Helsinki University of Technology
Dissertations in Computer and Information Science
Espoo 2006 Report D15

APPROACHES FOR CONTENT-BASED RETRIEVAL OF SURFACE DEFECT IMAGES

Jussi Pakkanen

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering for public examination and debate in Auditorium T2 at Helsinki University of Technology (Espoo, Finland) on the 20th of October, 2006, at 12 o'clock noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory of Computer and Information Science
P.O. Box 5400
FI-02015 HUT
FINLAND

Distribution:
Helsinki University of Technology
Laboratory of Computer and Information Science
P.O. Box 5400
FI-02015 HUT
FINLAND

Tel. +358 9 451 3272
Fax +358 9 451 3277

http://www.cis.hut.fi

Available in PDF format at `http://lib.tkk.fi/Diss/2006/isbn9512283530/`

ISBN 951-22-8352-2 (printed version)
ISBN 951-22-8353-0 (electronic version)
ISSN 1459-7020

Otamedia Oy
Espoo 2006

Pakkanen, J. (2006): **Approaches for Content-Based Retrieval of Surface Defect Images**. Doctoral thesis, Helsinki University of Technology, Dissertations in Computer and Information Science, Report D15, Espoo, Finland.

Keywords: surface inspection, content-based image retrieval, tree-structured self-organizing neural networks, self-organizing maps, Evolving Tree.

ABSTRACT

There are two properties which all industrial manufacturing processes try to optimize: speed and quality. Speed can also be called throughput and tells how much products can be created in a specified time. The higher speeds you have the better. Quality means the perceived goodness of the finished product. Broken or defective products simply don't sell, so they must be eliminated.

These are contradicting goals. The larger the manufacturing volumes, the less time there is to inspect a single product, or the more inspectors are required. A good example is paper manufacturing. A single paper machine can produce a sheet of paper several meters wide and several hundred kilometers long in just a few hours. It is impossible to inspect these kinds of volumes by hand.

In this thesis the indexing and retrieval of defect images taken by an automated inspection machine is examined. Some of the images taken contain serious defects such as holes, while others are less grave. The goal is to try to develop automated methods to find the serious fault images from large databases using only the information in the images. This means that there are no annotations. This is called *content-based image retrieval*, or CBIR.

This problem is examined in two different ways. First the PicSOM CBIR tool's suitability for this task is evaluated. PicSOM is a platform for content-based image retrieval developed at the Laboratory of Computer and Information Science, Helsinki University of Technology. PicSOM has earlier been succesfully applied to various different CBIR tasks.

The other part involves developing new algorithms for efficient indexing of large, high-dimensional databases. The *Evolving Tree* (ETree), a novel hierarchical, tree-shaped, self-organizing neural network is presented and analyzed. It is noticeably faster than classical methods, while still obtaining good results.

The suitability and performance of both CBIR and ETree on this problem is evaluated using several different experiments. The results show that both approaches are applicable for this real world quality inspection problem with good results.

Pakkanen, J. (2006): **Pintavirhekuvien sisältöpohjaisesta hausta**. Väitöstyö, Teknillinen korkeakoulu, Dissertations in Computer and Information Science, Raportti D15, Espoo, Suomi.

Avainsanat: pinnantarkistus, sisältöpohjainen kuvahaku, puumaiset neuroverkot, itseorganisoivat kartat, Evolving Tree.

TIIVISTELMÄ

Kaikki teolliset tuotantolaitokset pyrkivat optimoimaan kahta ominaisuutta: nopeutta ja laatua. Nopeus kertoo kuinka nopeasti tuotteet pystytään valmistamaan. Mitä suurempi valmistunopeus on, sitä parempi. Laatu taas mittaa lopullisen tuotteen subjektiivista hyvyyttä. Koska viallisia tuotteita ei yleensä saa kaupaksi, ne on poistettava.

Nämä ovat vastakkaisia tavoitteita. Mitä enemmän tuotetta valmistetaan, sitä vähemmän aikaa yksittäisen kappaleen tarkastamiseen jää. Hyvänä esimerkkinä käy paperin valmistaminen. Normaali paperikone tuottaa muutamassa tunnissa monta sataa kilometriä paperia. Näin suurien tuotantomäärien tarkastaminen käsin on mahdotonta.

Tässä väitöskirjassa tutkitaan automaattisen tarkastuskoneen ottamien kuvien indeksointia ja hakua. Jotkut näistä kuvista sisältävät vakavia virhetilanteita, kuten reikiä. Toiset ovat vähemmän vakavia. Tavoitteena on kehittää automaattisia menetelmiä, jotka löytävät vakavat viat suurista tietokannoista käyttäen hyväksi vain kuvissa olevaa tietoa. Kuvia ei siis ole etukäteen luokiteltu. Tätä kutsutaan *sisältöpohjaiseksi hauksi*.

Tätä ongelmaa tutkitaan kahdella eri tavalla. Ensiksi selvitetään PicSOM-työkalun soveltuvuus tähän ongelmaan. PicSOM on Teknillisen korkeakoulun Informaatiotekniikan laboratoriossa kehitetty sisältöpohjaisen kuva-analyysin tutkimusalusta. PicSOM:ia on aiemmin käytetty menestyksekkäästi moniin sisältöpohjaisen haun ongelmiin.

Tämän lisäksi kehitetään uusia algoritmeja suurten ja korkeaulottuvuuksisten tietokantojen indeksointiin. Tuloksena on *Evolving Tree* (ETree), uusi hierarkkinen ja puumainen itseorganisoiva neuroverkko. Se on huomattavasti nopeampi kuin klassiset menetelmät mutta saavuttaa silti hyviä tuloksia.

Sisältöpohjaisen haun ja ETree:n soveltuvuutta tähän ongelmaan tutkitaan monin erilaisin kokein. Tulokset osoittavat, että molemmat lähestymistavat tuottavat hyviä tuloksia sovellettuna tähän tosielämän laaduntarkastusongelmaan.

Preface

The thesis you are holding in your hands represents over four years of work done at the Laboratory of Computer and Information Science, Helsinki University of Technology. During this time I have had the pleasure of working with several fascinating people, who have earned my gratitude.

The first one is the lab leader and my thesis supervisor Erkki Oja. Jukka Iivarinen worked diligently as my thesis instructor, gave insightful comments on various research problems as well as fought the bureaucratic war. I would also like to thank Jorma Laaksonen, Markus Koskela as well as the entire PicSOM group for their assistance and for indoctrinating me into the lab all those years ago. Thanks also to all the other members of our group: Antti Ilvesmäki, Rami Rautkorpi, and Petri Turkulainen for doing all of the actual work while I did my best to look busy.

The work in this thesis was done as part of the Knowledge Mining and Managing in a Distributed Image Datawarehouse (DIGGER) project. Our funding partners in this project have been Tekes (The Finnish Funding Agency for Technology and Innovation) and our industrial partner ABB Oy. My sincerest thanks to them, particularly to Mr. Juhani Rauhamaa of ABB Oy, for his insightful guidance on surface inspection. Portions of the work have been done under the Helsinki Graduate School in Computer Science and Engineering (HeCSE).

I have received financial grants from the Olga and Vilho Linnamo Foundation and TES (Tekniikan edistämissäätiö). Their support is gratefully acknowledged.

The manuscript was reviewed by Dr. Tech. Matti Niskanen, University of Oulu and by professor Andreas Rauber, Technische Universität Wien. I thank them for their time and encouraging comments.

To counter the seriousness of work I am glad to have several friends, with whom I have had loads of fun doing all sorts of pointless, silly and annoying things. Lastly I wish to thank my family for their continued support.

Otaniemi, September 2006

Jussi Pakkanen

Contents

Preface **vii**

Publications of the thesis **xiii**

List of abbreviations **xv**

1 Introduction **1**

1.1 Contributions of the thesis 2

1.2 Outline of the thesis 2

1.3 Author's contributions to publications 3

2 Visual quality assurance for surface defect images **5**

2.1 Phases of content-based visual quality assurance 5

2.1.1 Taking the image 5

2.1.2 Preprocessing 6

2.1.3 Segmentation 8

2.1.4 Feature extraction 8

2.1.5 Training a CBIR system 9

2.1.6 Queries 10

2.2 A word about difficulty 10

3 Using content-based image retrieval for surface inspection **11**

3.1 The surface inspection problem 12

3.1.1 Goals of surface inspection 12

3.1.2 Problems of surface inspection . . . 13
3.2 A proposed solution . . . 13
3.3 Content-based image retrieval . . . 14
3.3.1 Applications and uses . . . 15
3.3.2 Some existing CBIR systems . . . 15
3.4 PicSOM . . . 16
3.5 Content-based retrieval results with PicSOM . . . 18
3.5.1 Data sets . . . 18
3.5.2 Used features . . . 19
3.5.3 Adapting to user goals . . . 20
3.5.4 Feature pruning . . . 20
3.5.5 Retrieval efficiency . . . 23

4 Clustering **25**
4.1 Validating clustering results . . . 26
4.2 Clustering is an ill-posed problem . . . 28
4.3 Hierarchical clustering . . . 29

5 Managing large high dimensional data sets **31**
5.1 Motivation . . . 31
5.2 Simple nearest neighbor search . . . 32
5.3 Data space partitioning . . . 32
5.4 A look into classical indexing methods . . . 34
5.5 Problems with large data sets . . . 37

6 Data analysis algorithms based on and inspired by the SOM **39**
6.1 SOM variants . . . 40
6.2 Tree-shaped systems . . . 41
6.3 Case studies . . . 44
6.3.1 Method comparisons . . . 44
6.3.2 Applications to content-based retrieval . . . 44

7 The Evolving Tree **47**

7.1 Algorithm description 47

7.1.1 Basic operations and formulas 48

7.1.2 Inhibiting growth 49

7.1.3 Optimizing leaf node locations 50

7.2 Computational complexity 50

7.3 Benefits and disadvantages 52

7.4 Applications 53

7.4.1 Clustering 53

7.4.2 Data topology estimation 53

7.4.3 Data reduction 53

7.4.4 Density estimation 53

7.4.5 Approximate indexing 54

7.5 Adaptation to data 54

7.6 Software package 54

8 Conclusions **57**

References **59**

Publications of the thesis

I Jukka Iivarinen, Jussi Pakkanen and Juhani Rauhamaa, Content-Based Image Retrieval in Surface Inspection. In *Proceedings of 7th International Conference on Control, Automation, Robotics, and Vision*, pages 24–28, Singapore, December 3–6 2002.

II Jussi Pakkanen, Antti Ilvesmäki and Jukka Iivarinen, Defect Image Classification and Retrieval with MPEG-7 Descriptors. In J. Bigun, T. Gustafsson (Eds) *Proceedings of the 13th Scandinavian Conference on Image Analysis*, LNCS 2749, pages 349–355, Göteborg, Sweden, June 29 – July 2 2003.

III Jussi Pakkanen, The Evolving Tree, a New Kind of Self-Organizing Neural Network. In *Proceedings of the Workshop on Self-Organizing Maps '03*, pages 311–316, Kitakyushu, Japan, September 11–14 2003.

IV Jussi Pakkanen and Jukka Iivarinen, A Novel Self-Organizing Neural Network for Defect Image Classification. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2553–2558, Budapest, Hungary, July 25–29 2004.

V Jukka Iivarinen, Rami Rautkorpi, Jussi Pakkanen, and Juhani Rauhamaa, Content-Based Retrieval of Surface Defect Images with PicSOM. *International Journal of Fuzzy Systems*, vol 6, no 3, pages 160–167, September 2004.

VI Jussi Pakkanen, Jukka Iivarinen, and Erkki Oja, The Evolving Tree — a Novel Self-Organizing Network for Data Analysis. *Neural Processing Letters*, vol. 20, no. 3, pages 160–167, November 2004.

VII Jussi Pakkanen, Jukka Iivarinen and Erkki Oja, The Evolving Tree — Analysis and Applications. *IEEE Transactions on Neural Networks*, vol 17, number 3, pages 591–603, May 2006.

List of abbreviations

BMU	Best-matching unit
CBIR	Content-based image retrieval
DBI	Davies–Bouldin index
ETree	Evolving Tree
GCS	Growing Cell Structures
GNU GPL	GNU General Public License
kNN	k nearest neighbors
MAM	Metric access method
MPEG	Motion Picture Experts Group
PCA	Principal component analysis
SAM	Spatial access method
SOM	Self-organizing map
SVM	Support vector machine
TS-SOM	Tree-structured self-organizing map
QA	Quality assurance

Chapter 1

Introduction

It can be said that the biggest moment in the evolution of modern man was the invention of tools. The first tools were simple bone hammers, spears and such, but they gave man a notable advantage against predators and other men. It was quite soon discovered that the quality of tools was directly linked to the probability of surviving to the next day. Breaking a sword in the middle of a struggle most probably meant an immediate and messy death.

This lead to continuous improvements in technology and manufacturing in tools, weapons and other similar items. As society developed more and more products and product categories were born. Very few of these products were weapon related. Thus a minor defect in the product was no longer followed by an evisceration of its user. The downside was that these products faced a different kind of enemy: money. Since the manufacturers made a living through commerce rather than warfare they needed to create products other people were willing to pay for.

Prior to the industrial revolution this sort of quality assurance was relatively simple to do. Each product was handcrafted and inspected by a professional. When manufacturing was transferred to machines this became more difficult, since one machine could easily produce more than any one person can inspect. The advances in technology have increased this gap. For example a modern paper machine produces a paper sheet several meters wide with speeds exceeding 30 m/s. The extraordinary magnitude of paper produced is simply too much to be inspected by hand.

Fortunately advances in other branches of science have given us tools to solve this problem. Specifically it has given us fast digital photography, computer vision and classification methods. Cameras allow us to take pictures of the product line in real time. Computer vision makes it possible to detect potential quality errors. These can then be divided into different classes based on severity, and suitable corrective measures can be taken.

The problem that a quality assurance expert tries to solve can be formulated quite simply: how can we efficiently separate the serious defects from the others. The classical method is to build a classifier. Unfortunately most methods don't

scale very well to tens of thousands of data points. Another drawback of a fully automated classification system is that bringing human knowledge in the loop can be difficult. In some difficult cases human expertise may be essential in making decisions.

Our research has tried to solve this difficult quality assurance (QA) problem using content-based image retrieval (CBIR). In CBIR the user wants to find a certain kind of image from a possibly very large database. The system then tries to find the target image using only information extracted from the images themselves, without any prior annotations. CBIR seems like a suitable tool for our problem.

To see if this is the case we examine how the PicSOM CBIR tool handles large databases of surface defect images. We also present and analyze a new data structure called *The Evolving Tree*. It is a tree shaped neural network designed to work with very large scale problems. Our results show that these approaches can be succesfully applied to large defect image databases.

1.1 Contributions of the thesis

The main scientific contributions of this thesis can be summarized as follows.

- A survey of those unsupervised learning methods, variants of self-organising map and tree shaped neural networks that can be utilized for indexing large databases.
- Finding suitable features and establishing a ground truth for content based image retrieval (CBIR) for real world paper and metal defect images.
- Introduction of the *Evolving Tree* (ETree) neural network.
- Evaluating the performance and feasibility of ETree using a variety of different test methodologies and data sets.
- Applying ETree to the surface inspection problem using the features and methodologies examined earlier.

1.2 Outline of the thesis

This thesis consists of an introductory part, together with seven separate publications. The contents of the introductory part is as follows.

In chapter 2 the context in which the other portions should be seen is presented. In chapter 3 the surface inspection problem and the content-based retrieval approach to solving it is examined. Experimental results showing the system's performance are also presented. This rounds up the practical results of the thesis. Then the more theoretical portion follows. First some background information is discussed starting with chapter 4, which examines clustering, mostly in vector

spaces. Then in chapter 5 the difficulties of high-dimensional data analysis are presented. Chapter 6 contains a literary survey of related self-organizing network methods. Chapter 7 presents the Evolving Tree neural network, which is the main theoretical contribution of this thesis. Finally chapter 8 rounds up the achieved results and concludes the thesis.

1.3 Author's contributions to publications

Publication I outlines the PicSOM CBIR system, the research problem, and how we propose to solve the problem. Experiments show that our approach is feasible. The author performed all of the implementation tasks and experiments.

Publication II presents the MPEG-7 image content descriptors and examines their suitability to this particular problem. Experiments show that a subset of three features performs almost as well as using all six of them. This reduces the complexity of our problem. The author designed most of the experiments and also analyzed the results.

Publication III presents for the first time the Evolving Tree (ETree) self-organizing network. The motivation, architecture, and training formulas are thoroughly presented. Preliminary results show that ETree is suitable for CBIR tasks. The author personally developed the ETree network, created all the required programs, ran the experiments and wrote the paper.

Publication IV further examines ETree's suitability to the CBIR problem. Experiments with MPEG-7 datasets show that the method outperforms classical SOM methods. The author performed and analyzed the experiments and wrote most of the text.

Publication V further examines PicSOM's performance by analyzing a different database and also adding comparison results to a standard k nearest neighbor classifier. PicSOM is found to perform quite well. The author was mostly responsible for analyzing the results and writing the manuscript.

Publication VI analyzes the ETree theoretically and compares it against the TS-SOM, which is used in PicSOM. ETree performs better than TS-SOM in these experiments. The author was responsible for planning and running the experiments and also wrote most of the text.

Publication VII thoroughly analyzes ETree by proposing several additions and enhancements. It is discovered that most of the proposed enhancements are not beneficial, and some in fact decrease performance. Thus we can conclude that the basic ETree algorithm is the preferred one, since it is the most simple. ETree is also compared to several classical and tree-shaped systems. The author developed most of the experiments, performed roughly half of them and wrote most of the text in the article.

Chapter 2

Visual quality assurance for surface defect images

While our work mostly examines content-based image retrieval and the role of self-organizing networks in this problem, it is important to see these methods in perspective. In this case the end goal is to *create more products*, or to *create higher quality products*. These products can be anything but we examine mass produced flat sheet-like items, specifically paper and metal webs.

In this chapter we describe the background of the problem. We start by taking a candidate picture of a defect and follow it to the point where the potential defect gets classified. It should be noted that while we describe the process from a CBIR point of view, the same general approach is used in various other fields such as machine vision (Sonka et al., 1999). For further information the reader is referred to (Brzakovic and Vujovic, 1996) for a discussion on designing a defect classification system or (Graf et al., 1995) for a description of a defect detection system for nonwoven.

2.1 Phases of content-based visual quality assurance

There are many different ways of doing visual quality inspection. Most of them follow the outline shown in Figure 2.1. We will now briefly examine all those phases separately.

2.1.1 Taking the image

The images used in CBIR can come from a variety of sources. The most common ones are photos taken with digital cameras or documents digitized with regular tabletop scanners. There are enormous variations in technology even within these

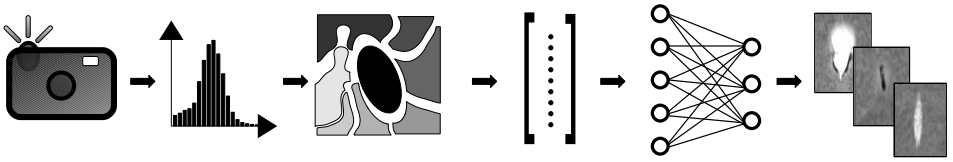

Figure 2.1: Possible phases of visual quality inspection: taking the image, preprocessing, segmentation, feature extraction, training, queries.

elementary methods. The digitization is usually done with a CCD cell, but other imaging elements can also be used, depending on the application.

While most images are taken using visible light, there are several applications where other imaging methods are preferable. The most common example is infrared imaging, which can be used to "see" the temperatures of objects. Applications include finding people lost in forests at night, night vision goggles, weather satellites and various military applications. Other parts of the electromagnetic spectrum are also widely used. Examples include UV light, which is widely used in astronomy, X-rays for medical imaging and gamma rays for inspecting shipping containers.

Imaging is not limited to electromagnetic radiation. Almost any natural phenomenon that produces measurable effects can be used to create images. Probably the most familiar example is sonogram imaging which is used to noninvasively examine babies of pregnant mothers. The imaging is based on reflections of ultrasonic waves. A similar system is the sonar, which can be used to detect the surroundings of a submarine using regular sound waves. Sonar's operating principle is principle is almost equivalent to a radar. The difference is that radar uses radio waves.

If we limit ourselves to imaging as used in paper manufacturing process we find many different imaging methods have been used. Our images have been taken with grayscale line cameras. However, others have used methods such as X-rays (Delgado and Gomes, 1995) and even scanning electron microscopy (Mott et al., 1995).

Taking high quality digital images is both a science and an art. Scientific problems include lighting, optics, radio wave interferometry, noise minimization and so on. For a more detailed description how our imaging system approaches these problems see (Iivarinen et al., 2000).

2.1.2 Preprocessing

Digitized images are very rarely perfect. They may be under- or overexposed or be out of focus. They may also contain all sorts of noise or artifacts, or suffer from various other degradations. Preprocessing aims at reducing these errors while emphasizing the relevant portions of the image. We briefly describe here some commonly used methods. Those interested in details are encouraged to

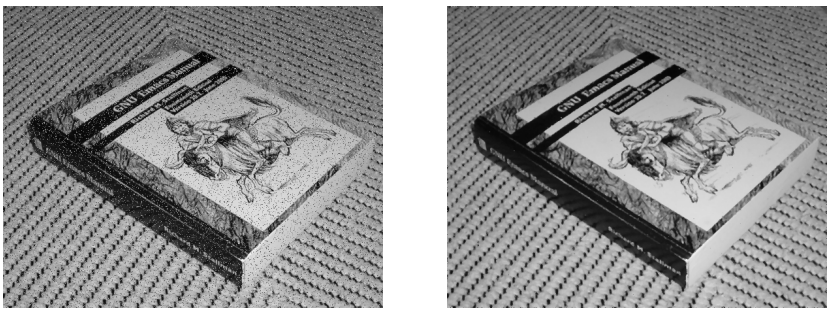

Figure 2.2: An example of image preprocessing: removing salt-and-pepper noise with median filtering. As a trade-off the image is slightly softened and small details are lost.

read any of the many textbooks on the subject, such as (Gonzalez and Woods, 1992; Castleman, 1995).

Basic methods include rotations, scalings, translations, histogram equalization, contrast adjustment and other methods that can even be found in common digital image processing applications such as *The Gimp* and *Photoshop*. An illustrating example can be found in (Bresee and Paniluk, 1997), where histogram modifications make the fine structure of nonwoven fabric clearly visible.

Color space conversion changes the underlying representation of the image, for example from RGB to CMYK or HSV. False coloring of grayscale images can also be seen as a colorspace conversion operation. The reason for the conversion is that certain phenomena are more prominent under different color systems (Jain, 1989).

Filtering is a wide-reaching term for all sorts of interesting image manipulation operations. Examples include low-pass filtering, which reduces noise, Gabor filtering (Dunn et al., 1994), median filtering (Tyan, 1981) which reduces salt-and-pepper noise and so on. See Figure 2.2 for an example.

Edge detection emphasizes the edges in the image. This makes it easier to detect the boundaries between objects. Edge detection is usually done by estimating and thresholding the gradient with a suitable operator. Well known examples include the Canny edge detector (Canny, 1986) and the Marr–Hildreth edge detector (Marr and Hildreth, 1980). It should be noted that many of these operations are calculated using filtering methods mentioned above.

In a nutshell the purpose of preprocessing is to enhance the desired structure in the image. Preprocessing methods are not discussed in depth in this thesis, since our images have been preprocessed by the imaging system.

2.1.3 Segmentation

Almost all digital images are rectangular. In contrast almost all image targets have wildly varying shapes. This means that even perfectly aligned images contain pixels that don't belong to the target. These are often called *background pixels*. These usually have different statistical properties than the target pixels. Since we want to only examine the properties of the target, we would like to separate the background pixels and discard them. This process is called image segmentation.

The image to be examined may also contain several target areas. Segmentation is also used in separating different targets to their own separate regions. This allows the different targets to be analyzed independently of each other. Interrelations between the different portions can also yield interesting information (Haralick and Shapiro, 1992). For example we can use segmentation to locate blood vessels in medical images. A doctor can determine the healthiness of a tissue based on the locations and sizes of the vessels.

Segmentation using only image analysis techniques is rather difficult, so several methods have been developed to help the segmentation process. As an example (Gustafson and Delgado, 1996) describes a chemical segmentation process. The basic idea is to develop dyes that chemically bind only to areas of interest. This makes the segments more visible and thus easier to analyze.

In this thesis we do not examine segmentation methods in depth. The reason is that we use segmentation masks that have been automatically extracted by the defect detection system. Each mask image shows which pixels form one or more defects in the image. Segmenting these defects is an interesting and difficult problem, but in this thesis we will not examine it any further due to issues of scope.

2.1.4 Feature extraction

Once we have separated different items in the image, we need to describe them somehow, so we can organize and query them. In the optimal case we would get high quality textual representation, such as: "a chair, ball shaped, designed by Eero Aarnio, on the set of *The Prisoner*". Obtaining this level of description automatically is not possible given current technology. Therefore we must utilize lower level descriptors that give us statistical properties of the image and its different regions (Sonka et al., 1999).

There are literally thousands of different features that can be calculated. They can be divided into tens of different groups (Gudivada and Raghavan, 1995). Given our problem setting we have focused on the following feature types.

Color features measure the amount and distribution of colors in the image. The term is usually used even for single-channel, or grayscale, images. The rationale for these features is that images of similar objects usually have similar colors. Color features include the color histrogram (Swain and Bal-

lard, 1991), color regions (Hsu et al., 1995) and color moments (Stricker and Orengo, 1995). In quality control the color of the product is very important, so deviations from the target color are considered errors.

Texture features try to characterize visual properties of the surface, such as coarseness, orientation, entropy, energy and so on. Popular methods include autoregression (Mital and Leng, 1994), co-occurrence matrices (Haralick et al., 1973) and various other statistical measures (Mäenpää, 2003; Chaudhuri and Sarkar, 1995). Textural properties are very sensible to scaling, which should be taken into account when calculating these features.

Shape features characterize the shapes of the objects found during the segmentation phase. The basic idea is the same as with texture features: calculating statistical and other measures usually using the object contour. These features measure various object properties such as convexity, elongatedness, thickness, amount of holes in the object and so on (Marshall, 1989).

For applications in paper manufacturing see for example (Bernié and Douglas, 1996) which analyzes features used to determine paper formation or (I'Anson, 1995) where a Fourier transform is used to discover repetitive markings in paper.

In our work we have used general features from the MPEG-7 standard (Manjunath et al., 2002) and complemented those with problem specific shape descriptors (Iivarinen et al., 1997). Deeper research on features for this particular problem can be found in (Kunttu, 2005).

2.1.5 Training a CBIR system

Once we have extracted data from the images, we can use that to train a retrieval system. While the steps above have been common with regular computer vision problems, the operations from here on are unique to content-based image retrieval. There are literally hundreds of different algorithms to choose from. Some of these are presented and the theory behind them examined in chapters 5 and 6.

We focus on neural computation methods. There are two main reasons for this. The first one is that in the last couple of decades neural networks have been shown to be very effective tools in various fields and applications. Neural networks have also been successfully applied to other parts of the paper making process (see for example (Edwards et al., 1999; Milosavjlevic and Heikkilä, 1999)). The other reason is that our research is a direct continuation of the work done in the Laboratory of Computer and Information Science for several years. The methods we have used are the tree-structured self-organizing map (section 6.2) and the Evolving Tree (chapter 7), which has been created by us for this particular problem.

2.1.6 Queries

The final step in CBIR is using the system to find relevant images. This is a two-phased problem. First the system has to determine what kinds of images the user is searching for. Then it has to find those images from the database. Following (Lew, 2001), we can list three basic query approaches.

Query by text where the user types some search terms. This method requires keyword annotation on the images and thus does not scale to very large data sets.

Query by user sample where the user has an image, and wants to find other similar images. The system analyzes the image by doing all the same preprocessing, segmentation and feature extraction steps as were used for the images in the database. The resulting feature values are then used as the query parameters.

Query by pictorial example where the system shows a set of images in its database to the user and then the user selects those that look similar to what he is searching for.

Since CBIR databases are huge, it is extremely rare that the system immediately finds the correct image or images. Therefore a typical search takes several rounds, where the search criterion is refined. This is called *relevance feedback* and is usually done by letting the user select which images look similar to his query target and which do not. These selections can then, for example, be converted into weights that show the relative relevance of used features.

Our CBIR query experiments have mainly been done with the PicSOM system (section 3.4) which belongs to the last category.

2.2 A word about difficulty

All the mentioned subproblems of CBIR are interesting research areas in their own right. Some of them, like those related to digital image processing and computer vision, have been extensively researched for decades. Even though there are very good generally accepted methods, none of these can be called a wholly solved problem.

Arguably the most difficult subproblem is image segmentation. In the general case the problem is "AI-complete". This means that it is hypothesised to be equivalent to creating an artificial intelligence comparable to a human. The reasoning is that to truly separate the different objects in the image, the segmenter must *understand* what real world items it contains. Edges and textures simply do not convey enough information. However in limited application domains, such as ours, segmentation becomes a much more feasible problem.

Chapter 3

Using content-based image retrieval for surface inspection

There is a common rule of thumb for all manufacturing processes: any product failing quality assurance (QA) requirements is a cost. A corollary to this is that the further along the production chain the defective item gets, the more it costs. Depending on the details of the manufacturing process the costs can increase either slowly or exponentially. There is a big incentive, then, to eliminate as many of these defective products as early as possible (Roberts, 1983; Davies, 1990).

This problem is especially prevalent when producing items where appearance and esthetics are important. These include clothing, cars, visible construction materials, paper, and handheld electronic devices. In these cases a visual defect that makes it to the end user may cause him to switch to a different supplier. In these fields an effective QA methodology is not only money saver, it is an absolute necessity.

One way of doing surface inspection is to utilize content based image retrieval (CBIR). In our problem context CBIR can be understood as a process of finding images that look similar to a given query image using only the information present in the images. That is, we do not use any a priori class information in the actual query process. One of the advantages of CBIR is that queries are guided by a human being. This makes it easy to bring the human expert into the decision process. This is difficult to do with classical classifier-based methods.

Figure 3.1: A paper web inspection machine in operation. Image courtesy of ABB Oy.

3.1 The surface inspection problem

Traditionally visual quality inspection has been done by people. A requirement is that the flow of products must be relatively slow. For example on a steel mill the finished steel sheets move at moderate speeds of approximately 1 m/s. A human inspector can look for artifacts quite effectively. In contrast the paper coming out of a paper mill moves at speeds over 30 m/s. No human being can inspect that by themselves.

As computing power and imaging systems developed, more and more inspections could be done automatically. This was culminated in the mid to late 90s when cameras became fast enough to take pictures of even the fastest product lines. An example can be seen in Figure 3.1, which shows a paper web inspection machine in use in a paper mill. The bottom beam contains high-powered lights and the top beam houses high resolution cameras. These kinds of efficient surface inspection machines allow us to better understand and estimate the behaviour of the manufacturing process.

3.1.1 Goals of surface inspection

The benefits from surface inspection, and QA in general, can be roughly split into three different cases. There are naturally several other ways of grouping inspection tasks, see for example (Newman and Jain, 1995).

Defect detection is the most obvious application. If a factory is supposed to produce red balls and is instead putting out blue cubes, there is usually something wrong in the system. Similarly we can detect holes in metal sheets that should be intact, bent rods, shattered bottles and so on.

Imperfection detection detects smaller errors, such as variations in hue, minor scratches, water spots on paper and so on. Depending on the quality

requirements the definition of "minor imperfection" can vary quite a lot. In extreme cases, such as aerospace engineering, even defects that are invisible to the naked eye can cause a part to be immediately discarded. Usually small deviations are tolerated and in some cases the products are divided into several quality groups and sold at a different price.

Gathering state information measures the manufacturing process rather than the actual products. For example if we notice that the amount of defects suddenly doubles or triples, there is probably something wrong in the manufacturing process. In addition to process error diagnosis, the measurement data can also be used to prevent errors from happening. If it is known from experience that major production malfunctions are usually preceded by certain kinds of imperfections, preventive measures can be taken whenever those defects are detected.

3.1.2 Problems of surface inspection

There are several different problems in surface inspection. Firstly the product must be passed by the imaging system. Depending on the product line the difficulty varies from simple to extremely hard. Lighting and imaging also present their own difficulties, especially if the end product is not flat, such as corrugated iron. Most factories have somewhat hostile surroundings: high temperatures, dust and dirt particles and so on. These cause image degradation in optics, lighting and so on.

While these are interesting and challenging problems, we focus our attention to the things that happen after the images have been taken. That is either classification or content-based retrieval of defect images. The problems facing those parts can be found in section 2.1.

3.2 A proposed solution

The classical approach to these kinds of problems has been to utilize an automatic classifier. Those images that are deemed to belong to a serious defect class are marked and removed during later stages of the manufacturing process. This is a simple and working solution. There are also some intrinsic problems with this method.

One drawback is that keeping the classifier up to date can take a considerable amount of effort. If we consider a paper mill, we find that each paper making machine is unique. They all produce slightly different defects. The defect classes also vary depending on what kind of wood is used, what processing chemicals are used and so on. Trying to account for all these variations using a single model is extremely difficult. To obtain best possible results the classifiers must be hand tuned for each process line separately.

Even this is not sufficient, since defect types vary with time. There are several environmental and other reasons for this. Factors such as outside temperature

and humidity affect the raw material. Machines wear down in use and parts get replaced. The classifiers must thus be updated periodically to correct these biases.

Perhaps the biggest obstacle, however, is that with an automatic classifier it is hard to bring a human expert into the decision making loop. An experienced human has a better understanding of the whole system than a classifier trained with a bunch of prototype vectors.

We propose that content-based image retrieval (CBIR) can be applied to the visual quality inspection problem. CBIR can be roughly described as the discipline of finding desired images from a vast database using only pictorial information. That is, there are no labels or other such information. CBIR systems have been designed to be adaptive and to be used by humans. This makes them overcome the difficulties mentioned above, since traditional methods have not been very efficient in bringing a human being into the loop. Given the problem setting we want to achieve two things. Firstly we want to find serious defects. Secondly we want to give the machine operator a good view of the overall state of the system.

As an example use case let us assume that an operator has some images of defects he considers serious. Let us further assume that we train a CBIR system for all defect images discovered during one day. The operator can then query the system using the defect images as examples. The system returns those images it deems similar to the query images. The operator is thus given an overview of the types and amounts of defects. He can combine this information with his knowledge of the paper machine to refine the process, allowing him to eliminate defects in the future.

3.3 Content-based image retrieval

CBIR is a term used to describe various methods and systems for searching for desired kinds of images in very large image databases. The first steps towards this were done in the mid 80s. These approaches were based on manually annotating the images with keywords and then using text query methods. Unfortunately this approach simply does not scale up (see section 5.5).

Since the early 90s the exponential growth in computer speed made it possible to examine other kinds of approaches. These eventually gave birth to modern CBIR systems. What separates CBIR systems from classical classification schemes is that they combine image processing, classification and especially user feedback into one coherent package and that they perform efficiently at very large databases. Bringing human experts into the loop is beneficial, since they have intuitive knowledge which is difficult to utilize with other approaches. A CBIR system should handle 10 000 images effortlessly, preferably it should scale to hundreds of thousands, even millions of images without slowdowns (Del Bimbo, 1999).

3.3.1 Applications and uses

In a slightly broader sense CBIR can be seen as a method of assigning data-derived statistical semantics to images. Thus CBIR can be utilized in finding desired objects from large unorganized image collections. Thus CBIR has as many applications as there are search interests (Fleck et al., 1996). We illustrate real world uses of CBIR with a couple of examples.

(Shyu et al., 1999) describe a system designed for medical imaging, specifically tomographic images of lungs. The task is to reliably detect diseases such as emphysema. The basic idea is to have a qualified physician mark fissures and similar points of interest on images. These are used as a basis for queries. Their query system uses a decision tree combined with a hash table. This speeds up the queries noticeably with only a slight negative effect on accuracy.

Astronomical images are difficult to analyze. The data sets are usually very large and image quality can vary enormously. (Csillaghy et al., 2000) presents a CBIR system that uses textural features and a self-organizing map (SOM, see chapter 6) for the indexing method. They find that this approach is feasible, but training the SOMs turns out to be computationally expensive.

A web-based method of browsing geographic data is presented in (Smith, 1996). It allows users to browse aerial photos of Earth. This is quite similar to the popular Google maps service[1]. It also has a texture based search facility. This allows the user to find, for example, housing projects or other areas of interest.

The ARTISTE project (Addis et al., 2003) tries to build a query engine that can find pieces of art from several distributed databases. The system holds over 160 000 images and 5 million metadata items. The system supports several query methods, such as query by color and subimages, shape of frame and even UV light reflectance (variations in UV reflectance indicate restoration work). Queries can also be done based on textual features such as the artist's name.

3.3.2 Some existing CBIR systems

One of the first true CBIR systems was *QBIC* (Flickner et al., 1995) which was developed by IBM in the early 90s. After that the number of CBIR papers and systems has grown dramatically. We now briefly present some interesting systems to illustrate different approaches to CBIR.

The basic idea of *Photobook* (Pentland et al., 1994) is that there should be a set of semantic features that describe an image, similarly to how letters form words and sentences. In their experiments they use textural properties and "eigenitems" calculated using the Karhunen-Lòeve transform.

PicHunter (Cox et al., 2000) enhances the CBIR process by applying a strict bayesian framework to the problem. It tries to model the query target probabistically. It then uses this model to return the best possible images in the

[1] `http://maps.google.com/`

database.

Those interested in the implementation details of a CBIR system should examine *VIPER* (Squire et al., 1999), whose entire source code is available under the GNU GPL (Free Software Foundation, 1991). The released version is called *GIFT*, or Gnu Image-Finding Tool, and can be downloaded from `http://www.gnu.org/software/gift/gift.html`.

There are dozens of other CBIR systems, such as *Chabot* (Ogle and Stonebraker, 1995), *MARS* (Huang et al., 1996), *Virage* (Bach et al., 1996) and *VisualSEEk* (Smith and Chang, 1996). For further information and discussion we refer interested readers to e.g. (Johansson, 2000; Koskela, 2003) or (Kiranyaz, 2005), which explores the algorithmic and implementation issues.

3.4 PicSOM

In our work we have used the *PicSOM* CBIR tool (Koskela, 2003; Laaksonen et al., 2000; Laaksonen et al., 1999). PicSOM was developed in the Laboratory of Computer and Information Science as a platform for researching content based image retrieval. It has since evolved to search not only images but also audio, text, and video (Koskela et al., 2005). Because PicSOM is not limited to still images it is more properly called a content based information retrieval system.

PicSOM's architecture is based on the tree-structured SOM, or TS-SOM (see section 6.2). One TS-SOM is trained for each of several visual features. PicSOM gains its power by efficiently combining several TS-SOMs into a single result. To understand how the system works, we first examine the system's user interface, which can be seen in Figure 3.2.

At the top we can see the bottom layers of the three TS-SOMs. In this case we have three different feature sets. Below them are three images that the user decided to be representative of the target image he is searching for. In this case the query target seems to be a white rectangular object at an angle. Finally at the bottom are the images that PicSOM chooses as closest to what the user wants to find. The returned images are very similar to the query images, so we can probably call this particular query a success.

At the beginning of the query process PicSOM shows the user a random sampling of images in the database. The user then selects those images that look like his target image. PicSOM then places these on the TS-SOM surfaces. The selected images should contain similar items and thus should have similar feature vectors. PicSOM's task is to find images that are as similar as possible to the selected images but different from the discarded images.

TS-SOM sets a topological ordering to the map nodes. Thus we can find similar images in neighboring nodes. PicSOM takes advantage of this by "spreading out" the user's responses on the map. For every selected image a small positive value is placed on the map surface. Similarly each discarded image obtains a small negative value. The 2D map grid is then low pass filtered. If the selected images

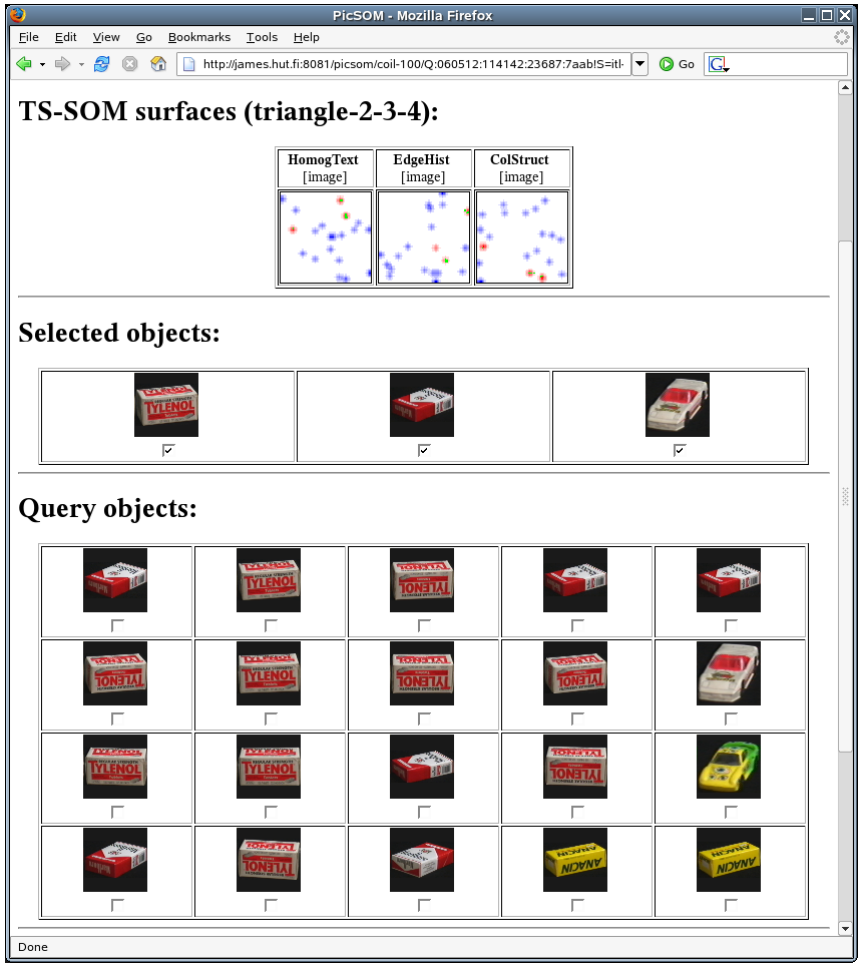

Figure 3.2: The PicSOM interface during a sample query of various objects.

are close to each other, as we suspect, this process creates dense positive regions. The negative images should spread out evenly to the rest of the map.

PicSOM now has sufficient information to select new images from the database. It selects those images which are mapped to the positive nodes of the map and are as far away as possible from the negative areas. PicSOM can also combine the results from several different TS-SOMs simply by selecting those images with best overall performance.

The selected images are then shown to the user, who again selects those that are similar to what he is searching for. This is repeated until the user finds the correct image or wants to stop searching. This process of adapting to user's query requirements is called *relevance feedback*.

Unfortunately doing large amounts of queries manually is very slow and somewhat

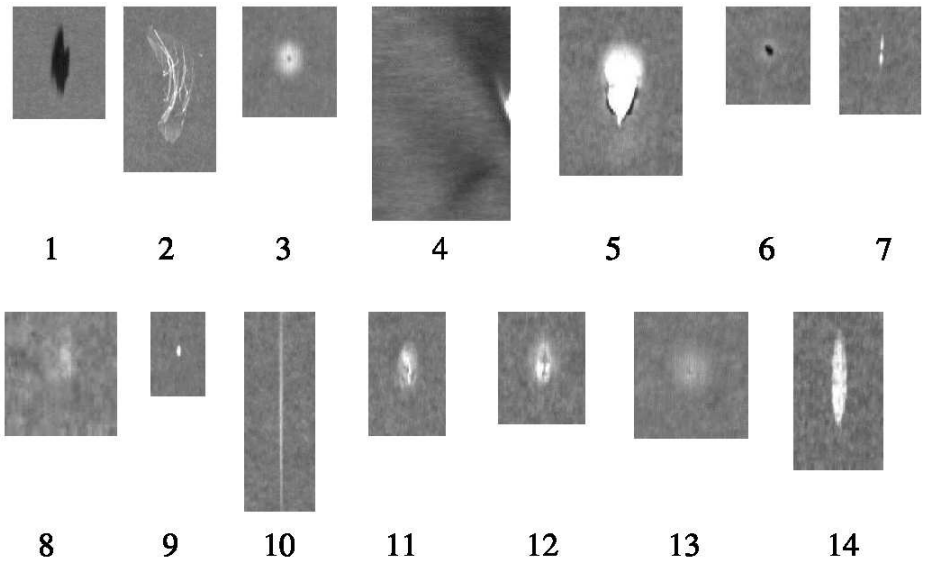

Figure 3.3: Examples of paper defect images. Note how several of the classes are very similar, especially classes 11 and 12.

unreliable due to human errors. Therefore PicSOM can do automatic queries based on class information. This allows us to easily test retrieval efficiency under various circumstances.

3.5 Content-based retrieval results with PicSOM

Thus far we have mostly talked about theory and background. In this chapter we look at the actual CBIR results obtained with PicSOM. First we describe the data and testing methodology used. Then we describe our main results. More detailed results can be found in Publications II and V and in (Pakkanen and Iivarinen, 2003).

3.5.1 Data sets

The main data used in our experiments has approximately 1300 surface defect images that were obtained from a real, online paper web inspection system. There are several different kinds of defects, such as light spots, dark streaks, wrinkles, holes, oil stains, and so on. The database have been pre-classified into 14 different classes. All the images were gray-scale with 256 colors. The size of the images varies considerably. Some are only 200 by 200 pixels in size while others are several thousand pixels high. Examples of the images can be seen in Figure 3.3. Each class has roughly 100 images except class 11, which has 30 and class 12, which has 70 images.

The additional database contains 2004 defect images from an online metal web inspection system. The database is preclassified into 14 different classes, with each class containing from 101 up to 165 images. All images are gray-scale with 256 gray levels, and their dimensions range from less than 100 pixels up to over 1000 pixels. Each image was supplied with a segmentation mask, indicating the defect areas. These masks were computed automatically. All images and the segmentation masks were provided by our industrial partner ABB Oy. For further information on this data set see Publication V.

3.5.2 Used features

We have utilized mainly two different kinds of features. The first ones com from the MPEG-7 standard, ISO/IEC 15938, formally named "Multimedia Content Description Interface" (Manjunath et al., 2001; Manjunath et al., 2002). This standard defines standardized descriptions of streamed or stored images or video, to be used in searching, identifying, filtering and browsing images or video in various applications. The standard defines several still image descriptors, which we have used in our experiments.

Color Layout (CL) specifies a spatial distribution of colors. The image is divided into 8×8 blocks and the dominant colors are solved for each block in the YCbCr color system. Discrete Cosine Transform is applied to the dominant colors in each channel and the DCT coefficients are used as a descriptor.

Color Structure (CS) slides a structuring element over the image. The numbers of positions where the element contains each particular color are stored and used as a descriptor.

Scalable Color (SC) is a 256-color histogram in HSV color space, which is encoded by a Haar transform.

Edge Histogram (EH) calculates the amount of vertical, horizontal, 45 degree, 135 degree and non-directional edges in 16 sub-images of the picture, resulting in a total of 80 histogram bins.

Homogeneous Texture (HT) filters the image with a bank of orientation and scale tuned filters that are modeled using Gabor functions. The first and second moments of the energy in the frequency domain in the corresponding sub-bands are then used as the components of the texture descriptor.

Region-based Shape (RS) utilizes a set of 35 Angular Radial Transform (ART) coefficients that are calculated within a disk centered at the center of the image's Y channel.

The descriptors were calculated using the MPEG-7 eXperimentation Model (XM) software versions.

In addition to these we use a combination of simple shape descriptors (SSD), which were developed for surface defect description in our earlier project (Iivarinen and Visa, 1998). These features are calculated from an object's contour. The descriptors are convexity, principal axis ratio, compactness, circular variance, elliptic variance, and angle. The descriptors are not very efficient individually, but the combination of them has been shown to produce good results with low computational costs (Iivarinen et al., 1997).

3.5.3 Adapting to user goals

One of the advantages of CBIR we have discussed has been the easy way to bring a human expert into the loop. This means that the user must have a simple interface for directing the queries. As was discussed earlier, PicSOM achieves this through relevance feedback. Figures 3.4 and 3.5 shows two examples of query adaptation after a few iterations.

In the first case the user has searched for dark horizontal spots. The images returned by PicSOM are extremely similar to the selected query images. This is an especially good result when we consider that the only information PicSOM has available are the selections done by the user. There are no given classifications or annotations. PicSOM also combines the features efficiently. During training all the features are separate. We can see that the query results have similar color, shape, and texture as the query images. The weighting of different features has been automatically done by PicSOM.

The image in Figure 3.5 shows a different query using paper defects. This time the user is searching for round white spots. The user has found more example images than in the previous query and the results are a lot more homogeneous. Again we find that PicSOM can determine what the user was searching for and returns those kinds of images.

These similar images can then be examined and analysed by a human expert. Suppose we have some kind of metadata on the images used in training, such as defect severity or descriptions of defect origins, and a new defect image. When an expert queries the database for similar images he finds a bunch of relevant images. Examining the metadata of the returned images, he can assess the severity of the query image.

CBIR is beneficial even without additional metadata. When the human expert obtains a collection of similar defect images his task is simplified. He can use his experience and prior knowledge more efficiently when he is given some defect context by the CBIR system.

3.5.4 Feature pruning

Since MPEG-7 features are relatively new, there are not many experiments on their relative efficiencies. This is especially true if we focus only on surface inspection. Thus we had to determine which of the features perform best.

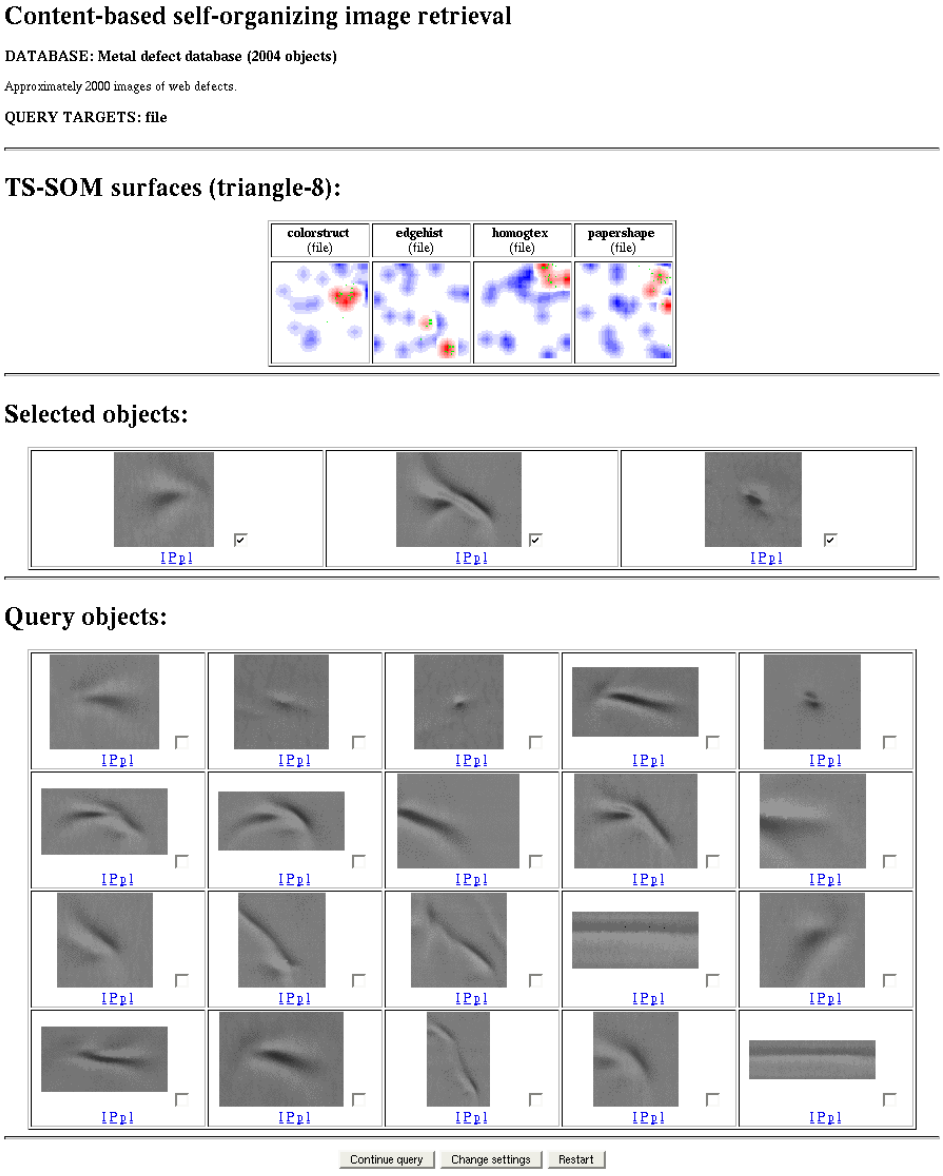

Figure 3.4: PicSOM adapting to user queries using the metal database.

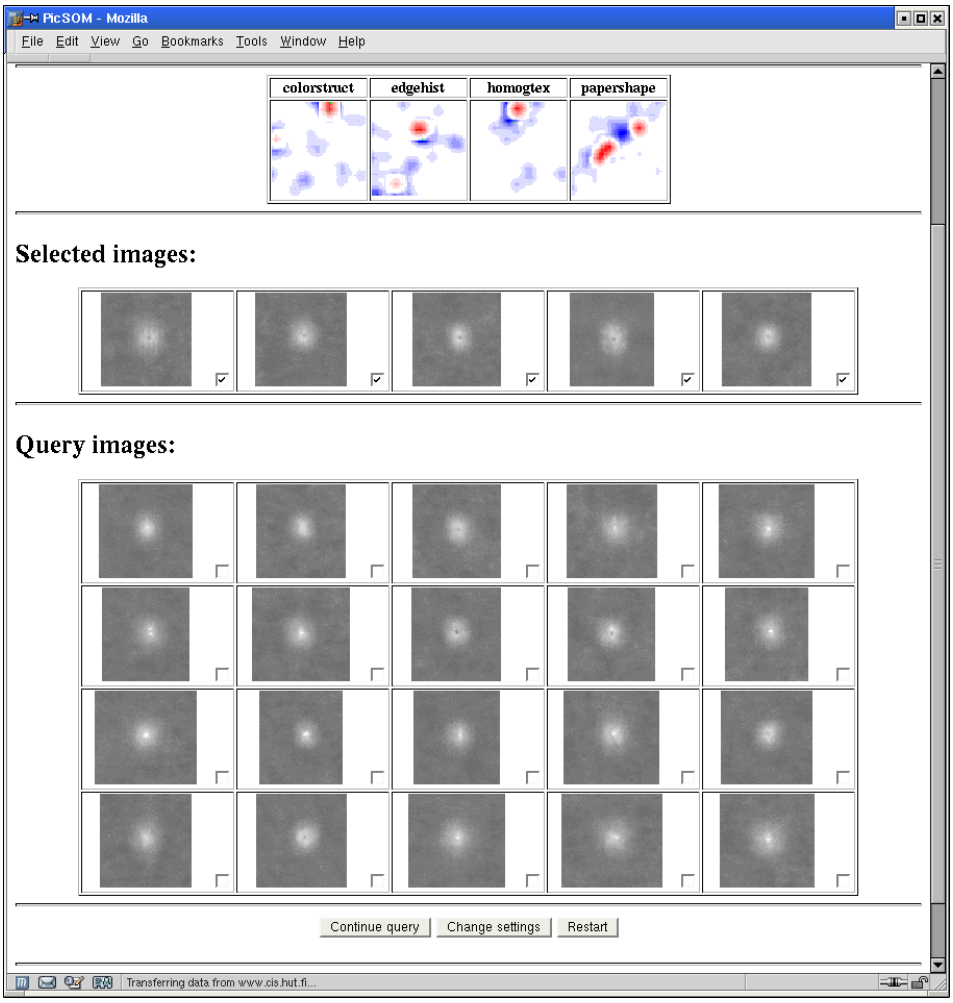

Figure 3.5: PicSOM adapting to user queries on the paper database.

MPEG-7 contains dozens of different feature descriptors. These can calculate various things such as camera movement. We have only examined those that can be utilized for our problem: color, texture and shape descriptors. Some of these, such as *contour-based shape* had to be dismissed since its feature vectors have varying length. SOM, and therefore also PicSOM, only works on feature vectors which have the same dimension.

Early on in our experiments (see Publication II) we discovered that three of the features were noticeably better than the others for our problem. These were color structure, edge histogram and homogeneous texture. Another important thing is that none of the shape features were particularly good. They all lost noticeably to our simple shape descriptors. Even using all features together yielded no significant improvements. Thus in further studies we only used these four features.

3.5.5 Retrieval efficiency

PicSOM contains a system for automatically testing query performance. This works by selecting all shown images from the desired class and deselecting all others. This is done for a specified amount of rounds. We measure two important features after each round: precision and recall. Precision tells which percentage of the images returned by PicSOM are "correct", that is, in the class we are querying. When precision is larger than the a priori probability of the class we know that the CBIR system is working properly. Recall lists the total amount of correct images returned. When recall reaches 1, all images in the class have been found. This takes a minimum of 5 rounds, since PicSOM returns 20 images per round and the classes have at most 100 images.

Figure 3.6 shows how PicSOM performs on the paper database using the three best features determined earlier. The first image has a precision/recall graph. The plots start from the left (recall 0) and progress to the right edge (recall 1). The performance is especially good with the easy class, precision remains over 70% until almost all images have been found. The earlier mentioned very difficult class 12 has noticeably worse classification percentage. The a priori probability of this class is only 0.02, whereas PicSOM's precision is ten times that at 0.2. In the case of the easy class precision increases on some iterations. This is very atypical, as usually precision decreases when the first "easy" cases have been found.

Another way of looking at the results can be seen in the second image in Figure 3.6. It has the recall plotted as a function of query iterations. A human being can be assumed to do about 10 rounds before quitting. At ten rounds PicSOM has found 80% of the correct images on average. Further results and discussion on these can be found in Publication II.

Experiments with metal defect database give similar results. Since metal classes have more images, it takes eight rounds to retrieve all of them. Again we find that after two times the optimal amount of rounds PicSOM has found 80% of all images. The fact that we can obtain these good results on two different databases is a very desirable result. For more details see Publication V.

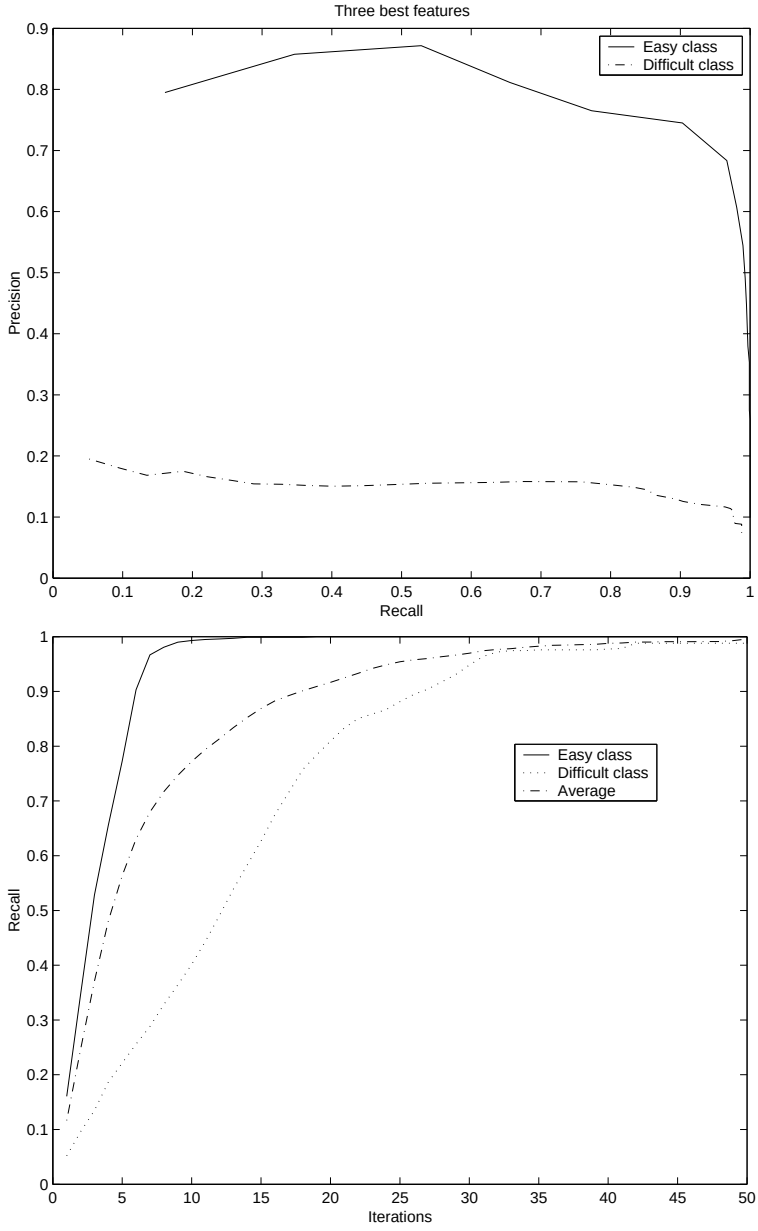

Figure 3.6: Precision/recall results and recall as a function of query rounds using the paper defect database.

Chapter 4

Clustering

The visual QA problem we are trying to solve can be seen as a form of clustering. Our task is to separate serious defect images from others. Another related task is to group defects according to their underlying causes. This is a classification problem, but since the data sets are huge and we don't have full class information we must use clustering tools.

Clustering is an enormously wide field of research and describing it in detail would easily fill dozens of books. However a detailed examination of clustering would be out of scope of this thesis, since in this work clustering has been a tool to evaluate system performance. In this chapter we describe some facets of clustering that are relevant from this point of view.

Clustering is one of those terms that everyone uses but which is notoriously difficult to define precisely. One very broad description is that clustering is the process of grouping together similar elements. While this seems like an overly loose definition, attempts to define the process more precisely rules out some things which could definitely be called clustering. Those interested in a more mathematical approach may read, for example, (Theodoridos and Koutroumbas, 1999).

One common approach is to view clustering as a way of doing unsupervised classification. Typically classification relies on supervised training which can be formalized as follows: given a data set $\{\mathbf{x}_i, C_i\}$, where $\mathbf{x}_i$ is a data vector and C_i is the corresponding class label, estimate the class of a new test vector $\mathbf{z}$. This is all well and good, but what if we don't have any class information. What can we do then?

Suppose we have partitioned the data vectors in disjoint groups. Now we can rephrase the classification problem: select the partition whose elements are the most similar to the query vector. The elements allow us to estimate some features in the query vector in a similar fashion as a class label. This gives us one possible objective for clustering: partition data so that the system gives us a maximal amount of information for any new sample $\mathbf{z}$. By suitably defining "information" we can obtain different kinds of algorithms.

The most basic and well known clustering method is *K-means clustering* (MacQueen, 1967). It has k cluster centers, which are usually initialized randomly. Then we do the following:

1. Map all data vectors $\mathbf{x}_i$ to their nearest cluster center
2. For every cluster, compute the center of mass of the data mapped to it. Move cluster center to that spot.
3. Go to step 1 until convergence

For a new vector $\mathbf{z}$ we find the nearest cluster center and then do inference based on the data vectors that form the cluster. K-means' drawbacks include the need to select k in advance. It is also sensitive to initialization and slow when used with very large data sets. Other popular clustering methods include Ward's clustering (Ward, 1963) and single-link hierarchical clustering (Sneath and Sokal, 1973). For a survey article discussing different aspects of clustering, see (Jain et al., 1999).

It should be noted that if we place each data vector into its own cluster, this method reduces to standard nearest neighbor classifier. Many real world clustering algorithms do some kind of unsupervised approximate k-NN search. We can also do the opposite and place each vector into several different clusters by defining a membership function. This approach is known as fuzzy clustering (Zadeh, 1965).

4.1 Validating clustering results

Most of the time simply doing a clustering is not sufficient, we also want to know how good our result is. If there is no measure for fitness there would be no need for clustering, because any partioning of the data space would be as good as any other. This would basically reduce clustering to grouping samples randomly, which is mostly an exercise in futility.

Cluster validity is a problem that has seen a lot of research. A commonly accepted approach is to calculate the compactness and separation of clusters, since that follows our intuitive feeling of cluster shape. One way to measure these properties is the *Davies–Bouldin index* (DBI) (Davies and Bouldin, 1979), which we have used in our experiments.

$$\mathrm{DBI} = \frac{1}{c}\sum_{i=1}^{c} \max_{i \neq j} \left(\frac{S_n(C_i) + S_n(C_j)}{S(C_i, C_j)} \right) \tag{4.1}$$

Here we have c clusters, $S_n(C_i)$ is the average distance of data vectors to the cluster center in cluster C_i. $S(C_i, C_j)$ is the distance between the centers of clusters C_i and C_j. This function gives small values for clusters that are dense and well separated from each other. This is consistent with our intuitive definition of a cluster.

Another popular clustering measures is the *Dunn index* (Dunn, 1974):

$$D = \min_{1 \leq j \leq c} \left(\min_{\substack{1 \leq j \leq c \\ j \neq i}} \left(\frac{d(C_i, C_j)}{\max_{1 \leq k \leq c} d'(C_k)} \right) \right). \tag{4.2}$$

The function $d(C_i, C_j)$ represents the distance between clusters C_i and C_j, whereas the function $d'(C_k)$ is the intracluster distance or "diameter". Similarly to DBI, Dunn Index favours dense, well-separated clusters that correspond to large values of D.

The *C index* (Hubert and Schultz, 1976) is another slightly different way of measuring cluster validity:

$$C = \frac{S - S_{\min}}{S_{\max} - S_{\min}} \tag{4.3}$$

This index bases its calculations on sums of pairwise distances. S is the sum of those pairs within the same cluster. Suppose there are l of those pairs. $S_{\min}$ is the sum of l smallest distance pairs among all data vectors and $S_{\max}$ is the sum of l largest pairs. Dense clusters correspond to small values of S, and thus small values of C indicate a good clustering.

Finally we look at a slightly different way of calculating cluster validity called the *Isolation index* (Pauwels and Frederix, 1999):

$$I_k = \frac{1}{N} \sum_{i=1}^{N} v_k(\mathbf{x}_i) \tag{4.4}$$

Here N indicates the amount of data vectors. The function $v_k(\mathbf{x}_i)$ tells which fraction of vector $\mathbf{x}_i$'s k nearest neighbors has been assigned to the same cluster as $\mathbf{x}_i$. Large values of I_k indicate that data vectors close to each other have been assigned to same clusters.

Unfortunately many real world data sets do not form such well separated clusters. Instead they are fuzzy and overlapping. On such data a well separated compact clustering would probably be nonoptimal, because it has different topology than the underlying data. There are several other indices and measures for clustering quality, but they all make some assumptions about the shape of the data cloud. This makes these methods more or less subjective.

We can obtain a more objective result if we have class information available. If a clustering algorithm is working correctly, any cluster should contain elements only or mostly from one class, depending on the application. If we label the clusters using, for example, majority voting, we can now use our clustering result for classification. Since clustering methods are almost always nonsupervised, we can naturally not reach the performance of supervised classification algorithms such as support vector machines (SVM). However if we find that some algorithm produces consistently good results we can extrapolate that it would perform well even on data sets that don't have any class information.

Ultimately the information we receive from clustering should not be considered definitive. What we usually derive from clustering are various kinds of hypotheses. They must then be verified by other means, such as human experts.

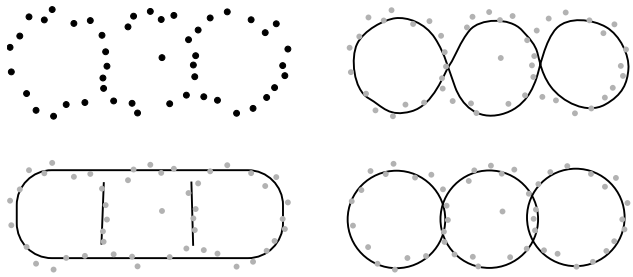

Figure 4.1: A data cloud and three possible cluster sources.

4.2 Clustering is an ill-posed problem

Sometimes even an expert can not tell whether one clustering is better than some other. For a simple example, see Figure 4.1. It shows a two dimensional data set whose internal structure seems to consist of three rings. There are several different ways of clustering this data. The trivial choices would be to assign each data point to its own cluster or all points to one big cluster. These clusterings are not very interesting, since they don't take into account the data cloud's internal structure.

In the figure we see three possible generators for the data. The first one consists of two consecutive figures of eight. In this case we only have one cluster, but its shape adequately describes the data set. The second clustering forms three clusters: a large oval-shaped cluster and two smaller vertical clusters. The third clustering consists of three consecutive rings.

These three clusterings give a very different view of the properties of the underlying data. It is very difficult to reliably determine which one of them is the "correct" one, even in this extremely simple case. If we had further information, such as class distributions, we could make a more educated guess. This ambiguity problem becomes even more pronounced when we have large databases and the data vector dimension grows.

Another serious problem can be seen in Figure 4.2. It has the same data cloud with two different scalings. The bottom one is obtained from the top one by spreading it in the x-dimension. The circles represent some possible clusterings.

In the first case the data points form a compact, round group, so placing them all in one cluster is reasonable. In the scaled version this is no longer the case. The data splits clearly into three subclusters, and some of those could easily be split further. This shows us that the underlying cluster structure comes and goes as the data is scaled. Usually this is dealt with by using a general data normalization approach such as normalizing variance or Karhunen-Loève transform (Haykin, 1994). This is also known as data whitening or normalization using PCA.

In most cases we use data vectors whose different elements have different and independent units. For example a measurement vector for a human being might contain mass which is measured in kilograms, height in centimeters, cholesterol in mmol/L and so on. The perceived cluster structure changes depending on used units of measurement and normalization. Thus we can see that clustering

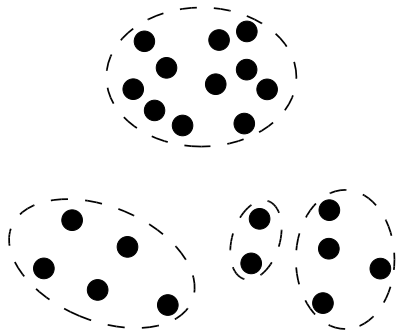

Figure 4.2: Two example clusterings for a data cloud with different scaling in the x-dimension.

as a problem is fundamentally ill-posed. Clustering results should not be taken at face value and they should be verified with other methods.

4.3 Hierarchical clustering

In regular clustering we only partition the data into disjoint sets. Hierarchical clustering (Everitt et al., 2001) arranges clusters in a usually tree-shaped hierarchical shape. The end result is one or more clusters, which have subclusters, subsubclusters and so on.

There are two basic ways of doing hierarchical clustering: agglomerative and divisive. In the first method we start with a bunch of clusters that have been obtained through some regular clustering method. In the extreme case we can assign each data vector to its own cluster. Then we combine clusters which are deemed to be the most similar, usually this means selecting clusters who are the closest in the data space. These are then combined into one cluster. This joining of clusters is continued until we have only one cluster.

Divisive clustering is the exact opposite: it starts with one big cluster and splits it into several smaller parts. This is repeated until some criterion for subclusters is reached. The process is very similar to the divide & conquer approach to data analysis explained in Chapter 5. A hierarchical clustering example can be seen in Figure 4.3.

There are two main benefits of hierarchical clustering. The first one is that it reveals interrelations between clusters. That is, it gives them a topology. The other one is that it gives us an easily browsed multi-resolution view of the data. If a cluster is too big or coarse, we can examine its subclusters. If we find that clusters are too small and, for example, corrupted by noise we can easily go up the hierarchy and get a larger view of the problem.

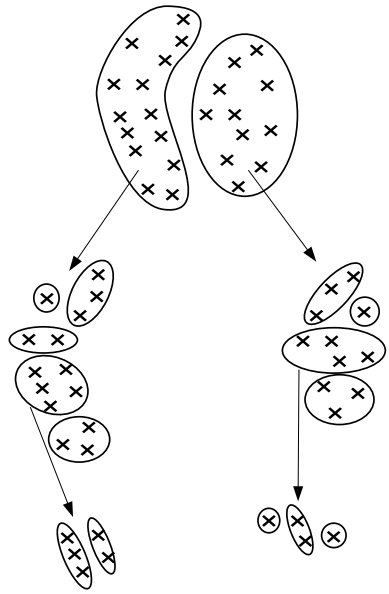

Figure 4.3: A simple hierarchical clustering example. Notice how only some subclusters are divided further.

Chapter 5

Managing large high dimensional data sets

5.1 Motivation

It is remarkable how several different problems in computer science, pattern recognition, and other fields reduce to calculating distances between points in d-dimensional vector spaces. Most analysis methods such as clustering, nearest-neighbor classification, and support vector machines (Vapnik, 2000) are based on finding out which data vectors are close to each other and which ones are distant.

If we look at this problem from a clustering and indexing point of view we find at least the following problem scenarios. Let us assume that we have a data set $\{\mathbf{d}_i\}$ and a query vector $\mathbf{x}$.

Simple search Find out whether the data set contains a vector which is identical to $\mathbf{x}$. This is mostly a problem in relational databases, where the elements of $\mathbf{x}$ may contain non-numerical data such as strings.

Range search Find all those data vectors $\mathbf{d}_i$ whose elements are within some limits related to $\mathbf{x}$.

Nearest neighbor search Find those k vectors among $\mathbf{d}_i$ that are the closest to $\mathbf{x}$.

Basic clustering Divide the elements in $\mathbf{d}_i$ in c groups so that elements within one group are similar and elements in different groups are different, and determine which group $\mathbf{x}$ belongs to.

In this thesis we mostly focus on the last two cases. The third case is commonly known as *k-nearest neighbor classification* (kNN). It is among the simplest classification algorithms, but still yields very good results in practice. Suppose we have a pre-classified data set $\{\mathbf{d}_i, C_i\}$. When we want to classify a new vector $\mathbf{x}$

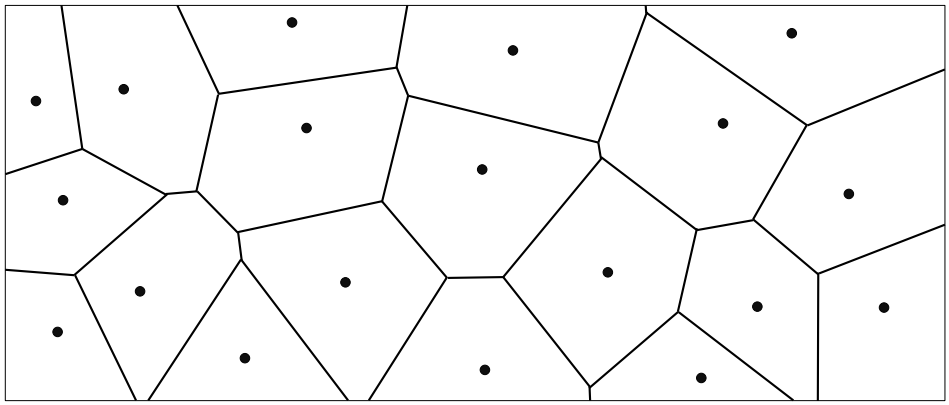

Figure 5.1: A simple two dimensional data set and the corresponding Voronoi regions.

we find the k nearest data vectors and use majority voting to assign a class C to the vector. If desired the neighbors can also be used to create a class probability distribution. kNN is very simple and it requires no training, but every query is an $O(N)$ operation, where N is the size of the data set. For many applications this is too slow.

5.2 Simple nearest neighbor search

As discussed above, finding the nearest neighbor of a data vector is a common subproblem in many applications and methods. In principle this problem is extremely simple to solve. Just calculate the distance between the query vector and all data vectors and pick the one with the smallest distance value. A simple two dimensional example of this problem can be seen in Figure 5.1.

The figure contains some data vectors $\mathbf{d}_i$, which are drawn with dark circles. The lines form so-called *Voronoi regions* (Aurenhammer, 1991) or cells for the vectors. A Voronoi region consists of all the points that are closer to data vector $\mathbf{d}_i$ than any other vector. Each vector $\mathbf{d}_i$ is inside its own Voronoi region. Any query vector and its nearest neighbor always reside in the same region. From this we see that an equivalent problem to finding a vector's nearest neighbor is determining which Voronoi region it maps to. Thus if we create the Voronoi regions for our data we can easily solve the problem. This requires dividing the data space into disjoint regions.

5.3 Data space partitioning

Nearest neighbor search is one of those problems that are easy to describe and understand, but is enormously hard to solve efficiently. While going through the

entire data set for every query is simple, it is extremely slow, especially when there are several queries. We want to speed up the search so that we can find the result by examining only a subset of the data vectors.

What this means, in essence, is dividing the data space in partitions. The search starts by finding out which partition the query vector falls into. The other partitions can be discarded. The partitioning process can then be done again on the subpartitions and so on. This gives us a hierarchical system that can efficiently be described using a standard search tree.

Even though this approach makes sense, we are not any closer to a working solution. A major subproblem is what kind of partition boundaries should be used. There are two approaches that are often used, because they are computationally inexpensive (Chávez et al., 2001; Gaede and Günther, 1998).

Hyperballs A hyperball is defined by its center $\mathbf{c}$ and radius r. It divides space in two parts: the first part that is inside the ball ($d(\mathbf{c}, \mathbf{x}) \leq r$) and the other that is outside it ($d(\mathbf{c}, \mathbf{x}) > r$). Hyperball based algorithms are often called *pivot methods*.

Hyperplanes A hyperplane is defined by two points $\mathbf{c_1}$ and $\mathbf{c_2}$. The partitioning hyperplane is formed by all the points whose distance to the two points is the same ($d(\mathbf{c_1}, \mathbf{x}) = d(\mathbf{c_2}, \mathbf{x})$).

Another subproblem is how the partitions should be placed in the data space. As we saw earlier, the edges of Voronoi regions are formed by portions of hyperplanes (line segments in 2D, polygons in 3D and so on). No matter how we place a hyperball in this space we can not follow the edges of the solution exactly.

Since the boundary lines are linear, hyperplanes seem to be the correct way of solving the partitioning problem. Figure 5.2 shows an example of a partitioning line. We can see that also in this case we intersect many Voronoi regions. If we use this partitioning, all the shaded areas get assigned the wrong nearest neighbor.

This figure shows that any non-trivial problem can not be partitioned without errors using hyperplanes. We can overcome this problem using a simple trick. Instead of partitioning data into two segments, we divide it into three groups: (1) those that are wholly on one side of the partition boundary, (2) those that are wholly on the other side and (3) those whose Voronoi region is intersected by the partition boundary.

Now we check which side of the boundary the query vector is. Let us assume it is on the same side as group (1). Now we can safely discard all elements in group (2) and only focus on groups (1) and (3). Similarly the query vector was on the same side as group (2), we could discard all elements in group (1). We have now discarded approximately half of the search space without causing any error.

As nice as this algorithm seems, there is unfortunately still one very large problem: we need to calculate which Voronoi cells the partition boundary actually dissects. This turns out to be a very expensive operation. The actual inter-

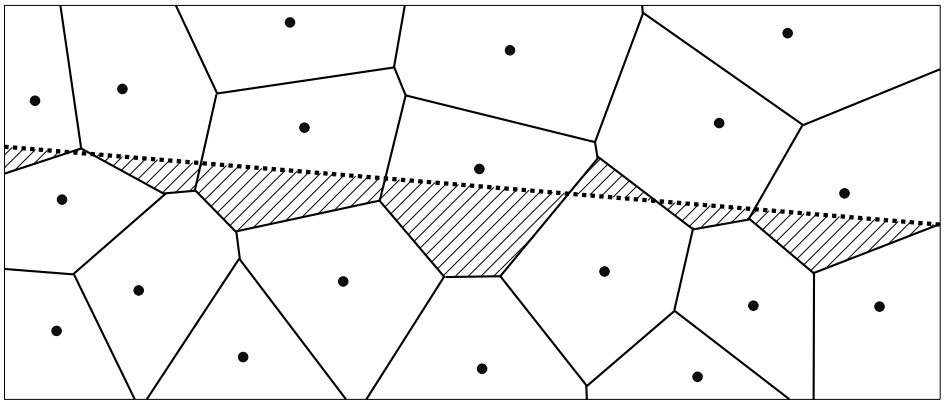

Figure 5.2: Partitioning the data space with a hyperplane, shaded areas get assigned to incorrect nearest neighbor after the split.

sections are relatively simple to calculate but creating the Voronoi tesselation is not.

It turns out that Voronoi tesselation is related to the problem of computing convex hulls. Even the most efficient convex hull algorithms, such as *QHull* (Barber et al., 1996), get exponentially slower as the data dimension grows. This makes large-dimensional problems intractable.

Another way of looking at the problem is that any partitioning scheme must somehow parameterize the Voronoi cell boundaries. Simply trying to visualize these boundaries in one, two, and three dimensions shows how difficulty of the problem grows as a function of the dimension. Attempts to visualize any data set with 4 or more dimensions is impossible due to the human brain's inability to understand more than three spatial dimensions.

If we look at the discussion above we find that we are no nearer to solving our problem: how to do exact nearest neighbor queries efficiently. It turns out that we can not do this. Even algorithms that do not split the space into disjoint regions, such as the coordinate sorting method (Friedman et al., 1975), become extremely slow as the data dimension grows. Since exact results are not obtainable we have to do the next best thing and fake them. This means computing an approximate solution and using that. It turns out that this can be done efficiently and yields quite good results.

5.4 A look into classical indexing methods

No look into managing large data sets would be complete without examining database systems. Database management has been researched since the birth of computers. The most common database type is the *relational database* (Date, 2003). Its data usually consists of strings or numbers and are arranged in tables. Queries usually specify only a few variables, which means they have low

dimension. Basic database methods are not suitable for our purpose, but there are several advanced techniques that are worth examining.

These can be roughly divided into two different classes: metric access methods and spatial access methods. Metric access methods (MAMs) have very few requirements. They only require a sensible distance function, or metric, between data elements $\mathbf{x}_i$. While this is a very interesting and challenging area of research, we feel that it is out of scope of this thesis and refer interested readers to (Chávez et al., 2001).

The other class of methods are called spatial access methods (SAMs). While MAMs operate on metric spaces, SAMs require vector spaces, the most common being the euclidean space $\mathbb{R}^d$. This is the framework for our CBIR methods, which are based on real valued feature vectors. The basic idea of SAMs is to leverage geometric and other properties of vector spaces to speed up searches. The most common approach combines divide-and-conquer to data space partitioning as discussed earlier in this chapter. It should be noted that since all vector spaces are also metric spaces, all MAMs can also be used as SAMs.

Arguably the simplest way of partitioning multidimensional data spaces is the *quadtree* (Samet, 1984)[1]. Quadtrees operate on rectangular two dimensional areas. If the current region has too many data elements it is divided into four subregions. These subregions have the same shape as the original region. The subregions are then further subdivided until the data sets they contain are deemed "simple enough". The subdivision process can be seen on the left in Figure 5.3. While quadtrees can only handle 2D data, the same principle can be applied to larger dimensions. For example the corresponding subdivision method in three dimensions is called an *octree*.

The k*d-tree* (Bentley, 1979) subdivides space with hyperplanes that are aligned along the coordinate axes. The right side of Figure 5.3 illustrates the splitting process. First the area is split in two with a vertical hyperplane. These areas are split with horizontal hyperplanes. The left one is split in the second image from the top. It results in two new areas, which are to be split with vertical hyperplanes. Their subregions are again divided with horizontal lines and so on. If there are more than two dimensions we cycle through them in a similar fashion. kd-tree is very popular and is used e.g. in databases.

A different approach to partitioning is taken by the *R-tree* (Guttman, 1984). It consists of a hierarchical structure of possibly overlapping iso-oriented boxes. A simple two layer R-tree can be seen in Figure 5.4. The basic idea is that all data points lie inside one or more boxes. R-tree's aim is to cover all data points with sufficient accuracy using as few boxes as possible. When querying we first find all the top level boxes that contain the query point. Then we recursively examine their children in the same way until we reach the leaf nodes. Data insertion is done in two phases. First we find the nearest box using a regular query. If the added vector is inside the box then we add it to the box's element list. Otherwise the box is expanded to contain the vector, possibly causing the parent boxes to be expanded also. When the box is deemed to contain too many vectors it is

[1]There are several slightly different methods that are all called quadtrees. We describe the region quadtree.

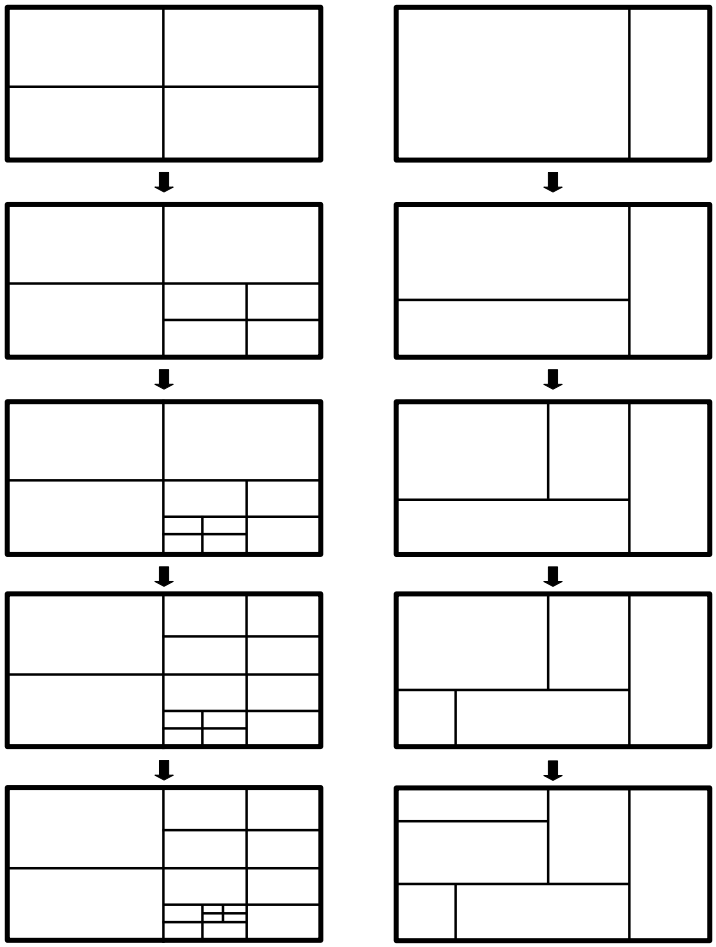

Figure 5.3: Subdividing 2D space with a quadtree (left) and a kd-tree (right).

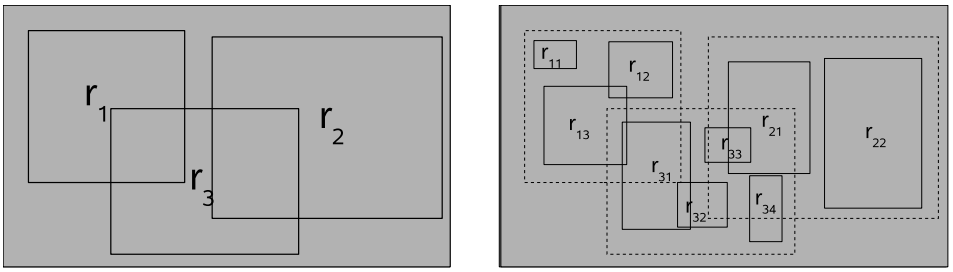

Figure 5.4: Two layers of a simple R-tree.

subdivided.

R-tree is an efficient indexing method when data dimension is relatively low, but it is susceptible to degenerate performance for some data sets. To overcome this problem several refined versions of R-tree have been suggested. Two notable examples are *R+-tree* (Sellis et al., 1987) and *R*-tree* (Beckmann et al., 1990).

As was discussed earlier, all these methods become slower than simple linear search when data dimension grows. In R-tree's case this is caused by extreme overlapping of hyperrectangles. *X-tree* (Berchtold et al., 1996) tries to work around this by creating a hybrid of an R-tree and a linear search structure. X-tree tries to maintain an R-tree like hierarchical structure as long as possible. When some node overlaps others too much it is replaced with a linear search structure called a supernode. In extreme cases X-tree reduces to a linear list of data. According to the authors' experiments X-tree can be up to 450 times faster than R*-tree. This makes it one of the most efficient high dimensional query structures.

While the methods discussed above partition data with iso-oriented hyperplanes, the *binary space partition (BSP) tree* (Fuchs et al., 1980) uses freely oriented hyperplanes. This allows it to find more efficient partitioning planes, but the tradeoff is increased computational cost. BSP is probably the best known data partitioning scheme among the general population. This is due to its use in the computer game *Doom* (Carmack et al., 1993). BSP made it possible to render a large, fully texture-mapped game world in real time with very low powered home computers of early 1990s. BSP has since been used in dozens of computer games.

While we have described several algorithms, it is only possible to scratch the surface in a short review. There are literally hundreds of other methods each with their own advantages and disadvantages. The interested reader is recommended to seek out a survey article, such as (Gaede and Günther, 1998).

5.5 Problems with large data sets

Large classified datasets have one fundamental problem. It is extremely difficult to obtain objectively classified high quality data sets. To see why, let us examine different kinds of data we could have. There are two basic types of data sets: synthetic and real world.

Creating synthetic data sets is relatively simple. One defines some generator functions or distributions for different classes and use these to create the required number of points. The resulting data cloud is readily usable for comparison tests and the like. The problem is that ultimately we want to use the methods to analyze real world data. Several of these problems do not follow the distributions that are used to create the synthetic data. We see that results obtained with synthetic data, while useful, are not definitive. Therefore we want to test the different systems with real world data.

Real world data can also be divided into two classes. The first one is data that has been automatically classified. This data has a serious bias if it is used in comparison testing. These tests do not measure the absolute performance of the different methods. Rather they measure how much their classifications differ from the baseline method. Therefore if some method perfoms absolutely better than the baseline method, but in a different way, it is penalized. This is undesirable.

The best kind of testing data is therefore real life data that has been pre-classified by a human expert. Unfortunately this is also the most difficult and expensive data to produce. Suppose we have a trained human doing the classification. Let us further suppose that he can do a single classification in 30 seconds. This may contain analyzing images, specimen or other such task. How long does it take for this person to create a database of 100 000 samples?

If we assume he works 8 hours a day only on this task, it takes $30 \cdot 100000/(60 \cdot 60 \cdot 8) \approx 104$ working days. Adding weekends we find that this corresponds to almost five months. It is very clear that working this long invariably leads to non-optimal performance due to tiring and other psychological factors. Similarly a database with one million elements would take almost four years to create.

One could try to work around this by parallelizing the classification task to several people. The problem with this is that different people have different opinions on what is the correct classification. This may lead to different biases for different portions of the training data. Unfortunately this does not change the fact that hiring experts full time for several years is prohibitively expensive.

All this means that almost all large classified data sets are intrinsically defective in some way or another. This must be kept in mind whenever dealing with them.

Chapter 6

Data analysis algorithms based on and inspired by the SOM

Multidimensional data analysis is a problem with very large and long reaching roots. Like many other disciplines, it got a big boost with the birth of computers in the 1960s. Since then a plethora of different algorithms and methods have been invented, and in several cases re-invented. In this chapter we examine some methods based on self-organizing networks.

The methods discussed here are based on neural computation and fields closely related to it. Algorithms used in databases and classical computer science were already discussed in section 5.4. Specifically we focus on methods related to the *Self-organizing map (SOM)* (Kohonen, 2001). SOM is an unsupervised data analysis method that tries to emulate the cognitive processes in the brain. It should be noted that the methods discussed earlier can be directly applied to training a SOM (Cuadros-Vargas and Romero, 2005), but here we examine new and different architecture types. The methods discussed are generally used for clustering and visualization tasks.

SOM is especially useful for visualization tasks and is used in thousands of academic and real world processes. The basic algorithm moves a two dimensional, usually hexagonal, $m \times n$ grid in the data space. During training the grid seeks out the shape of the data set. In SOM each node has a prototype vector $\mathbf{w}_i$, which is updated during training. The vectors in the data set are used one at a time. There are two main phases: finding the best matching unit and updating the nodes. The first step is simple: select the node closest to the current training vector $\mathbf{x}$:

$$c = i_{\mathrm{BMU}} = \arg\min \|\mathbf{x} - \mathbf{w}_i\| . \tag{6.1}$$

Updating is done with the Kohonen learning rule:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{w}_i(t)]. \tag{6.2}$$

This formula shows how all nodes are updated towards the training vector. SOM's power arises from the fact that the training factor h_{ci} depends on how far along the grid the current prototype $\mathbf{w}_i$ is from the BMU. This distance is not measured in the data space but along the 2D grid. This grid distance d between two nodes r_c and r_i is then input to a neighborhood function:

$$h_{ci}(t) = \alpha(t) \exp\left(\frac{-d(r_c, r_i)^2}{2\sigma^2(t)}\right). \tag{6.3}$$

In this formula α is an adaptation value that usually decreases exponentially with time. The width of the neighborhood is defined by σ, which is usually varied so that the neighborhood gets narrower and narrower as time passes.

The basic idea is that the further away from the BMU we go the less the nodes are updated. Here "further away" is defined as distance along the grid rather than in the input space. One way of visualizing the training is to consider the SOM as an elastic plane. Each training vector stretches the map slightly. Eventually the SOM adapts itself to the shape of the data manifold.

6.1 SOM variants

SOM is a popular and widely used method, but it has some drawbacks. The main ones are that you have to choose the network size in advance and that the grid is not flexible enough for some applications. To overcome these problems several variants of SOM have been proposed.

Neural gas (Martinez and Schulten, 1991) (NG) discards the SOM grid to achieve a more flexible network for better topology preservation. Connections between cells are added between the best matching unit and second best matching unit whenever a data vector is presented. Old connections are pruned away periodically. The resulting grid is not regular as in the SOM, which makes it easier to adapt to complex data manifolds.

Growing neural gas (GNG) is a version of neural gas that automatically adjusts its size to match the data (Fritzke, 1995b). It tracks a quantization error for each node and periodically adds a new node between the node with the biggest total error and its immediate neighbor with the largest error. The resulting network is very similar to that of neural gas, but without the need to select the network size in advance.

Dynamic cell structure (Bruske and Sommer, 1997) does not use a regular grid like SOM. Instead it starts small and adds neurons to underrepresented areas and connects them to nearby neurons. It can also trim unnecessary connections between neurons. This allows separation of distant areas of the data space.

Incremental grid growing (Blackmore and Miikkulainen, 1993) relaxes SOM's requirements that nodes have links to all their neighbors. Neighboring grid nodes

only have a link if they are deemed to be close to each other in the data space. Another main feature is that the amount of nodes increases as the training continues.

Growing grid (Fritzke, 1995a) starts with a small map and grows it by adding rows or columns to locations that are deemed underrepresented. This makes the map automatically adjust its height/width ratio to obtain a better fit. The system has no functions with user-choosable parameters, so they don't have to be determined by trial and error.

Growing cell structure (Fritzke, 1994) (GCS) is based on nodes that form hypertetrahedrons, which can be one, two, three or higher dimensional. During training new hypertetrahedrons are formed and superfluous ones are removed. Even though the lattice structure is very irregular it can be visualized as long as the hypertetrahedrons' dimensionality is less than four. GCS can also be used for supervised learning by using it as a base for an RBF network (Moody and Darken, 1989).

Most systems use a two-dimensional rectangular grid like the regular SOM. *Hypercubical Self-Organizing Map* (Bauer and Villmann, 1997) extends the method by allowing higher dimensional grid lattices that take hypercubical form. Their method starts with a small SOM that is trained as usual. The grid is grown peridically. There are two different ways of growing: adding rows or columns to the existing dimensions, or adding a totally new dimension. The grid can thus grow to have 3D lattice structure, then 4D and so on. The growth is done in the direction of largest error amplitude, as it is usually an indication of the map folding to represent a higher dimensional data manifold.

Growing Self-Organizing Map (GSOM) (Alahakoon et al., 2000) starts with a small SOM (usually 2×2) and grows it outwards. The growth direction is decided by calculating a value called the error distance. New nodes are grown from the one with the biggest error. The resulting grids are two dimensional and regular as in the SOM, but they are not rectangular in shape.

High-Dimensional Growing Self-Organizing Map (HDGSOM) (Amarasiri et al., 2004) is an improvement on the GSOM algorithm to make it better cope with high dimensional data. This is obtained mostly by adding a special "calibration phase" to the training. This phase spreads out the nodes more evenly to the data space than the regular GSOM algorithm. The result is a less twisted map than with the GSOM. Adding a random component to the training in a similar fashion as in simulated annealing seems to improve the clustering results with very little computational effort (Amarasiri et al., 2005).

6.2 Tree-shaped systems

Another problem with the SOM is that it becomes slow when the data set size grows. Various acceleration methods have been suggested. A common approach is to subdivide the space hierarchically and represent the division with a search tree.

Tree-Structured Self-Organizing Map (TS-SOM) (Koikkalainen and Oja, 1990; Koikkalainen, 1994) is a simple hierarchical extension of the SOM. It consists of several different sized SOMs that are arranged in a pyramidal shape. A common choice for map sizes is 4×4, followed by 16×16, 64×64 and so on. Training proceeds one layer at a time. The first layer is trained just like a regular SOM. Then the next layer is divided into smaller groups. Each group is given a parent node in the top layer. Finding the BMU is an iterative process: first the BMU of the first map is found. Then the BMU is chosen among its children. More layers can be added in the same fashion. This scheme fastens the BMU search from $O(N)$ to $O(\log N)$. Using lateral searching, that is, keeping track of more than one BMU per layer, further improves the results with only slight increase in computational time.

Self-growing neural tree (Wen et al., 1992) (SGNT) starts with a single node. It is split whenever the distance between the BMU and the input vector is larger than some threshold ξ. The update factor does not depend on node distance as in ETree (chapter 7), but rather simply decreases as time passes. The resulting tree is nonoptimal, so a second round follows in which the tree is pruned and balanced. Experimental results indicate good performance, but the methodology and compared methods are not explained thoroughly.

Self-organizing neural grove (Inoue and Narihisa, 2003) (SONG) extends SGNT by effectively combining the output of several SGNTs. The final result is obtained by committee voting. SONG also adds and modifies the pruning rules of SGNT. The pruning has two phases, a supervised pruning round that reduces computation cost, and a structure improving pruning round which removes tedious leaf nodes by cross-validation. SONG is found to perform slightly better than k-means in various classification tasks, but it is noticeably faster.

(Bhandarkar et al., 1997) present the *hierarchical SOM* (HSOM) as a tool for segmenting images. The system contains SOMS of size 1×1, 2×2, 4×4 and so on. While most other algorithms have a top-to-bottom training method, HSOM does it bottom-to-top. First the largest (bottom level) SOM is trained in the usual way. Then the second largest SOM is trained. In this phase the data is not used directly, but it is first fed to the bottom layer SOM and the resulting BMU node vectors are fed to the layer to be trained. For the third lowest layer the data is first fed through the lowest layer, then the second lowest and so on. The segmentation is then done by traversing the tree from the top downwards and stopping at the suitable level. Experiments show that HSOM outperforms a Canny edge detector based segmentation scheme.

Growing hierarchical SOM, or GHSOM (Dittenbach et al., 2001), is similar to TS-SOM. Instead of having one SOM in each layer, GHSOM has several. It starts by training a small map, say 3×3. After a while the average quantization error is computed. Depending on the error the map is either grown by adding a row or column, or some nodes are assigned child SOMs. All data that would be mapped to a parent is instead used to train the child map. GHSOM thus has two independent growth directions: lateral (size of individual SOMs) and hierarchical (depth of the "tree"). Relative growth between these can be controlled with parameters (Dittenbach et al., 2005).

TreeGCS (Hodge and Austin, 2001) is a hierarchical improvement to Growing cell structures. When nodes and connections between them are being removed from GCS, the group of cells may be split into two or more different parts. TreeGCS maintains a tree that is updated whenever these splits occur, but it is not used in the BMU search. The final tree shows how the different clusters got separated from each other. This can be compared to hierarchical agglomerative clustering, such as Ward's clustering (Ward, 1963).

Self-organizing tree map (SOTM) (Kong and Guan, 1998) is similar to the growing grid, but it forms a hierarchical tree structure rather than a flat grid. The neighborhood function is replaced with a hierachy control function that defines the tree growth. When applied to impulse noise reduction in digital images the method outperforms median filters and other similar methods.

Hierarchical GCS (HiGS) (Burzewski and Mohan, 1996) is another extension of GCS. The basic idea is similar to GHSOM, but the layers consist of slightly modified GCS networks instead of SOMs. Experiments show that the system adapts to separate clusters about as well as Neural gas, but the hierarchical topology makes the training faster.

Competitive neural trees (CNeT) (Behnke and Karayiannis, 1998) is a tree shaped system that is similar to ETree (see chapter 7). The system has greedy BMU search and the update rule is the same as in SOM. There are several differences. One of them is that there is no neighborhood function, only the BMU is updated. The nodes also change their behaviour depending on how old they are. Finally, the splitting of nodes is done in a supervised fashion. Therefore CNeT always requires class information, and therefore is not suitable for unsupervised learning. Experiments with various data show that the method performs favorably when compared to various other classifiers.

Structurally adaptive intelligent neural tree (SAINT) (Song and Lee, 1998) is another tree-shaped SOM variant that is very similar to GHSOM. Experiments with several hand-written text data sets show that the system can perform better than a SPA tree model (Li et al., 1995).

Self-Organizing Tree (S-Tree) (Campos and Carpenter, 2001) is a binary tree that grows in much the same way as ETree. The splitting rule is more complex. It is based on calculating the cumulative error and splitting a node when the error grows large enough. The algorithm also prunes away nodes that it thinks are superfluous, i.e. they have small enough error values. Experiments on clustering, vector quantization, and image compression show the feasibility of the system.

Dynamic adaptive hybrid model (Hung and Wermter, 2003) (DASH) is a hierarchical system that combines features of GNG and GHSOM. It also has a method for automatically tuning its parameters for each data set. The training consists of consecutive phases of learning, pruning, and growing. This makes DASH suitable for non-stationary data (Hung and Wermter, 2005).

Hyperbolic self-organizing map (Ontrup and Ritter, 2001) embeds the SOM nodes in a hyperbolic space instead of a regular flat euclidean space. The advantage is a better visualization of large maps due to an adjustable "fish eye" effect of

the transformation. The system can also be modified so that it forms a hierarchical search structure (Ontrup and Ritter, 2005). This speeds up the training significantly.

6.3 Case studies

In this section we examine the performance of various SOM methods. While there are hundreds of application papers, we focus on those that are related to CBIR or compare some mentioned algorithms in a meaningful way. The latter ones are somewhat rare, since baseline comparison methods are usually k-means or other similar algorithm. Another reason is that most algorithms never become popular enough to attract independent performance comparison tests.

6.3.1 Method comparisons

In (Köhle and Merkl, 1996) the visualization properties of SOM and GCS is compared. The data consists of high dimensional text data describing computer terminology. The results show that GCS separates clusters more strongly. In contrast specifying cluster boundaries on the SOM is not automatic and thus more difficult. The tradeoff is that SOM is very stable with regard to initialization and parameters. Also since the different clusters are separated in GCS, information about their interrelations is lost in the visualization.

The performance of SOM and two slight variations of GCS is compared in (Fritzke, 1992). Three different artificial 2-dimensional data sets are used to calculate three different performance values: topology preservation, distribution error and mean square quantization error. GCS outperformed SOM in all tests except for the simplest case. Visualization aspects were not considered.

The pattern recognition properties of SOM and Neural Gas are compared in (Zhang et al., 1998). The experiments were done with 10 000 handwritten digits. Neural gas is found to achieve better results (approximately 95% versus 92%) and it is also faster.

6.3.2 Applications to content-based retrieval

(Niskanen, 2003) describes a method of using SOM to detect wood imperfections such as knots. A SOM is trained with sample images taken from various wood boards. Once the SOM has organised the images to a 2D grid the human operator can draw decision boundaries on the map. After this the SOM can be used for classification. Using this method classification rates of almost 80% can be reached. This is almost as good as using a kNN classifier.

The suitability of GCS to video browsing is examined in (Koprinska and Clark, 2004). 16 dimensional gray scale histograms of the frames' dc components are

used to train a TreeGCS. The result is a relatively good hierarchical set of key frames. This is useful in visualization and video browsing.

In (Wu et al., 2005) a method called *growing hierarchical self-organizing quadtree map* is discussed and applied to content-based image retrieval. The proposed method is a cross between TS-SOM and GHSOM. The system is tested with a database of 1000 images and 13 features for each image. The resulting system is relatively fast and robust to various kinds of image degradations.

Using SOMs for video browsing is described in (Eidenberger, 2004). The system has two browsing methods: a tree-shaped index of key frames and a content index for the segments between key frames. The trees are composed of several layers of SOMs. This approach makes it possible to combine temporal and contextual information in the same query, mimicking the way a human brain stores information. A further advantage is that the system can be used from a regular web browser.

A scheme for using hierarchical SOMs for image indexing is presented in (Zhang and Zhong, 1995). Using the SOM based kNN search method is almost as good as doing a regular kNN query, but the query happens approximately five times as fast using a database of 1008 images.

A different kind of hierarchical SOM image retrieval system is described in (Sethi and Coman, 1999). The system has three layers of SOMs, where each SOM node in the upper layers has its own child SOM. Features consisted of HSV histograms of image subblocks. The example queries show that the system is working, but there are no performance numbers in the paper.

(Muneesawang and Guan, 2002) combines SOTM with a supervised query methods such as RBF networks. It is found that RBF networks outperform basic relevance feedback in all test cases. The resulting system achieves quite good query results with only a few iterations.

The PicSOM system (see section 3.4) that we have used in our experiments has been applied to several other CBIR tasks, such as video retrieval (Koskela et al., 2005), mail order catalog browsing (Viitaniemi and Laaksonen, 2003), and facial image database indexing (Yang and Laaksonen, 2005). The performance levels have been roughly similar to the ones in our experiments. These results seem to indicate PicSOM's suitability to function as a CBIR research platform.

Chapter 7

The Evolving Tree

The Evolving Tree (ETree) is a novel neural network system first introduced by the author in Publication III. It has been designed to solve unsupervised learning problems, such as clustering or data mining. What separates ETree from classical methods is that it is very efficient with large scale problems. In data analysis this usually means one of two things: either the amount of data vectors is very large or their dimension is large. Informal names for these phenomena are *information explosion* and *the curse of dimensionality*, respectively. ETree has been designed to cope with both of them.

7.1 Algorithm description

This section presents the Evolving Tree algorithm. For a detailed discussion see Publicatios IV, VI and VII.

Like many other neural computation algorithms, the Evolving Tree is based on nodes with prototype vectors and connections between those nodes. More specifically ETree is a tree-structured network with two different kinds of nodes: leaf nodes and trunk nodes. These can be seen in Figure 7.1, where white nodes are trunk nodes and black ones are leaf nodes. Leaf nodes have the same function as nodes in classical algorithms like SOM and Neural gas. That is, their interconnections and locations in the data space define how the system explains the data. The trunk nodes have two tasks: they act as a search tree to the leaf nodes, which makes best matching unit (BMU) searches faster and they also form a topology for the leaf nodes.

To obtain a working system we also need a training algorithm. Its job is to position the nodes in the data space so that we obtain the best possible explanation for the data. Usually this is done by defining a fitness criterion such as mean quantization error and then optimizing that. The basic training algorithm of ETree is patterned after the SOM. Like SOM it has two basic portions, finding the BMU and updating the leaf nodes towards the training vector. To see how

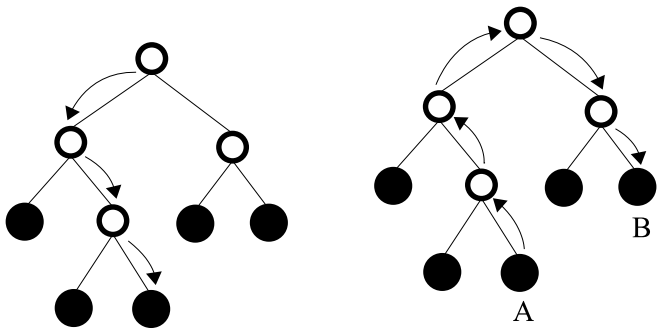

Figure 7.1: Basic operations of ETree, finding the BMU (left) and calculating tree distance (right).

the algorithm works, let us now go through it step by step.

7.1.1 Basic operations and formulas

Suppose we have a training data set $\{\mathbf{x}(t)\}$. ETree starts by placing a single node in the data space, usually in the center of mass of the data cloud. The first step in the training algorithm is finding the BMU. Since we have only one node, this step is trivial. Then we update the node using the Kohonen learning rule (Kohonen, 2001):

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{w}_i(t)]. \tag{7.1}$$

The h_{ci} is a function that defines how much the prototype vector is adapted. It is usually a gaussian function such as this:

$$h_{ci}(t) = \alpha(t) \exp\left(\frac{-d(r_c, r_i)^2}{2\sigma^2(t)}\right). \tag{7.2}$$

These functions and their parameters are essentially the same as in equations 6.2 and 6.3.

What we have described thus far is essentially a 1×1 SOM, which is hardly an advanced data analysis method. To get a bit further we add to the node i a counter b_i which tells how many times it has been the BMU. We also set a *division threshold* θ. When the node's counter b_i reaches the value θ we split the node. That is we give it some amount of children (this value is called the *splitting factor* or the *fanout*). In the data space the child nodes are initially placed in the same place as their parent node.

At this point we face two new problems: how to define the BMU and how to update the nodes. Suppose we have a slightly larger tree, such as the one shown in Figure 7.1. The trunk nodes are marked with white circles whereas the leaf nodes are black. First we have to find the BMU. This process is shown in the first

image of Figure 7.1. We use a greedy top-down search. That is, we examine all the child nodes of the root and select the one that is closest to the query vector in the data space. Then we examine its children and select the best match. This is repeated until we reach a leaf node, which is chosen as the BMU. As was discussed in section 5.3, the chosen node is not necessarily the true, global BMU, since ETree creates only an approximate index to data.

Updating nodes is a bit trickier. We use the same training formulas as the SOM, but that gives us a problem. One of the fundamental properties of the SOM is the neighborhood function (equation 7.2), which tells the topological distance of two nodes along the grid. The larger this distance is, the less adaptation is done to the node. ETree does not have any grid structure in it. The leaf nodes are simply scattered in the data space. We use a similar metric called the *tree distance*. It is shown in the second image in Figure 7.1.

Tree distance $d(r_c, r_i)$ between two leaf nodes r_c and r_i ($c \neq i$) is defined as the number of "hops" it takes to go from one node to the other along the tree minus one. One is subtracted because the shortest distance between two leaf nodes is two hops. This happens when they share a common parent and we want these to have the minimal distance of one. When $c = i$ the distance is 0. Now we can substitute grid distance with tree distance and can thus use all the same training formulas as in the SOM.

If these parameters are set so that the neighborhood is extremely narrow, only the BMU gets moved. This is equivalent to totally ignoring the tree topology. We experimented with several parameter values and found that this kind of a narrow neighborhood performed consistently worse than a slightly wider one. Thus the neighborhood function and tree topology are useful as we get better performance than without them.

It should be noted that only leaf nodes are adapted according to these formulas. Once a leaf node is transformed into a trunk node, it is no longer moved at all.

7.1.2 Inhibiting growth

A common problem with data modelling methods is overfitting. It is caused by modelling the training set too closely, which worsens the system's generalization abilities. In the case of ETree we want to limit the amount of leaf nodes. This is directly related to another fundamental choice: when to stop training.

ETree's final size is determined by a method based on regularization. After each one pass through the data (an epoch) the BMU counter b_i of each leaf node is multiplied by a constant factor γ, where $0 \leq \gamma \leq 1$. Empirically we have discovered that values between 0.85 and 0.95 produce quite good results. This decrease in BMU counts inhibits the tree growth. After each epoch we measure how much the tree has grown. If the growth is very small, say less than 5%, we can stop the training process.

7.1.3 Optimizing leaf node locations

The final phase of ETree training is fine-tuning the leaf node locations. This is done with a simple k-means -based algorithm. First all training data is mapped to leaf nodes. Then each leaf node is moved to the center of mass of all the vectors mapped to it. This procedure is repeated a few times to obtain the final ETree.

This process can be motivated by examining the final stages of regular ETree training. At the end, each node has only a few data vectors, and is not split any further. The updating neighborhood has also shrunk, so that in practice only the BMU gets moved. Thus each leaf node is trained with its data as if it was a single node SOM. This is roughly equivalent to an iterative k-means algorithm. By using the k-means optimization directly we can reach convergence more quickly.

7.2 Computational complexity

An important aspect of any computer algorithm is its computational complexity. In this case we want to find out the computational complexity as a function of data set size and data dimension. Let us use the following notation.

d data space dimension

N data set size

h tree depth

u # of updated neighbours

b branching factor (fanout)

θ splitting threshold

l # of leaf nodes

r # of k-means rounds

The computational complexity for a single training vector is the sum of finding the best matching unit (C_{BMU}) and updating the neighbours (C_{ud}). The complexity of one epoch is simply the complexity of a single operation multiplied by the epoch size plus the complexity of weight decay. When we add to this the complexity of the final k-means adjustment we obtain the total complexity.

To find the BMU we have to calculate a vector distance b times at every level of the tree. The amount of calculations needed is

$$C_{BMU} = d \cdot h \cdot b. \tag{7.3}$$

Updating the leaf locations means moving u nodes. We also have to add the cost of splitting the leaf nodes. This happens, on average, every $1/\theta$ steps, making the total cost of the update for a single vector

$$C_{ud} = u \cdot d + \frac{b}{\theta} \tag{7.4}$$

Weight decay is a very light operation, which is run at most once per epoch. It affects every child node once, so the complexity is

$$C_{wd} = l \cdot d \tag{7.5}$$

The final part of the training is the k-means updating, which is quite simple to calculate. That part consists of mapping all data vectors to leaf nodes and then moving the leaf nodes to the center of mass of their respective vectors.

$$C_{km} = r \cdot N(d \cdot h \cdot b + d). \tag{7.6}$$

Adding these and multiplying by the epoch size N we find that the amount of calculations needed for a single epoch is

$$\begin{aligned} C_{tot} &= N\left(C_{BMU} + C_{ud}\right) + C_{wd} + C_{km} && (7.7)\\ &= N\left(d \cdot h \cdot b + u \cdot d + \frac{b}{\theta}\right) + l \cdot d + r \cdot N(d \cdot h \cdot b + d). && (7.8) \end{aligned}$$

This function can be simplified by noting that θ, l, and r are constants and can thus be eliminated. u can also be set to a constant value, since our implementation only updates those nodes, whose training factor is larger than a predefined value. During a single round the training factor is only defined by the tree distance, thus only a fixed subset of leaf nodes near the BMU gets updated.

The algorithm forms a search tree by divide and conquer, therefore it follows that $h \propto \log N$. The branching factor b is a bit more problematic. It is usually set to a constant for a tree. On the other hand people seem to intuitively choose a larger b if their data set dimension is large, presumably because there is "more space" for the nodes to disperse into. The exact nature of this relationship varies from one person to the next and determining it is more of a question of psychology rather than engineering. Since that would be out of scope of this text we will simply define the value as $b = f(d)$. Experience tells us that the rate of growth of f is sublinear and likely at most logarithmic.

Putting all this together we find that the computational complexity as a function of N and d is

$$O(f(d) \cdot N \cdot \log N + N \cdot d + N \cdot f(d) + N \cdot d \cdot f(d)). \tag{7.9}$$

Let us first examine the complexity caused by data set size. In this case d is a constant and the complexity reduces to

$$O(N \cdot \log N). \tag{7.10}$$

This is a very nice result. Most classical methods are quadratic, which means that their complexity is $O(N^2)$. These methods include SOM and k-means clustering.

If, on the other hand, we only want to examine the effect of the data dimension, N is a constant and thus we get:

$$O(d \cdot f(d)). \tag{7.11}$$

Depending on how we set the function f, we can obtain a linear dependency in the best case and quadratic in the absolute worst case. As was discussed above, a very probable upper limit for f is a logarithm. This gives the complexity $O(d \cdot \log d)$, though we can reach a linear complexity if we simply fix b regardless of d. This is also a positive result. The earlier discussed classical methods have also linear complexity, but most hierarchical structures, such as the R-tree, have an *exponential* complexity (Chávez et al., 2001). As was discussed in section 5.3, this is because they must produce an exact result.

7.3 Benefits and disadvantages

The main advantage of ETree is that it is very fast. This allows it to tackle large scale problems that have been too complex for classical methods. Another very good property is that the algorithm and its implementation are very simple. This allows other people to understand and modify the system to suit their needs.

The downside to ETree's speed is that it only computes approximative answers. As we have seen it is not guaranteed to find the true BMU for any query vector, and thus the clusters it forms are most likely nonoptimal (if we assume an "optimal clustering" is even feasible for a problem). While this is usually acceptable, it should be kept in mind when analyzing the results.

A notable drawback when compared to the SOM is that the nodes do not form a 2D lattice. The main power of SOM comes from the powerful visualizations that can be derived from the lattice. Visualizing hierarchical tree structures is a lot more difficult task. On the other hand, since ETree's neighborhood is less rigid, it can achieve smaller quantization errors.

ETree also has quite a few parameters, such as the splitting threshold, gaussian widths and so on. Finding optimal values for these can be troublesome. Fortunately our experiments seem to have shown that, as is the case with SOM, most of the parameters have sensible default values that don't usually need fiddling.

7.4 Applications

ETree can be used in almost all the tasks that unsupervised learning methods are suited to. The most common unsupervised learning task is clustering, which ETree performs quite fast. ETree also forms a hierarchical description of data, which can be utilized in various ways. We now briefly describe some application areas.

7.4.1 Clustering

Given how the ETree algorithm works, clustering is probably the most suitable application. Details of clustering have been covered in Chapter 4. Whenever analysis methods are used, though, their user should have a grasp of how the algorithm works. This allows them to interpret the results more accurately.

ETree forms disjoint clusters, each of which has a number of data vectors. The final size depends on the training parameters, especially the decay constant γ. The smaller it is, the larger clusters ETree forms. The clusters are also probably not globally optimal for the reasons explained in Chapter 5. But since ETree is very fast, it is usually applied to very large scale problems, where approximate solutions are acceptable.

7.4.2 Data topology estimation

When data is clustered with ETree, we obtain not only the clusters, but their interrelations. The trunk nodes that connect leaf nodes together give us an estimate of the data topology. This can be also called hierarchical clustering, which was examined in more detail in section 4.3.

7.4.3 Data reduction

When data is clustered with ETree, the leaf nodes form a new, smaller data set, whose shape and properties are similar to the original data. We can now analyze this smaller data set. We can even utilize those methods that would not be able to handle the entire data set in a reasonable time. Other applications include lossy data compression and vector coding (Theodoridos and Koutroumbas, 1999).

7.4.4 Density estimation

Density estimation is closely related to data reduction. In it we replace each cluster with a probability distribution which is fitted to the data vectors that map into each cluster (e.g. Parzen kernels (Parzen, 1962)). There are several methods for this. Since each cluster only has a relatively small amount of vectors, the fitting is a lot easier than if we examined the whole data set at once. These

local estimates can then, for example, be summed to obtain the final estimate. It should be noted that depending on parameters ETree leaf nodes may contain only a few data vectors, so the individual estimates can be somewhat imprecise.

7.4.5 Approximate indexing

As has been established before, traditional database queries are exact, which makes them difficult to do efficiently. If we relax this requirement the problem becomes easier. Suppose we train an ETree and map all data vectors to its nodes. Now we can very quickly find an approximate answer. It should be noted that this can also be seen as assigning a hierarchical structure to the clusters that are formed at the leaf nodes.

7.5 Adaptation to data

Figure 7.2 shows how ETree adapts to the shape of the training data. The first image shows the uniform 2D distribution consisting of the words "Evolving Tree". We drew one thousand samples from this distribution. The following images show the locations of the leaf nodes as the training progresses. The first of those is at the beginning of the training when the tree has only a few leaf nodes. But as time goes on, the amount of nodes increases and they model the training data more and more precisely.

One might imagine that ETree's inability to always find the true BMU would cause some irregularities in the distribution of nodes. We can see in the last picture in Figure 7.2 that this is not the case at least for this data set. The leaf nodes spread out very evenly among the data. No area of the data is noticeably underrepresented, and conversely the areas of zero probability have very few spurious nodes.

This experiment verifies that ETree grows pretty much how one would intuitively expect. It also shows ETree's *data affinity*, that is, its tendency to only focus on those areas of the space that have data. Other methods, such as the SOM, often have nodes in the "void" areas between data clusters. Nevertheless, we should remember that this is a simplified two dimensional case. In higher dimensions the nodes have much more freedom to move around, and thus the results could be different. Unfortunately visualizing reliably anything that has more than three dimensions is extremely difficult.

7.6 Software package

To encourage the use of the ETree algorithm we have released our reference implementation under the GNU GPL (Free Software Foundation, 1991). It can be downloaded from `http://www.cis.hut.fi/research/etree/`. The package

Figure 7.2: Training data and the state of ETree leaf nodes during training.

consists of core ETree programs, which are coded in `C++` and an assortment of helper scripts in Python.

The package also includes extensive documentation including a tutorial and reference manual. There is also a fully browsable HTML documentation describing each class, file, and function in the source code. This transparency makes our implementation easy to examine and adapt to individual needs.

Chapter 8

Conclusions

In this thesis the starting point is the surface inspection problem. We have utilized content-based image retrieval tools and developed new neural computation methods for this difficult problem.

We have discovered that CBIR is a valid way to query huge databases that are quite common nowadays. CBIR methods give us efficient tools to manage, and analyze data. While we have only applied these tools to surface defect images, there is nothing that limits them to this particular area.

We have examined the PicSOM system, which has been designed as a general platform for content-based information retrieval. Despite this rather general approach, the system has been found quite suitable to this specific problem. PicSOM is especially good at weighting and combining several different features based on user feedback. Bringing a human being into the decision-making loop has traditionally been difficult, so PicSOM's good performance is a very desirable feature. This is the main practical contribution of this thesis.

The main theoretical contribution is without a doubt the Evolving Tree neural network, which was developed entirely by the author. ETree is an example of "power through simplicity". The algorithm description is extremely simple which makes it easy to understand. Still it performs quite well in the difficult problem of high dimensional data analysis, such as surface defect image clustering.

In the future it would be interesting to apply ETree to other areas, especially to those that benefit from the hierarchical structure that is automatically created during training. An example of this kind of area is bioinformatics, where tree based visualizations of gene expression data are very popular.

Ultimately the suitability of any algorithm is not decided on technical merits, elegance, implementation complexity, or convergence proofs. The true test for any method is whether or not it is being succesfully used to solve real world problems. We have found that having a freely available reference implementation greatly lowers the barriers for other people to test the algorithm. While there are no publications to refer to yet, we have learned through personal communication

that ETree has been used by third parties for such tasks as weather data analysis and robot vision. The availability of the package allows other people to more easily build on our work, which is one of the basic principles of science.

ETree will even be discussed in an upcoming book on neural networks (Samarasinghe, 2006). This will lead, we hope, to an entire new generation of students being exposed to the ideas that have been presented in this thesis.

One promising future research direction is fully integrating ETree with the PicSOM query engine. Since our experiments have shown that ETree seems to perform better than PicSOM's TS-SOM, this replacement should yield performance improvements. This has not been done yet, since PicSOM's combination power arises from the regularity of its SOM grid. Since ETree does not have a grid, this portion would have to be redesigned.

Bibliography

Addis, M., Boniface, M., Goodall, S., Grimwood, P., Kim, S., Lewis, P., Martinez, K., and Stevenson, A. (2003). Integrated image content and metadata search and retrieval across multiple databases. In *Proceedings of International Conference on Image Video Retrieval (CIVR 2003)*, pages 91–100, Urbana, IL, USA.

Alahakoon, D., Halgamuge, S., and Srinivasan, B. (2000). Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614.

Amarasiri, R., Alahakoon, D., Premaratne, M., and Smith, K. (2005). Enhancing clustering performance of feature maps using randomness. In *Proceedings of the 5th Workshop on Self-Organizing Maps*, pages 163–170, Paris, France.

Amarasiri, R., Alahakoon, D., and Smith, K. (2004). HDGSOM: A modified self-organizing map for high dimensional data clustering. In *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems*, pages 216–221. IEEE.

Aurenhammer, F. (1991). Voronoi diagrams — a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405.

Bach, J. R., Fuller, C., Gupta, A., Hampapur, A., Horowitz, B., Humphrey, R., Jain, R., and Shu, C.-F. (1996). The Virage image search engine: An open framework for image management. In Sethi, I. K. and Jain, R. J., editors, *Storage and Retrieval for Image and Video Databases IV*, volume 2670 of *Proceedings of SPIE*, pages 76–87. SPIE.

Barber, C. B., Dobkin, D. P., and Huhdanpää, H. (1996). The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483.

Bauer, H.-U. and Villmann, T. (1997). Growing a hypercubical output space in a self-organizing feature map. *IEEE Transactions on Neural Networks*, 8(2):218–226.

Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD international conference on management of data*, pages 322–331.

Behnke, S. and Karayiannis, N. (1998). Competitive neural trees for pattern classification. *IEEE Transactions on Neural Networks*, 9(6):1352–1369.

Bentley, J. (1979). Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering*, 5(4):333–340.

Berchtold, S., Keim, D. A., and Kriegel, H.-P. (1996). The X-tree: An index structure for high-dimensional data. In Vijayaraman, T. M., Buchmann, A. P., Mohan, C., and Sarda, N. L., editors, *Proceedings of the 22nd International Conference on Very Large Databases*, pages 28–39, San Francisco, U.S.A. Morgan Kaufmann Publishers.

Bernié, J.-P. and Douglas, W. J. M. (1996). Local grammage distribution and formation of paper by light transmission image analysis. *TAPPI Journal*, 79(1):193–202.

Bhandarkar, S. M., Koh, J., and Suk, M. (1997). Multiscale image segmentation using a hierarchical self-organizing map. *Neurocomputing*, 14(3):241–272.

Blackmore, J. and Miikkulainen, R. (1993). Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 450–455.

Bresee, R. B. and Paniluk, T. S. (1997). Characterizing nonwoven web structure using image analysis techniques. *TAPPI Journal*, 80(7):133–137.

Bruske, J. and Sommer, G. (1997). Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7(4):845–865.

Brzakovic, D. and Vujovic, N. (1996). Designing a defect classification system: A case study. *Pattern Recognition*, 29(8):1401–1419.

Burzewski, V. and Mohan, C. K. (1996). Hierarhical growing cell structures. In *Proceedings of the International Conference on Neural Networks*, pages 1658–1663, Washington D.C., USA. IEEE.

Campos, M. and Carpenter, G. (2001). S-TREE: self-organizing trees for data clustering and online vector quantization. *Neural Networks*, 14(4–5):505–525.

Canny, J. F. (1986). A computational approach to edge detection. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 8(6):679–698.

Carmack, J., Abrash, M., Romero, J., Taylor, D., and Radek, P. (1993). The Doom engine. id Software, `http://doom.wikia.com/wiki/Doom_rendering_engine`.

Castleman, K. R. (1995). *Digital Image Processing*. Prentice Hall.

Chaudhuri, B. B. and Sarkar, N. (1995). Texture segmentation using fractal dimension. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):72–77.

Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L. (2001). Searching in metric spaces. *ACM Computing Surveys*, 33(1):273–321.

Cox, I. J., Miller, M. L., Minka, T. P., Papathomas, T. V., and Yianilos, P. N. (2000). The bayesian image retrieval system, PicHunter: Theory, implementation and psychological experiments. *IEEE Transactions on Image Processing*, 9(1):20–37.

Csillaghy, A., Hinterberger, H., and Benz, A. O. (2000). Content-based retrieval in astronomy. *Information Retrieval*, 3(3):229–241.

Cuadros-Vargas, E. and Romero, R. A. F. (2005). Introduction to the SAM-SOM* and MAM-SOM* families. In *Proceedings of International Joint Conference on Neural Networks 2005*, pages 2966–2970, Montréal, Canada.

Date, C. J. (2003). *An Introduction to Database Systems*. Addison Wesley, eighth edition.

Davies, D. and Bouldin, D. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(4):224–227.

Davies, E. (1990). *Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, London, UK.

Del Bimbo, A. (1999). *Visual Information Retrieval*. Morgan Kaufmann Publishers, Inc.

Delgado, H. M. S. and Gomes, L. S. F. (1995). X-ray diffraction for quantifying calcium carbonate fillers in printing and writing papers. *TAPPI Journal*, 78(4):135–139.

Dittenbach, M., Rauber, A., and Merkl, D. (2001). Recent advances with the growing hierarchical self-organizing map. In *Proceedings of the 3rd Workshop on Self-Organizing Maps*, Advances in Self-Organizing Maps, pages 140–145, Lincoln, England. Springer.

Dittenbach, M., Rauber, A., and Pölzlbauer, G. (2005). Investigation of alternative strategies and quality measures for controlling the growth process of the growing hierarchical self-organizing map. In *Proceedings of International Joint Conference on Neural Networks 2005*, pages 2954–2959, Montréal, Canada.

Dunn, D., Higgins, W. E., and Wakeley, J. (1994). Texture segmentation using 2-d gabor elementary functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):130–149.

Dunn, J. C. (1974). Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4:95–104.

Edwards, P. J., Murray, A. F., Papadopoulos, G., Wallace, A. R., Barnard, J., and Smith, G. (1999). Paper curl prediction and control using neural networks. *TAPPI Journal*, 82(7):145–152.

Eidenberger, H. (2004). A video browsing application based on visual MPEG-7 descriptors and self-organising maps. *International Journal of Fuzzy Systems*, 6(3):125–138.

Everitt, B. S., Landau, S., and Leese, M. (2001). *Cluster Analysis*. A Hodder Arnold Publication, 4th edition.

Fleck, M., Forsyth, D., and Bregler, C. (1996). Finding naked people. In *Proceedings of the European Conference on Computer Vision*, volume II, pages 592–602.

Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., and Yanker, P. (1995). Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23–32.

Free Software Foundation (1991). The GNU general public license, version 2. `http://www.gnu.org/licenses/gpl.html`.

Friedman, J. H., Baskett, F., and Shustek, L. J. (1975). An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, C-24:1000–1006.

Fritzke, B. (1992). Kohonen feature maps and growing cell structures — a performance comparison. In *Advances in Neural Processing Systems*, volume 5, pages 123–130.

Fritzke, B. (1994). Growing cell structures — a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460.

Fritzke, B. (1995a). Growing grid — a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13.

Fritzke, B. (1995b). A growing neural gas network learns topologies. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA.

Fuchs, H., Kedem, Z., and Naylor, B. (1980). On visible surface generation by a priori tree structures. In *Proceedings of SIGGRAPH '80*, pages 124–133.

Gaede, V. and Günther, O. (1998). Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231.

Gonzalez, R. C. and Woods, R. E. (1992). *Digital Image Processing.* Addison-Wesley.

Graf, J. E., Enright, S. T., and Shapiro, S. I. (1995). Automated web inspection ensures highest quality nonwovens. *Tappi Journal*, 78(9):135–138.

Gudivada, V. N. and Raghavan, V. V. (1995). Content-based image retrieval systems. *IEEE Computer*, 28(9):18–21.

Gustafson, F. and Delgado, J. (1996). Determination of Post-It®note adhesive particle size in handsheets and filter paper by image analysis. *TAPPI Journal*, 79(7):127–134.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA. ACM Press.

Haralick, R., Shanmugam, K., and Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(6):610–621.

Haralick, R. M. and Shapiro, L. G. (1992). *Computer and Robot Vision*, volume 1. Addison-Wesley.

Haykin, S. (1994). *Neural Networks. A Comprehensive Foundation*. Macmillan College Publishing Company, Inc., New York.

Hodge, V. J. and Austin, J. (2001). Hierarchical growing cell structures: TreeGCS. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):207–218.

Hsu, W., Chua, T. S., and Pung, H. K. (1995). An integrated color-spatial approach to content-based image retrieval. In *Proceedings of 3rd International ACM Multimedia Conference*, pages 305–313, Seattle, Washington, USA.

Huang, T. S., Mehrotra, S., and Ramchandran, K. (1996). Multimedia analysis and retrieval system (MARS) project. In *Proceedings of 33rd annual clinic on library application on data processing — Digital image access and retrieval*, Urbana-Champaign, IL, USA.

Hubert, L. and Schultz, J. (1976). Quadratic assignment as a general data-analysis strategy. *British Journal of Mathematical and Statistical Psychologie*, 29:190–241.

Hung, C. and Wermter, S. (2003). A dynamic adaptive self-organising hybrid model for text clustering. In *Proceedings of The Third IEEE International Conference on Data Mining*, pages 75–82, Melbourne, USA.

Hung, C. and Wermter, S. (2005). A constructive and hierarchical self-organising model in a non-stationary environment. In *Proceedings of International Joint Conference on Neural Networks 2005*, pages 2948–2953, Montréal, Canada.

I'Anson, S. (1995). Identification of periodic marks in paper and board by image analysis using two-dimensional Fast Fourier Ttransforms. *TAPPI Journal*, 78(3):113–119.

Iivarinen, J., Heikkinen, K., Rauhamaa, J., Vuorimaa, P., and Visa, A. (2000). A defect detection scheme for web surface inspection. *International Journal of Pattern Recognition and Artificial Intelligence*, 14(6):735–755.

Iivarinen, J., Peura, M., Särelä, J., and Visa, A. (1997). Comparison of combined shape descriptors for irregular objects. In *Proceedings of the 8th British Machine Vision Conference*, volume 2, pages 430–439, University of Essex, UK.

Iivarinen, J. and Visa, A. (1998). An adaptive texture and shape based defect classification. In *Proceedings of the 14th International Conference on Pattern Recognition*, volume I, pages 117–122, Brisbane, Australia.

Inoue, H. and Narihisa, H. (2003). SONG: self-organizing neural grove. In *Proceedings of the workshop on Self-Organizing Maps '03*, pages 161–166, Kitakyushu, Japan.

Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323.

Johansson, B. (2000). A survey on: Contents based search in image databases. Technical report, Linköping University, Department of Electrical Engineering, http://www.isy.liu.se/cvl/Projects/VISIT-bjojo/.

Kiranyaz, S. (2005). *Advanced techniques for content-based management of multimedia databases*. PhD thesis, Tampere University of Technology.

Köhle, M. and Merkl, D. (1996). Visualizing similarities in high dimensional input spaces with a growing and splitting neural network. In van der Malsburg, C., von Seelen, W., Vorbrüggen, J. C., and Sendhoff, B., editors, *Proceedings of the International Conference on Artificial Neural Networks*, number 1112 in Lecture Notes in Computer Science, pages 581–586, Bochum, Germany.

Kohonen, T. (2001). *Self-Organizing Maps*. Springer, Berlin, 3. extended edition.

Koikkalainen, P. (1994). Progress with the tree-structured self-organizing map. In Cohn, A. G., editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 211–215, Amsterdam, The Netherlands.

Koikkalainen, P. and Oja, E. (1990). Self-organizing hierarchical feature maps. In *Proceedings of 1990 International Joint Conference on Neural Networks*, volume II, pages 279–284, San Diego, CA.

Kong, H. and Guan, L. (1998). Self-organizing tree map for eliminating impulse noise with random intensity distributions. *Journal of Electronic Imaging*, 7(1):36–44.

Koprinska, I. and Clark, J. (2004). Video summarization and browsing using growing cell structures. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2601–2606, Budapest, Hungary.

Koskela, M. (2003). *Interactive image retrieval using self-organizing maps*. PhD thesis, Helsinki University of Technology.

Koskela, M., Laaksonen, J., Sjöberg, M., and Muurinen, H. (2005). PicSOM experiments in TRECVID 2005. In *Proceedings of the TRECVID 2005 Workshop*, pages 267–270, Gaithersburg, MD, USA. NIST.

Kunttu, I. (2005). *Shape and Gray Level Descriptors for Surface Defect Image Retrieval and Classification*. PhD thesis, Tampere University of Technology.

Laaksonen, J., Koskela, M., Laakso, S., and Oja, E. (2000). PicSOM - content-based image retrieval with self-organizing maps. *Pattern Recognition Letters*, 21(13-14):1199–1207.

Laaksonen, J., Koskela, M., and Oja, E. (1999). PicSOM - a framework for content-based image database retrieval using self-organizing maps. In *Proceedings of the 11th Scandinavian Conference on Image Analysis*, pages 151–156, Kangerlussuaq, Greenland.

Lew, M., editor (2001). *Principles of Visual Information Retrieval*. Springer-Verlag.

Li, T., Tang, Y. Y., and Fang, L. Y. (1995). A structure-parameter-adaptive (spa) neural tree for the recognition of large character set. *Pattern Recognition*, 28(3):315–329.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley. University of California Press.

Mäenpää, T. (2003). *The local binary pattern approach to texture analysis - extensions and applications*. PhD thesis, University of Oulu.

Manjunath, B. S., Ohm, J.-R., Vasudevan, V. V., and Yamada, A. (2001). Color and texture descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):703–715.

Manjunath, B. S., Salembier, P., and Sikora, T., editors (2002). *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons Ltd.

Marr, D. and Hildreth, E. C. (1980). Theory of edge detection. In *Proceedings of the Royal Society of London*, volume B 270, pages 187–217.

Marshall, S. (1989). Review of shape coding techniques. *Image and Vision Computing*, 7(4):281–294.

Martinez, T. and Schulten, K. (1991). A "neural-gas" network learns topologies. In Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, volume 1, pages 397–402, Amsterdam.

Milosavjlevic, N. and Heikkilä, P. (1999). Modeling a scrubber using feed-forward neural networks. *TAPPI Journal*, 82(3):197–202.

Mital, D. P. and Leng, G. W. (1994). An autoregressive approach to surface texture analysis. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(4):845–857.

Moody, J. and Darken, C. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294.

Mott, L., Shaler, S. M., Liang, B.-H., and Groom, L. H. (1995). The tensile testing of individual wood fibers using environmental scanning electron microscopy and video image analysis. *TAPPI Journal*, 78(5):143–148.

Muneesawang, P. and Guan, L. (2002). Automatic machine interactions for content-based image retrieval using a self-organizing tree map architecture. *IEEE Transactions on Neural Networks*, 13(4):821–834.

Newman, T. S. and Jain, A. K. (1995). A survey of automated visual inspection. *Computer Vision and Image Understanding*, 61(2):231–262.

Niskanen, M. (2003). *A visual training based approach to surface inspection*. PhD thesis, Oulu University.

Ogle, V. E. and Stonebraker, M. (1995). Chabot: retrieval from a relational database of images. *IEEE Computer*, 28:40–48.

Ontrup, J. and Ritter, H. (2001). Hyperbolic self-organizing maps for semantic navigation. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14 (NIPS)*, volume 14, pages 1417–1424. MIT Press.

Ontrup, J. and Ritter, H. (2005). A hierarchically growing hyperbolic self-organizing map for rapid structuring of large data sets. In *Proceegins of the 5th Workshop on Self-Organizing Maps*, pages 471–478, Paris, France.

Pakkanen, J. and Iivarinen, J. (2003). Content-based retrieval of surface defect images with MPEG-7 descriptors. In Jr., K. W. T. and Meriaudeau, F., editors, *Proceedings of Sixth International Conference on Quality Control by Artificial Vision*, SPIE Vol. 5132, pages 201–208, Gatlinburg, Tennessee, USA.

Parzen, E. (1962). On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076.

Pauwels, E. and Frederix, G. (1999). Finding salient regions in images: non-parametric clustering for image segmentation and grouping. *Computer Vision and Image Understanding*, 75(1–2):73–85.

Pentland, A., Picard, R., and Sclaroff, S. (1994). Photobook: Content-based manipulation of image databases. In *Storage and Retrieval for Image and Video Databases*, volume II of *Proceedings of SPIE*, pages 34–47, San Jose, USA. SPIE.

Roberts, G. W. (1983). *Industrial Engineering #8: Quality Assurance in Research and Development*. Marcel Dekker.

Samarasinghe, S. (2006). *Neural Networks for Pattern Recognition in Scientific Data*. Auerbach Publications. Not yet published.

Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260.

Sellis, T., Roussopoulos, N., and Faloutsos, C. (1987). The R+-tree: a dynamic index for multi-dimensional objects. In *Proceedings of the thirteenth international conference on very large data bases*, pages 507–518.

Sethi, I. K. and Coman, I. (1999). Image retrieval using hierarchical self-organizing feature maps. *Pattern Recognition Letters*, 20(11–13):1337–1345.

Shyu, C.-R., Brodley, C. E., Kak, A. C., Kosaka, A., Aisen, A. M., and Broderick, L. S. (1999). ASSERT: A physician in-the-loop content-based retrieval system for HRCT image databases. *Computer Vision and Image Understanding*, 75(1/2):111–132.

Smith, J. R. and Chang, S.-F. (1996). VisualSEEk: A fully automated content-based image query system. In *Proceedings of the 4th international ACM multimedia conference*, pages 87–98, Boston, MA, USA.

Smith, T. R. (1996). A digital library for geographically referenced materials. *IEEE Computer*, 29(5):54–60.

Sneath, P. H. and Sokal, R. R. (1973). *Numerical Taxonomy*. Freeman, London, UK.

Song, H.-H. and Lee, S.-W. (1998). A self-organizing neural tree for large-set pattern classification. *IEEE Transactions on Neural Networks*, 9(3):369–379.

Sonka, M., Hlavac, V., and Boyle, R. (1999). *Image Processing, Analysis, and Machine Vision*. Brooks/Cole Publishing Company, second edition.

Squire, D. M., Müller, W., Müller, H., and Raki, J. (1999). Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. In *The 11th Scandinavian Conference on Image Analysis*, pages 143–149, Kangerlussuaq, Greenland.

Stricker, M. and Orengo, M. (1995). Similarity of color images. In *Storage and Retrieval for Image and Video Databases III (SPIE)*, volume 2420 of *SPIE Proceedings*, pages 381–392, San Jose, CA, USA. SPIE.

Swain, M. J. and Ballard, D. H. (1991). Color indexing. *International Journal of Computer Vision*, 7(1):11–32.

Theodoridos, S. and Koutroumbas, K. (1999). *Pattern recognition*. Academic Press.

Tyan, S. G. (1981). Median filtering, deterministic properties. In Huang, T. S., editor, *Two-Dimensional Digital Signal Processing*, volume II, pages 197–217. Springer Verlag, Berlin.

Vapnik, V. (2000). *The Nature of Statistical Learning Theory*. Springer, 2nd edition.

Viitaniemi, V. and Laaksonen, J. (2003). Content-based browsing of mail-order catalogue with PicSOM system. In *Proceedings of the 2003 Conference on Visual Information Systems (VIS'2003)*.

Ward, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of American Statistical Association*, 58(301):236–244.

Wen, W. X., Jennings, A., and Liu, H. (1992). Learning a neural tree. In *Proceedings of International Joint Conf. on Neural Networks*, volume 2, pages 751–756, Beijing, China.

Wu, S., Rahman, M. K. M., and Chow, T. W. S. (2005). Content-based image retrieval using growing hierarchical self-organizing quadtree map. *Pattern Recognition*, 38(5):707–722.

Yang, Z. and Laaksonen, J. (2005). Interactive retrieval in facial image database using self-organizing maps. In *Proceedings of IAPR Conference on Machine Vision Applications (MVA 2005)*, pages 112–115, Tsukuba Science City, Japan.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*, 8:338–353.

Zhang, B., Fu, M., and Yan, H. (1998). Handwritten digit recognition by neural 'gas' model and population decoding. In *Proceedings of International Joint Conference on Neural Networks*, volume 3, pages 1727–1731, Anchorage, Alaska, USA.

Zhang, H. and Zhong, D. (1995). Scheme for visual feature-based image indexing. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 36–46, San Jose, CA, USA.