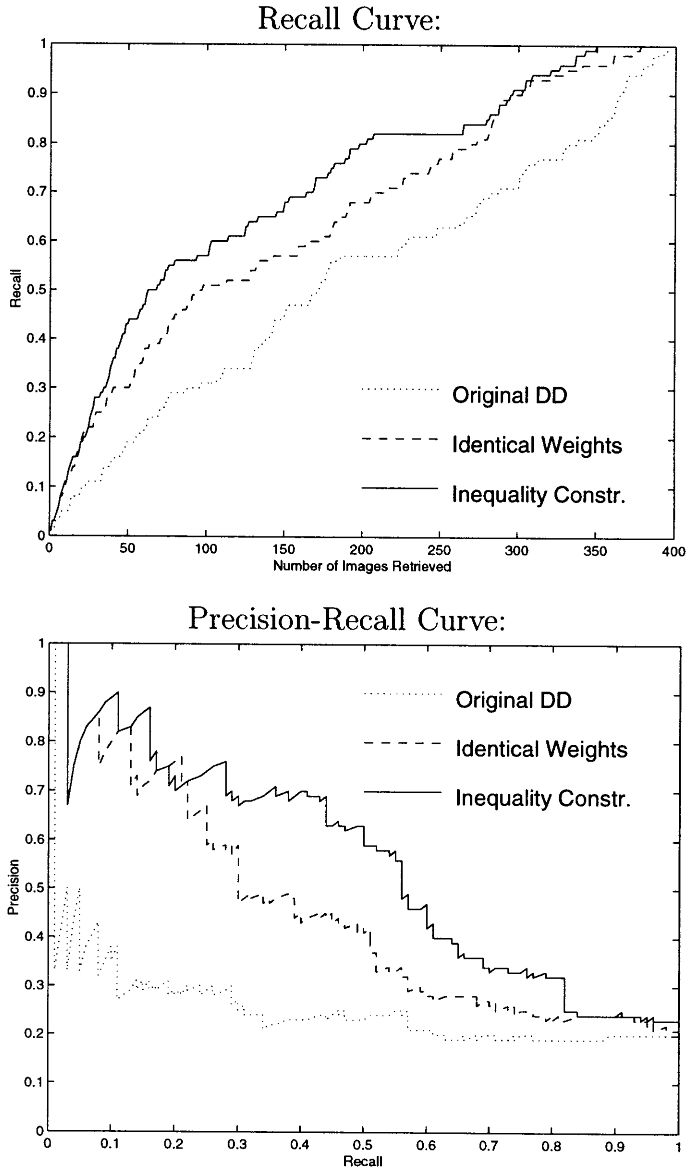
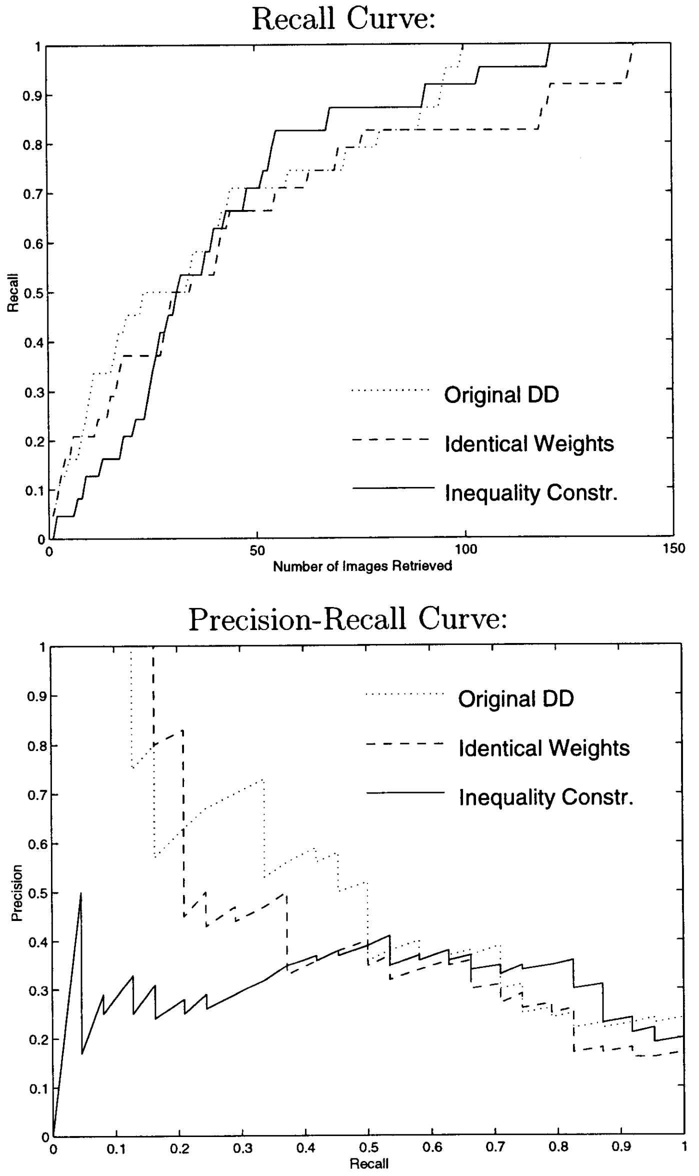
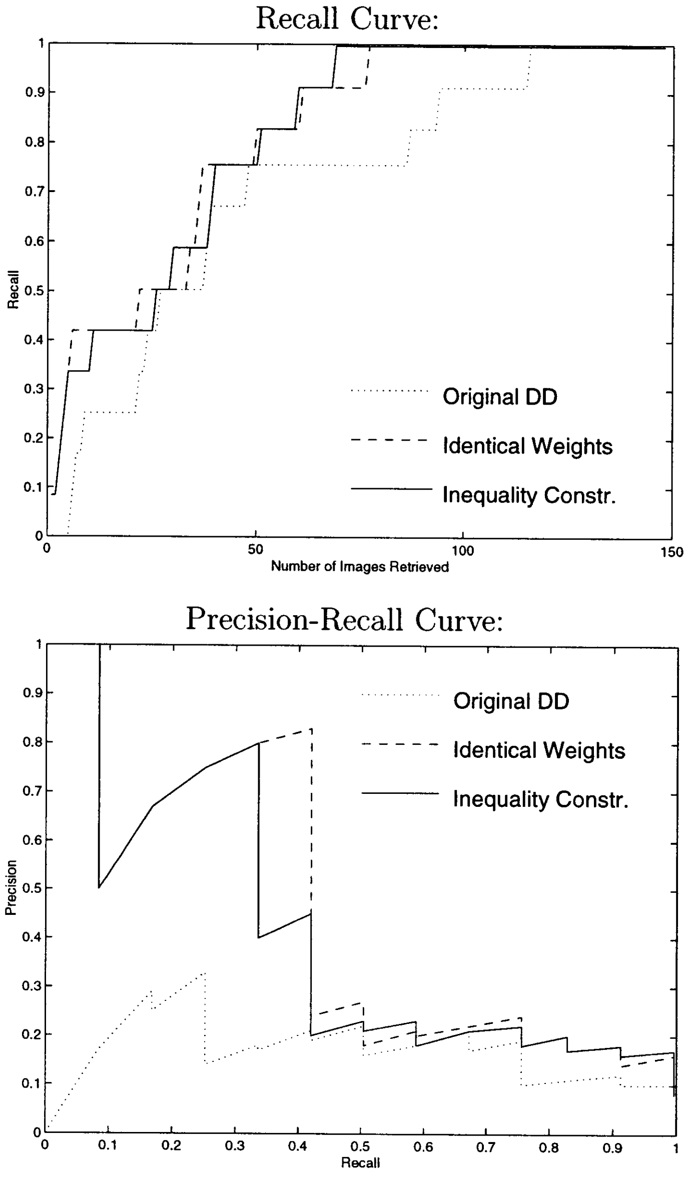


Figure 4-10: Retrieving Sunset/Sunrise Images


Recall Curve:

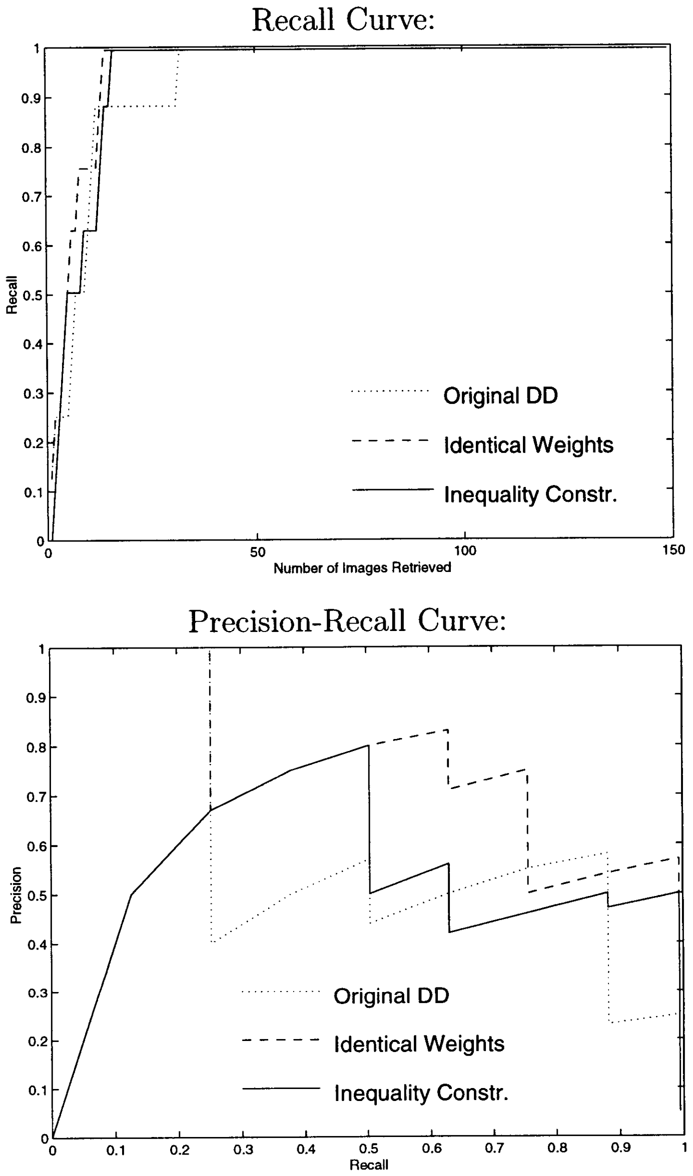
Precision-Recall Curve:

Figure 4-11: Retrieving Car Images


Recall Curve:

Precision-Recall Curve:

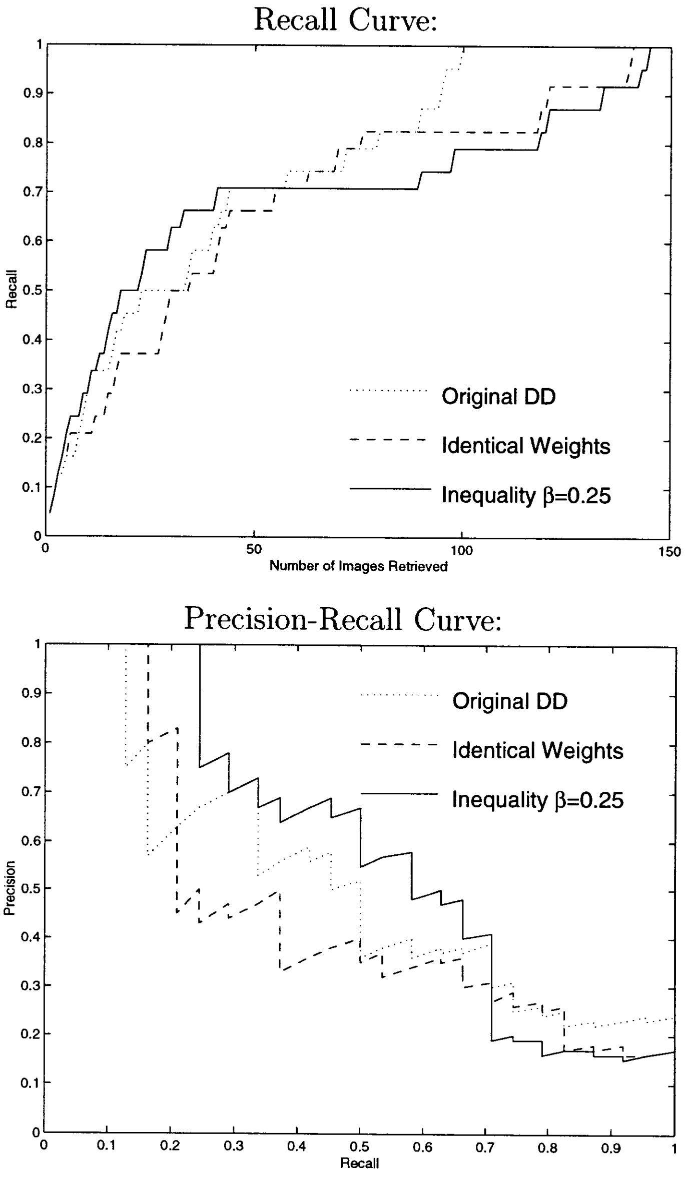
Figure 4-12: Retrieving Pant Images



Recall Curve:

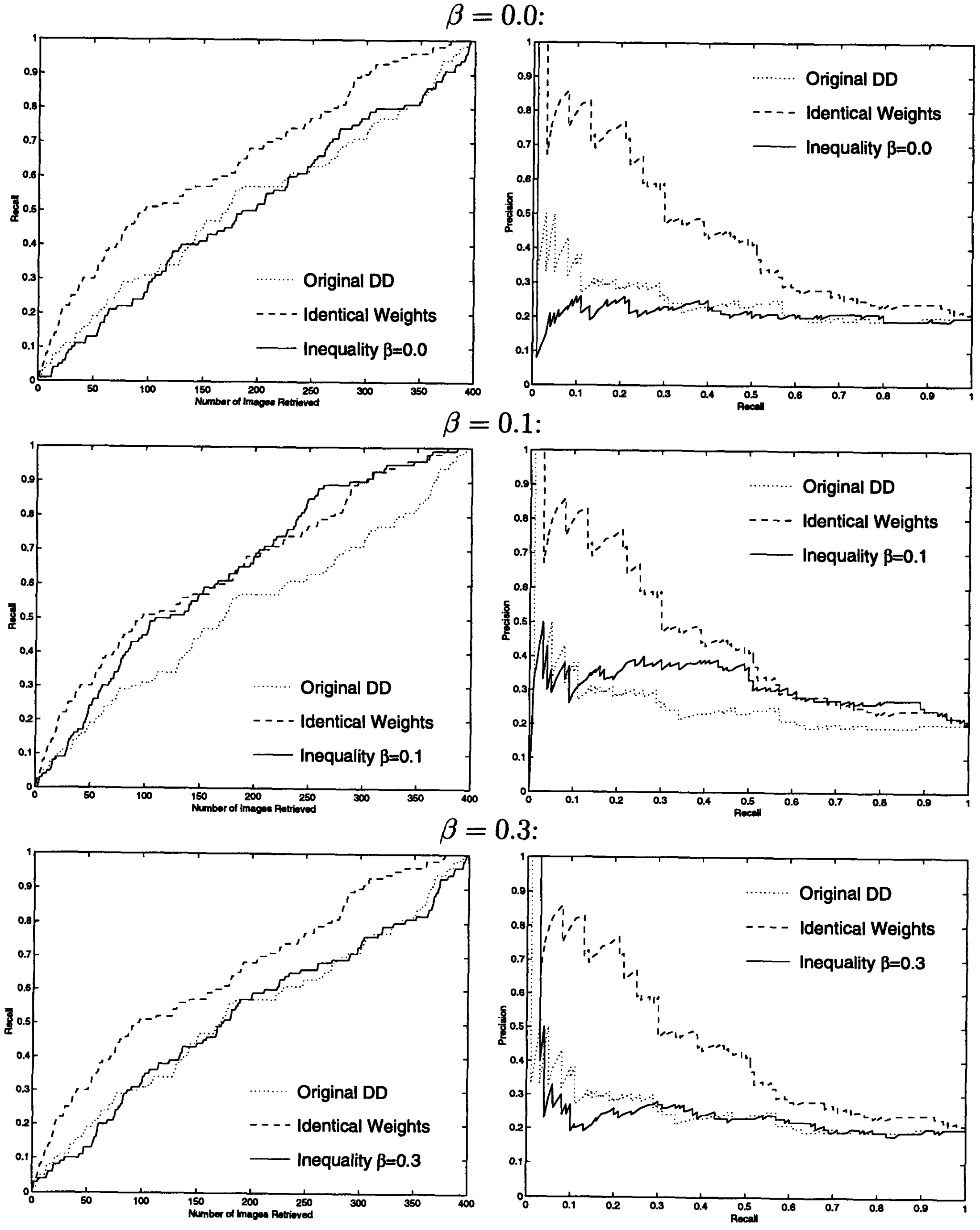
Precision-Recall Curve:

Figure 4-13: Retrieving Airplane Images


Recall Curve:

Precision-Recall Curve:

Figure 4-14: Retrieving Car Images ($\beta = 0.25$)

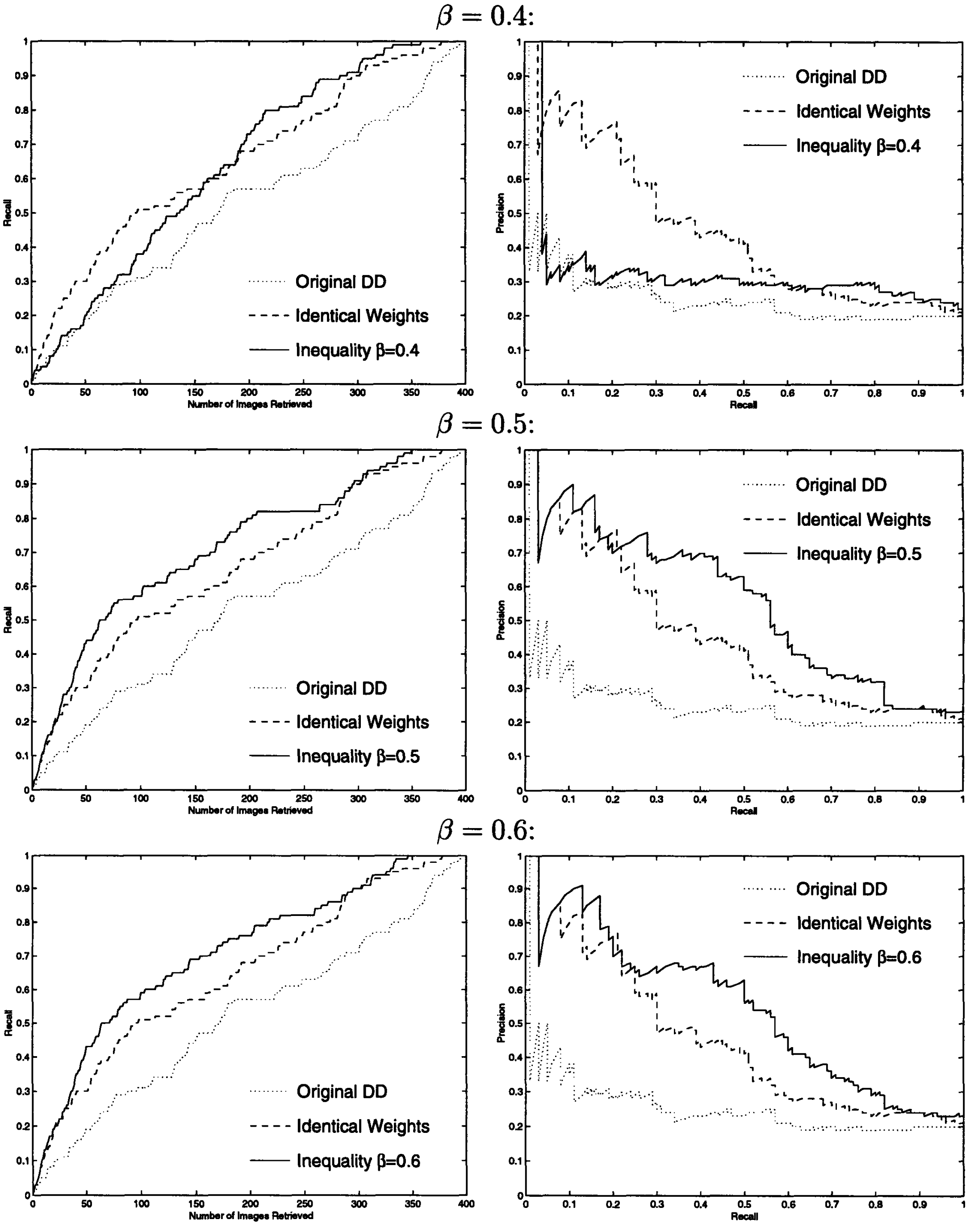


$\beta = 0.0$:

$\beta = 0.1$:

$\beta = 0.3$:

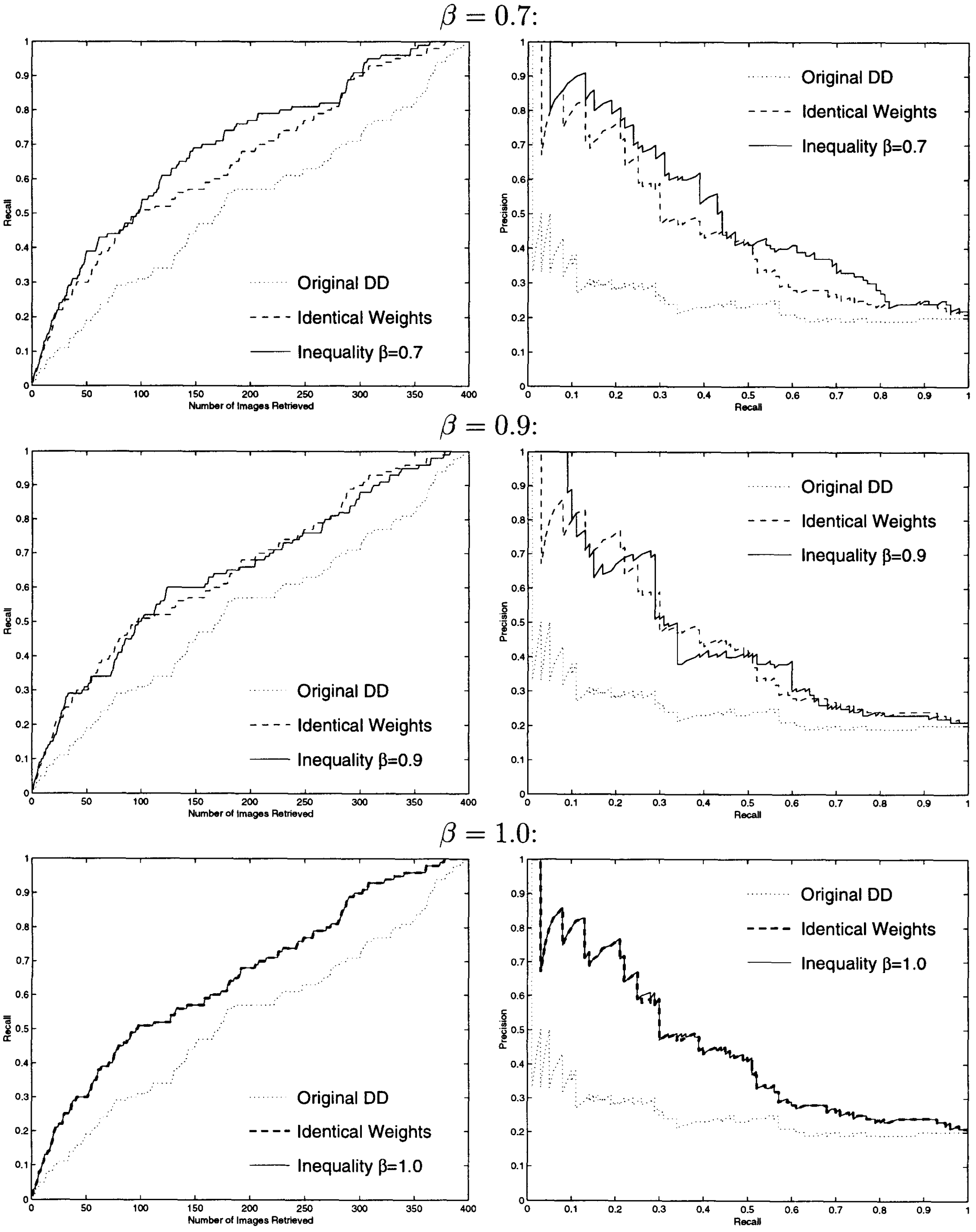
Figure 4-15: Changing β in the Inequality Constraint


$\beta = 0.4$:

$\beta = 0.5$:

$\beta = 0.6$:

Figure 4-16: Changing β in the Inequality Constraint (continued)

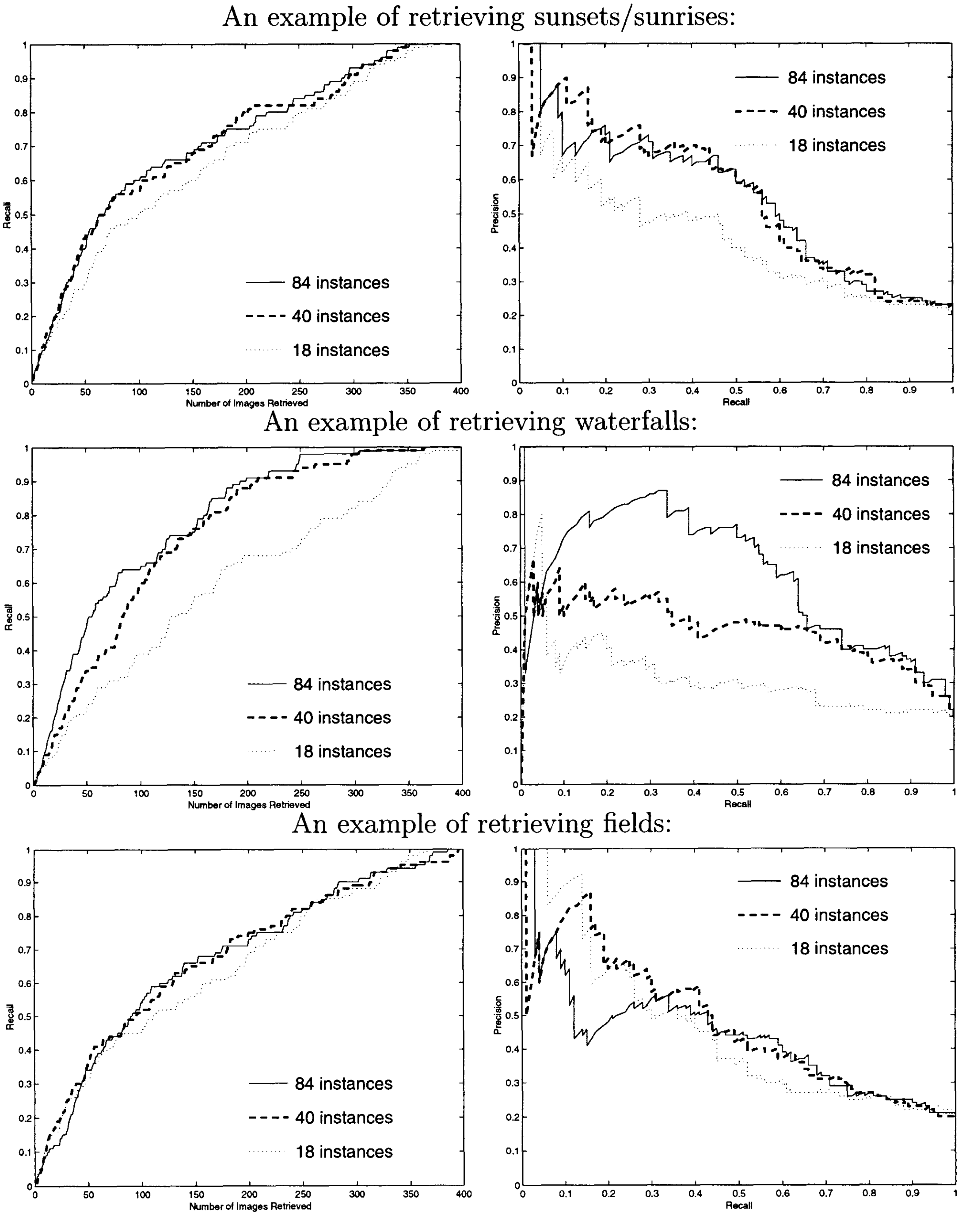


$\beta = 0.7$:

$\beta = 0.9$:

$\beta = 1.0$:

Figure 4-17: Changing β in the Inequality Constraint (continued)



An example of retrieving sunsets/sunrises:

An example of retrieving waterfalls:

An example of retrieving fields:

Figure 4-18: Choosing Different Number of Instances Per Bag

4.2.3 Changing Feature Vector Dimensions

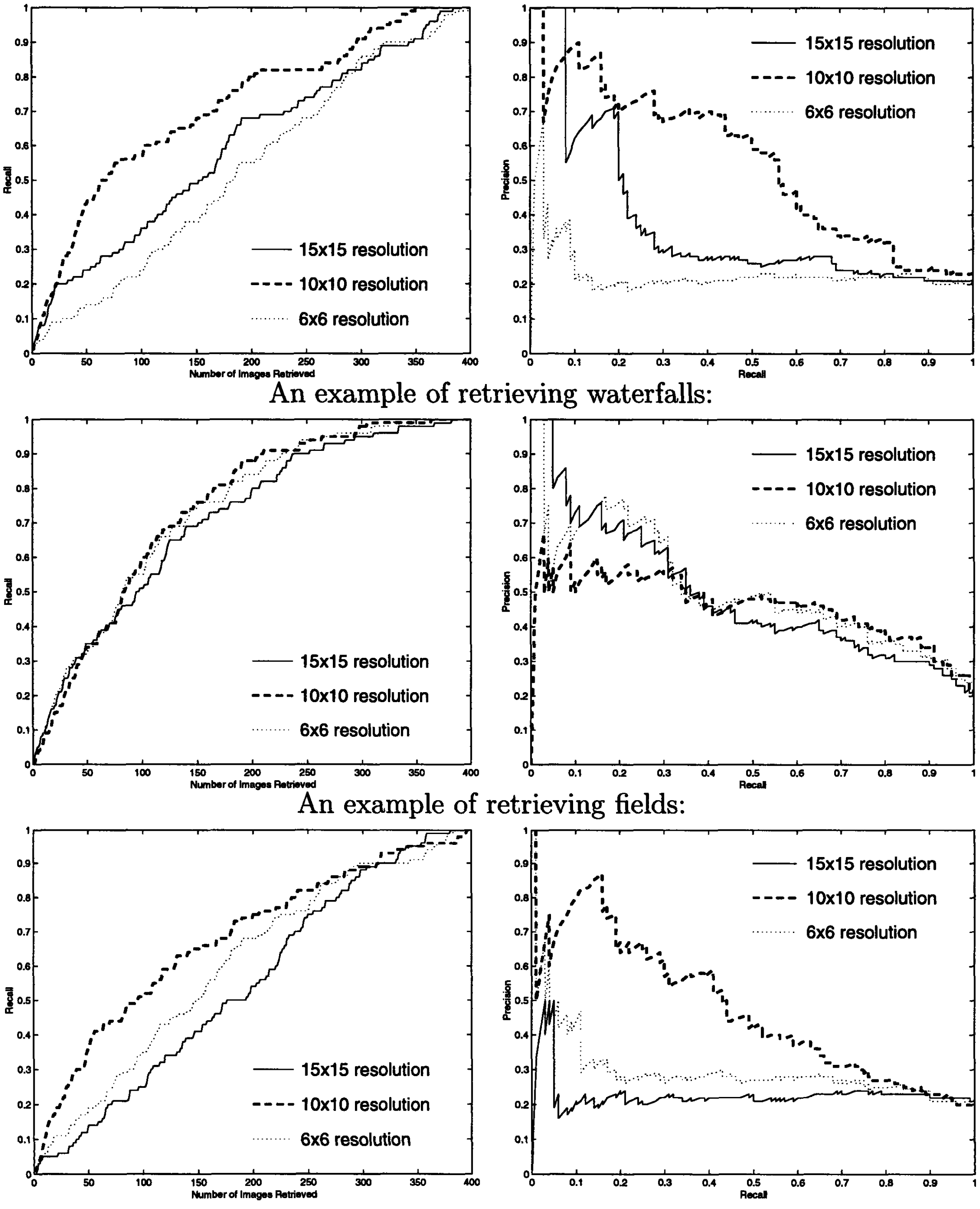
In most experiments, we smoothed and subsampled each image region to a low-resolution 10 × 10 matrix (a 100-dimensional feature vector) before comparing them against each other. We can use other resolutions (i.e. feature vector dimensions) as well. Figure 4-19 shows the effects of doing so. In many cases, as we increase the resolution, performance first rises, then declines. The problem with a very low resolution is that it does not give much information to compare for similarity. The problem with a very high resolution is that it makes our correlation similarity measure very sensitive to image shifts, and a higher resolution brings more noise. The "ideal" resolution which gives the best performance is highly dependent on the actual images.

4.2.4 Comparing with a Previous Approach

Now we compare our system with a previous approach developed by Maron and Lakshmi Ratan [11], which used the DD algorithm with image feature vectors of color statistics and color distribution patterns. With the natural scene database, the performance of our system (with either the original DD method or the inequality constraint method) is very close to that of [11], as shown in Figures 4-20 and 4-21. The approach in [11] has been specifically tuned to retrieving color natural scene images, and would not work with object images. Our system makes use of only gray-scale information from the images, and has obtained comparable results on the natural scene database without much tuning. Furthermore, it works with a wider range of image databases including object images.

4.3 Speeding Up Minimization Processes

The Diverse Density algorithm finds the maximum DD point by starting a minimization process from every instance in every positive bag. However, since every positive bag is supposed to contain an instance that is very close to the maximum DD point, we might be able to reach the maximum by starting from only a subset of positive

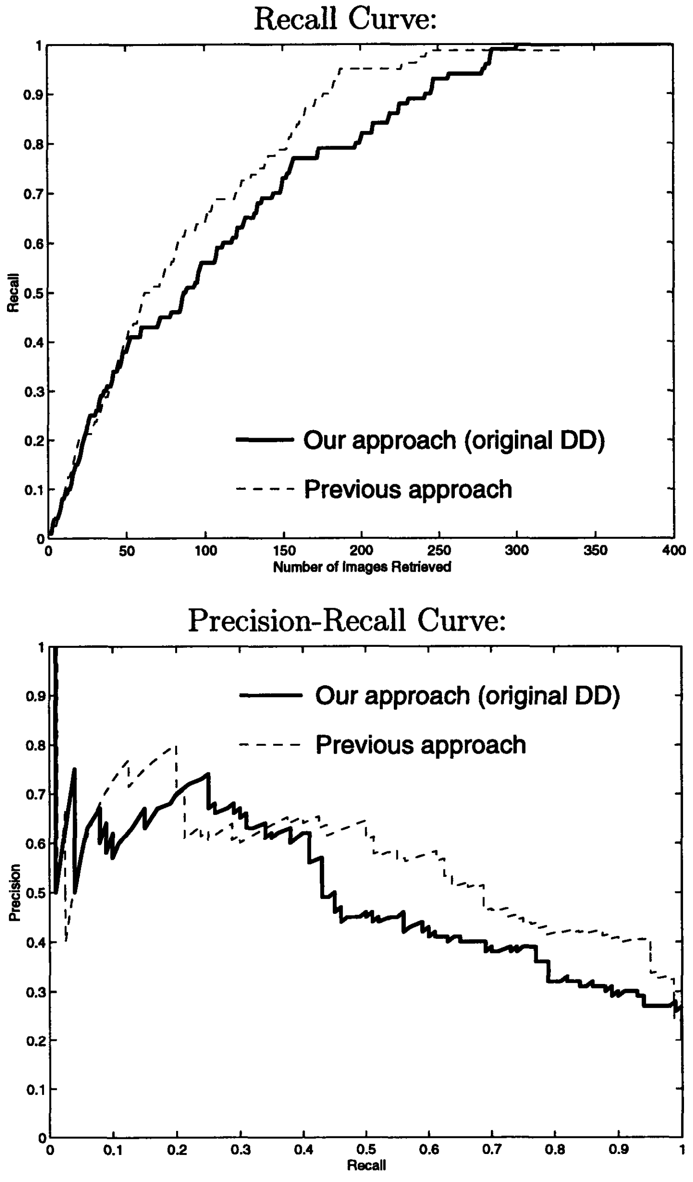


An example of retrieving sunsets/sunrises:

An example of retrieving waterfalls:

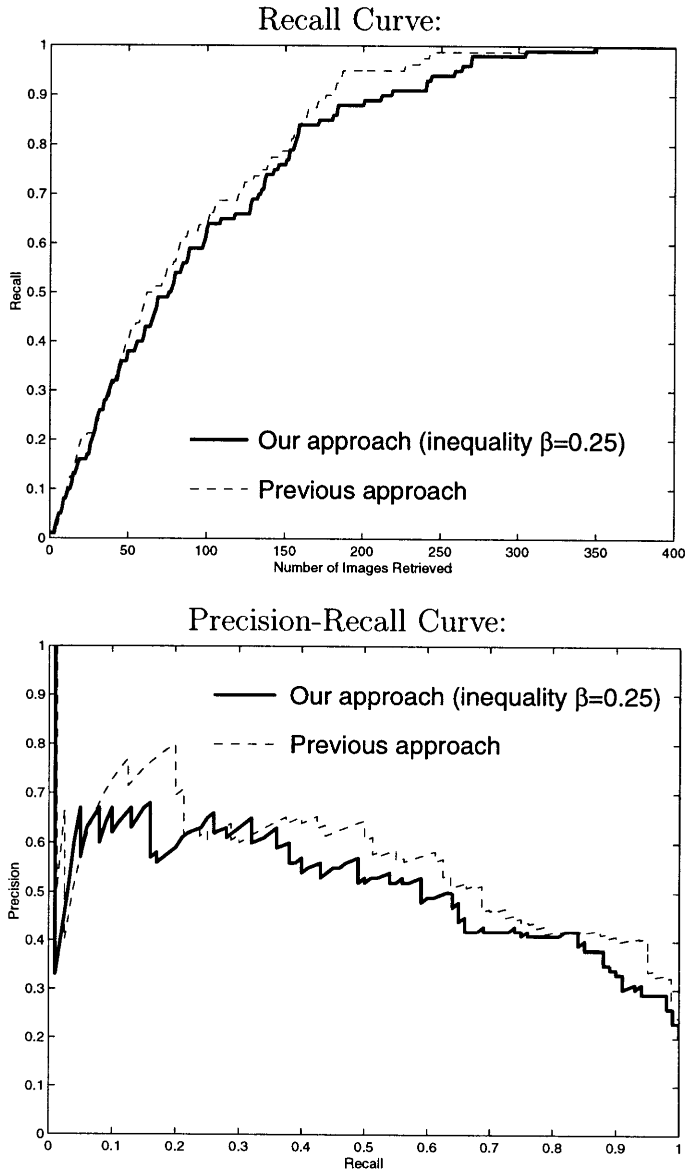
An example of retrieving fields:

Figure 4-19: Smoothing and Sampling at Different Resolutions



Retrieving Waterfall Images
(Solid line is our original DD approach)

Figure 4-20: A Comparison with a Previous Approach



Recall Curve:

Precision-Recall Curve:

Retrieving Waterfall Images
(Solid line is our inequality constraint approach with $\beta = 0.25$)

Figure 4-21: A Comparison with a Previous Approach (continued)

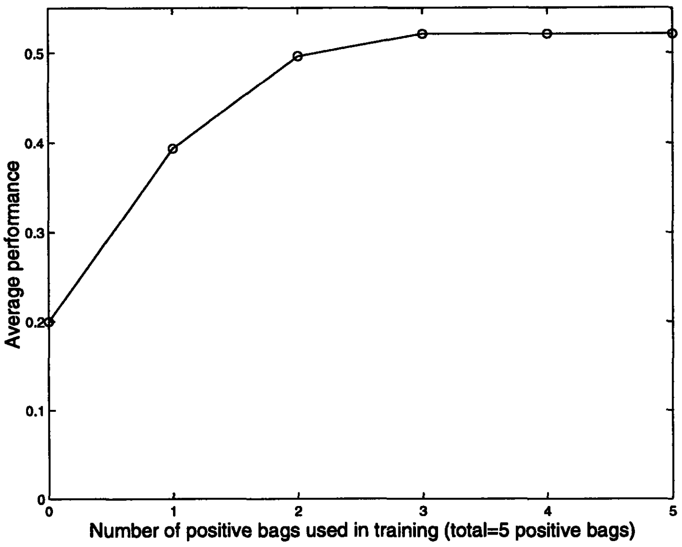


Figure 4-22: Start Minimization from a Subset of Positive Bags

bags. We have conducted a series of experiments in which the system picks a subset of positive bags and starts the minimization process from all instances in those bags. Figure 4-22 shows the performance of picking different number of positive bags as starting points, out of a total of 5 positive bags. The performance measure in this figure is somewhat arbitrarily defined as the average precision value for recall between 0.3 and 0.4 on precision-recall curves. Other performance measures are certainly possible; they would have generated similar comparison results as well.

It can be seen that, if we start from 2 out of 5 positive bags, the average performance is about 95% as good as the original approach. And if we start from 3 out of 5 positive bags, the results are indistinguishable from the original. Therefore, we can speed up the minimization processes and cut the training time significantly, without much sacrifice in performance.

Chapter 5

Conclusions and Future Work

We have presented a new approach to the problem of content-based image database retrieval, using a weighted correlation similarity measure and the Diverse Density multiple-instance learning techniques. We have built and tested a system which allows users to select positive and negative example images and then automatically retrieves similar pictures from a large database. As has been shown in the test results, this approach performs reasonably well on both natural scenes and object images.

Compared with a previous approach in [11] which was specifically tuned to retrieving natural scenes, our approach performs very close to theirs. Furthermore, our approach works very well on object image databases, which [11] was not designed to handle.

The treatment of feature space weight factors in the Diverse Density algorithm has significant effects on the performance of our system. The original Diverse Density algorithm gives the minimization process too much freedom, which drives most of the weight factors towards zero, leaving only a few large values. This is not desirable in the image retrieval domain. We experimented with several alternate approaches, including forcing all weights to be the same, modifying weight derivatives, and imposing different inequality constraints on the sum of weights. The system is quite sensitive to these changes. The inequality constraint approach usually works best or close to best, especially with the natural scene database. With the object image database, sometimes the best results are obtained by forcing all weights to be the same. This is due to the fact that most object images in our database have uniform backgrounds and little variation among objects, whereas the natural scene images have very noisy backgrounds and more variation.

We have studied the effects of putting more or fewer instances in each bag (by choosing more or fewer regions from each picture), and the effects of changing the number of feature vector dimensions (by smoothing and sampling image regions at different resolutions). Having more instances per bag does not guarantee better performance. Although the chance of hitting the "right" region increases as we put more instances into each bag, more irrelevant instances lead to more noise, which makes it more difficult for the DD algorithm to find the ideal point. On the other hand, as we increase the number of dimensions of each feature vector, performance first rises and then drops down in many cases. This is because a very low resolution does not give enough information to compare for similarity, while a very high resolution adds noise and also makes our correlation similarity measure very sensitive to image shifts.

In the original Diverse Density algorithm, minimization processes start from every point in every positive bag, trying to find the optimal answer. Our experiments have shown that, we can start from every point in only a subset of positive bags, and still get a near-optimal answer. If we start from points in only 2 bags out of 5 positive bags, the performance is about 95% as good as that of the original approach; if we start from points in 3 bags out of 5 positive bags, the results are indistinguishable from the original approach. Therefore, we can cut the training time significantly, without losing much accuracy.

In Figures 3-7, 3-8, and 3-9, we showed the output values of the DD algorithm in test cases where the system performed well. However, we have not been able to interpret those output values in an intuitive way. One possible future direction would be to explore those values in more detail, either to come up with reasonable interpretations, or to improve the algorithm so that it gives more intuitive output values which human can understand.

In Section 4.2.1, we discussed the effects of changing the β value in the inequality constraint. As another future direction, one might want to study how to choose β

automatically to get optimal performance.

All experiments shown in this thesis have been done on gray-scale images. Some attempts have been made to make use of color information in color natural scene images. We used RGB values separately and used a similar approach as we did with gray-scale images, tripling the number of dimensions of feature vectors. No significant improvements have been observed in this case. One other possible future direction would be to explore the effects of alternate color representation schemes, and to test on a larger variety of color images.

Also, one might want to try to use different feature vector representations and/or other similarity measures. We have attempted to preprocess the images with edge detection, and to use line and corner features in the feature vectors. However, the results we have got are not satisfactory.

Although our system is able to handle scaling changes across images, it is not designed to handle rotations. The correlation similarity measure can tolerate small rotations, but large rotations of the same object would be treated as dissimilar. One way to handle rotations would be to add more instances to represent different angles of view for each image region, although this would mean a significant increase in the number of instances per bag. There may be better ways, and this is yet another possible future direction.

The difficulty of content-based image retrieval problem is partly due to the ambiguous nature of images. The process of understanding an image is so complex that we cannot expect to write down explicit rules on how to achieve this. On the other hand, machine learning algorithms allow computers to learn from human users, and they are potentially capable of handling complex concepts without much low-level human intervention. Multiple-instance learning algorithms are specifically designed to learn from ambiguous data, and that is why it appears promising to open up a new direction in image retrieval research.

Bibliography

[1] J. S. De Bonet and P. Viola, "Structure driven image database retrieval", in *Advances in Neural Information Processing*, Vol. 10, 1997.

[2] T. G. Dietterich, R. H. Lathrop and T. Lozano-Pérez, "Solving the multiple-instance problem with axis-parallel rectangles", *Artificial Intelligence Journal*, 89, 1997, pp. 31-71.

[3] 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", *IEEE Computer*, Sept. 1995, pp. 23-30.

[4] W. I. Grosky and R. Mehrotra, "Index-based object recognition in pictorial data management", *Computer Vision, Graphics, and Image Processing*, Vol. 52, No. 3, 1990, pp. 416-436.

[5] R. Jain, R. Kasturi and B. G. Schunck, *Machine Vision*, McGraw-Hill, 1995.

[6] C. T. Lawrence, J. L. Zhou and A. L. Tits, *User's Guide for CFSQP Version 2.5: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints*, Institute for Systems Research TR-94-16r1, University of Maryland, College Park, 1997.

[7] P. Lipson, *Context and Configuration Based Scene Classification*, Ph.D. dissertation, Massachusetts Institute of Technology, 1996.

[8] P. Lipson, E. Grimson and P. Sinha, "Context and configuration based scene classification", in *Computer Vision and Pattern Recognition*, 1997.

[9] O. Maron, *Learning from Ambiguity*, Ph.D. dissertation, Massachusetts Institute of Technology, 1998.

[10] O. Maron and T. Lozano-Pérez, "A framework for multiple-instance learning", in *Advances in Neural Information Processing Systems*, Vol. 10, 1998.

[11] O. Maron and A. Lakshmi Ratan, "Multiple-instance learning for natural scene classification", in *Machine Learning: Proc. 15th International Conference*, 1998.

[12] R. Mehrotra and J. E. Gary, "Similar-shape retrieval in shape data management", *IEEE Computer*, Sept. 1995, pp. 57-62.

[13] T. M. Mitchell, *Machine Learning*, WCB/McGraw-Hill, 1997.

[14] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C: the Art of Scientific Computing*, 2nd Edition, Cambridge University Press, 1992.

[15] S. Ravela, R. Manmatha and E. M. Riseman, "Scale-space matching and image retrieval", *Proc. Image Understanding Workshop*, Vol. 2, 1996, pp. 1199-1207.

[16] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, 2nd Edition, Vol. 1, Academic Press, 1982.