Master Thesis
LiTH-ISY-3132

Computer Vision Classification of Leaves from Swedish Trees

Oskar J O Söderkvist

September 20, 2001

Abstract

The aim of this master thesis is to classify the tree class from an image of a leaf with a computer vision classification system. We compare different descriptors that will describe the leaves different features. We will also look at different classification models and combine them with the descriptors to build a system that could classify the different tree classes.

Acknowledgements

This thesis could not have been written without the support from a large number of people.
I am especially grateful to the following persons:

Klas Nordberg, for providing ideas and constructive criticism of this manuscript.

Björn Johansson, for all the discussions and trouble shooting.

Anders Moe, for providing ideas and being happy.

And the rest of the Computer Vision Laboratory, for providing a stimulating environement and sharing ideas.

Arne Anderberg and Cary Karp at Swedish Museum of Natural History for coming up with the idea of the master thesis and for the help about the ecology part of the master thesis.

This Master thesis project has been done using equipment donated to the Computer Vision Laboratory by Hewlett-Packard through their Art & Science philanthropy programme.

And finally to familly and friends for doing what you do best!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The principal goal of this master thesis project is to see if it is possible to build a computer vision classifier that classifies leaves and give the name of the tree, with only the image of the leaf as an input-value. The main part of the thesis has been directed towards finding suitable descriptors that describe a leaf with just a few values. But we also need some sort of adaptive classification system that can be thought as a system that takes the values from the descriptor as input values and tells us what leaf we have. The idea were borne at Swedish Museum of Natural History and they contacted the Computer Vision Laboratory at Linköping University and asked if it was possible to build a computer program that could do the leaf classification automatically with only a computer and a scanner. The master thesis was limited to Swedish trees because of the small number of tree classes, but also because most, if not all, can be classified by only looking at the leaf.

There are numerous ways to use a system like this, but there are two main directions. First of all, this system would for example help teachers at primary school when they want to teach children about nature. Normally we have one teacher helping the children with the identification of the leaves, and at the same time trying to explain the connection between different plants and how the ecological system works. With help from a computer vision classifier system, the teacher could do what he is best doing i.e. explain the ecological system, and let the computer do the recognition of the leaves. This is the main reason for Swedish Museum of Natural History interest in this project.

There is also another reason, and a far more complex problem. It would be a great achievement if we could build a computer vision classification system that could classify all tree classes in the world, only from their leaves. But since there are thousands of thousands of tree classes that would be a very difficult task, but it could be simplified if we accept an answer with, for example, the 20 most probable tree classes. But this is not the goal of this thesis and will only be handled as an offspring of the main goal.

The purpose of the thesis project is to clarify if it is possible to use computer vision when we want to identify the different Swedish tree classes from there leaves. There are around 50 different tree-sorts in the Swedish flora. This makes the classification a fairly simple task, providing a simple answer i.e. the name of the leaf, or the five most probable names.

An important question for the design of the system is, who will use the

system, and for what purpose? These may seem like irrelevant questions at a first glimpse, and they are if we are only interested to get a single name of a tree class with 100 percent accuracy. But machines are like humans, they make mistakes. There are a number of reasons for this, the most significant one in leaf classification is that leaves from the same type of tree do not look the same. The system should be built for a person without any knowledge of how a computer vision classification works and it should be able to distinguish between the approximately fifty different leaves that exist, in Sweden.

Disposition

The remaining part of this thesis is divided into the following eight chapters.

Chapter 2 is an overview of a few projects in computer vision that other universities have done. It also contains an overview of the system that begins when the leaf is scanned, and ends when the answer is presented. It also contains a section about possible applications.

Chapter 3 analyzes the leaf, its different features, and parts.

Chapter 4, presents different descriptors that give us different characteristics of the leaf. It also presents some preprocessing methods that could be useful. Chapter 5 presents the different learning structures that do the classification, and their advantages and disadvantages.

We also need to evaluate the precision, error that our system gives. What if we want to add a new possible outcome? Can we improve the final system after we have began to use it? Should we also use an expert system to improve the computer vision system? In chapter 6, we will look at the weakest parts and try to understand how they can be improved.

Chapter 7 will look into the system and try to evaluate and analyze the different parts.

In chapter 8 we will see were the next step towards the perfect system may be directed towards.

Chapter 2

Background

We will first study a few existing systems for classification of different objects. There have not been many projects particularly on leaves, so therefore we will study a broad range of other project on computer vision classification. We will look at which descriptors are used, how large their databases are, and other data that are relevant for a computer vision classification system on leaves. To compare and evaluate the systems we would need to use the same database and have two different systems that have the same objective. But we have different descriptors with different objects to classify and different recognition techniques for all the systems which makes it impossible to compare them, but instead we can study and learn how to optimize our own system by studying these systems that are presented in 2.1.

In the next section we will look at how the system will be used from the user to the computer classification system, and back from the computer vision classification system to the user, as described in 2.2.

In the end of this chapter, we will discuss what a system like this could be used to.

Name	*descriptors*	*builder*	*objects*	*system*
CLC	CSS	Univ. Cambridge	50	-
MARS	MFD	Univ. of Illinois	-	-
MIDS	Wavelets	Univ. of Washington	$\leq$ 10000	-
AMIS	-	Netherlands	-	-
SIID	Edge mapping	Univ. Brown	-	graph matching
ACRC	photos	Univ. Bristol	-	Radial Basis func.

Table 2.1: Some of the projects, and their main features.

2.1 Projects

We will study a few projects that could be of interests, for two reasons. First, we will use methods from some of them and second because it is useful to see were the computer vision area is moving and realize where the boundaries are.

All of the specific methods that we will use and which are mentioned here, will be described more in detail in chapter 4.

Chrysanthemum Leaf Classification

The CLC project is collaboration between the National Institute for Agricultural Botany in Cambridge and the University of Surrey. Their leaf database consists of 400 leaf images from 40 sorts of Chrysanthemum-leaves. Each image is processed to recover the leaf contour, which is represented by the maximum of curvature zero-crossing contours in its CSS (Curvature Scale Space) image as well as three global shape parameters (eccentricity, circularity, and aspect ratio of its CSS image). When comparing two CSS images, the main idea is to find the optimal horizontal shift that results in the best possible overlap of maximums from two images. The sum of the Euclidean distances (in CSS) between the corresponding pairs of maximums is then defined to be the matching value.

The system selects the best 15 similar images from the database by first applying the global shape parameters to prune the candidates, followed by CSS matching. These images are, in general, from different varieties. The best 5 varieties are then selected according to the number of samples among the retrieved images. The correct variety was among the top 5 choices of the system for over 95% of the inputs. The goal of the system, is to be used as a complement to the expert classifier who will then make the final choice. For further reading, see [1].

Multiresolution Image Database Search

This is a computer vision classifier at Washington University that uses the Haar wavelet transform to identify different color patterns in the picture. In this paradigm, the user expresses a query to the database either by painting a crude picture or by showing an example of the image to a video camera or a scanner. Their algorithm considers the basic shape and color information of the query when looking through the database for potential matches. The query image is typically different from the target image, so the retrieval method must allow for some distortions. If the query is generated from a scanned image, it may suffer from artifacts such as color shift, poor resolution, dithering effects and misregistration. If the query is a painted image, it is limited by perceptual error in both shape and color. The aim is to make the retrieval fast enough to handle databases with thousands of images at interactive rates. The classifier chooses the 4-60 (specified by the user) most significant parameters of the (WT) Wavelet Transform, and match them with a special error function for this purpose, and takes the most likely pictures for further examination. For further reading, see [16].

The MARS-project

The Multimedia Analysis and Retrieval System (MARS) from University of Illinois is a system for content-based searching and browsing of large-scale multimedia repositories. MARS represents the content of images using visual features like color, texture and shape along with textual descriptions. The similarity be-

tween two images is defined as a combination of their similarities based on the individual features.

One of the most important features that represent the visual content of an image is the shape of the object(s) in an image. It uses a novel adaptive resolution (AR) representation of 2-d shapes, and maps each shape, represented by an AR, to a point in a high dimensional space that can be indexed using a multi-dimensional index structure. A distance measure for shapes is defined and it is claimed that similarity queries, based on the distance measure, can be executed efficiently using the index structure. The experimental results demonstrate the effectiveness of their approach to the fixed resolution (FR) technique previously proposed in the literature. One of the descriptors that they have used is the MFD that we will study further in chapter 4.9 . For further reading, see [15]

The AMIS project

The AMIS project is a national research initiative in the Netherlands to broaden and deepen the understanding of methods for indexing and searching multimedia databases. In this project, several disciplines of computer science collaborate to achieve this one research goal of advancing insight into critical bottlenecks of multimedia technology. Key elements here are the proper and anticipating organization of the data in the database in order to answer any question quickly. In this respect, multimedia databases are distinctly different from the common category databases. The performance at this level is measured by response time. The second level deals with formal descriptions of the semantic content as the basis for explicit querying, as well as the expression of the domain invariant descriptions, enabling querying by example. At this level, knowledge of the search domain is integrated methods, such as transforms . This is the level where the performance is measured in terms of precision and recall.

At the third level, the system tries to be as fast as possible. The delivery of data 'just in time' plays a key role. Here, guaranteed low latencies of data rendered on user displays play a key role. This is the level where performance is measured in terms of perceived 'quality of service' at display.

A successful solution of the research issues requires a better understanding of the interplay between multimedia data organization. Leading groups in The Netherlands have been brought together to undertake a research program to address these issues within the context of a focused application domain: a database of pictures. The goal of the AMIS project is to provide efficient content-bases indexing and retrieval. They have divided the project into five major groups, or sub-projects.

- Scalable database architecture for image storage and retrieval
- Query formulation, effectiveness, and indexing
- Image indexes based on color and shape invariant features
- Shape oriented data types
- Operating system support and storage infrastructure

To succeed they believe that three major mechanisms must be provided:

- Compact organization and fast access of very large amounts of multimedia data from tertiary storage
- The provision of concise, expressive and invariant image features to generate discriminatory image indices for fast image search
- Database technology supporting storage and management of the images and their associated indices.

To read more about this project, see [4].

SIID

This project has its base at the Brown University in New England. The main goal is to create an image retrieval system that is primarily based on shape. The use of shape as a cue for indexing into pictorial databases has been traditionally based on global invariant statistics and deformable templates, on the one hand, and local edge correlation on the other. They have proposed an intermediate approach based on a characterization of the symmetry in edge maps. They have used graph matching, among other methods, when they built their system. To read more about this project, see [22].

ACRC

This joint venture between the ACRC Vision Group at Bristol University and Hewlett Packard aims to develop a computer vision system that is capable searching digital image databases for user-defined objects. Databases of interest would include those found on the World Wide Web as well as digital home photograph albums such as Photo-CDs. Such databases can often be too large to allow every image to be inspected visually, and automatic techniques are required to assist with searching and interpretation. To read more about this project, see [2].

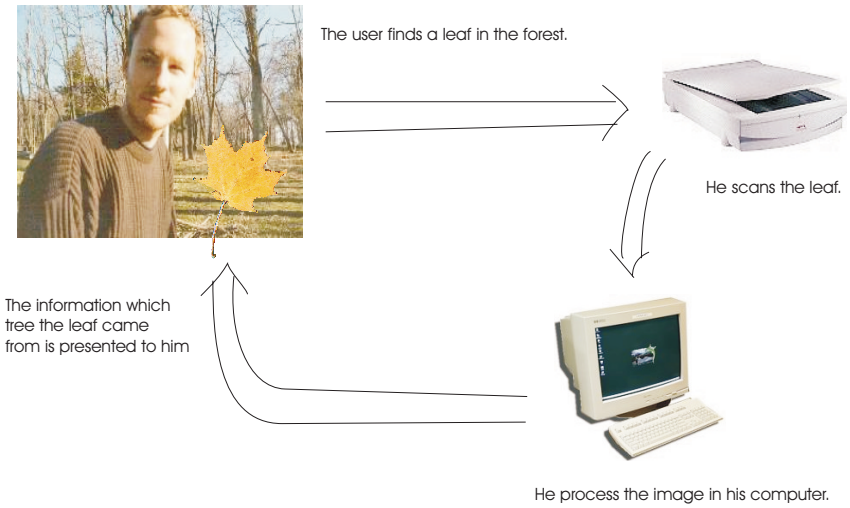

Figure 2.1: This is a picture that show the system.

2.2 Overview of the system

The main idea behind this master thesis is to build a computer vision system of tree classes that do the classification from an image of a leaf. The system, in figure 2.1 starts when someone picks a leaf, and ends when an answer is presented about what sort of tree the leaf was taken from. We will try to divide the full system into the following parts, which will be further discussed in separate sections.

- The user scans the leaf
- The information is transferred from the scanner to the computer vision classifier
- The data from the image of the leaf is processed in the computer vision classifier
- The result from the computer vision classifier is presented to the user

Scanning of the leaf

When we scan the leaf we may use different scanners with different adjustments and in different environments, and we may scan leaves with different transparency and color, which in turn depend on the weather, the season, and the time the leaf has been separated from the tree. All this together makes the scanning vulnerable and it is therefore important that the user is careful when scanning the leaf. For example, the leaf should be as clean and normal built as possible. But it is also important that the user knows the limitations and possibilities of the systems as a whole so that certain difficulties can be avoided and certain obstacles that the system can handle can be ignored. For example, if we use the inner structure of the leaf for one of the descriptors and there is

a hole in the chosen leaf, that will make the classification unreliable. Therefore the user should use another leaf to be sure that the values from the descriptors are sufficiently reliable.

Transfer the image to the computer

When we transfer the data to the computer we could just save it on a floppy disc and then walk to the computer that contains our computer vision program, or we could send it directly from the scanner where we are to the computer. This is not a problem and is more depending on what computer and scanner that we use and how they are connected.

Classification

The computer vision classifier starts with the extraction of the descriptors from the image of the leaf, for example the eccentricity, perimeter, area etc. These descriptors will be input values into a classification part of the system, that we train to determine what leaf we have.

Presenting the result to the user

When the classification is completed, the information that produces will be presented to the user. The information should be presented in a form that the user feels is useful and understandable. It could turn up directly on the screen or in some other form that is suitable for the user.

2.3 Applications

This could be neglected as a very easy question to answer, but instead of just saying that a system that automatically classifies trees from their leaves is useful to those who want to classify tree classes we will exemplify were this system could be valuable and useful. What probably come to your mind first are primary schools. A system with the Swedish leaves could be used for schools when pupils are at excursion, usually with approximately 30 children to every teacher. The ideal situation is when every pupil has a hand scanner and a handheld computer. The pupils can get the answers directly if the system is on-line or they can save all the images and analyze them when they return back to the school. It is not hard to imagine that not only pupils from primary school can benefit from a system like this but also ordinary interested people who are not experts in biology but feel that easier to use this system than getting the same information from a book. Also, we may have an expert out on the field and who may come up with a leaf of uncertain origin. If we expanded the database with all sorts of tree classes, we could use it as a database and help experts too. Especially if the ten most likely leaves can be presented and then let the expert decide what leaf it is by looking at the tree, or other things that is close to the tree.

The system could be used by everyone from school classes to advanced scientists depending on the size of the program, size of the database, number of descriptors, time that the program is allowed to work, etc. The only limitations that exist, is the fantasy that we have!

Chapter 3

The Leaf

We will start to divide the leaf features into four main groups. In the first group we have the shape of the leaf. Then we have the interior structure or bone-structure, the color and finally the surface.

3.1 The Shape

There are a large variety of leaf shapes and they are all highly connected to the tree class they belong to, and this feature is also very easy to obtain from a scanned picture. This makes it a very useful feature, as we can see in [10], [9], [19], and [3] where tree classification is done mainly from the shape of their leaves. This is a good feature as we have said above but, when the leaf contains of more than one part, the different part have a tendency to overlap each other, as can be seen in figure 3.1. This overlapping varies in a vast number of ways, depending in what way we put the leaf into the scanner. Therefor may leaves with more than one part be more difficult to scan. We should also note that if we scan a leaf with only one part the leaf will never be almost plane and this may cause problems for the classification.

Figure 3.1: In this figure we can see a leaf that is built with more than one part parts.

Figure 3.2: In this figure we can clearly see the interior structure inside the leaf.

Figure 3.3: In this figure we can see that the leaves color may vary, even for a single leaf.

3.2 The Inner structure

The inner structure, or bone structure, normally has a main link from the shaft to the top of the leaf, but there are varieties even here. We see an illustrated example of the Innerstructure in figure 3.2. This makes this feature is rather difficult to obtain because it is very dependent on which group of leaf we have and it is therefore rather difficult to find a general rule to obtain it. If we could it would be useful when we want to classify a leaf as we can see in most books written to classifies tree classes for example in [10] and [19]. A definition of the Inner structure : The Inner structure is that structure that could not be obtained from any other part than the whole leaf.

3.3 The Color

The task to decide the color of the leaf is fairly easy but to use this information to separate different leaves is a complex task. A leaf changes color between different seasons and also has different color depending on where the tree is positioned and if the tree faces the sun all day or stands in a shadowed place. The color also depends on how old the tree is and where the leaf is positioned on the tree. These are just a few examples that demonstrate that the color of a leaf varies both between different sorts of trees, but also within the same sort. It is still possible to draw some conclusions, with an empirical proof that the color is important when we use leaves to distinguish between different trees. We can just think how we do ourselves when we distinguish different leaf, for example, a Rhododendron leaf from an apple tree leaf. We can also, as in 3.3

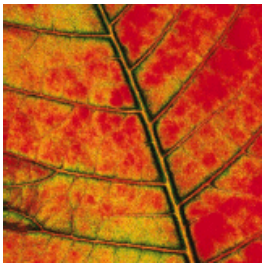

Figure 3.4: In this figure we can see parts of the interior structure but also the glossiness, the hairiness and other small details that we will define as the surface.

that a single leaf may have different colors.

3.4 The Surface

We will use this section to present all the different characteristics that are not already described in the previous sections, that could be viewed in 3.4. One is hairiness that many leaves do not have but a fair number do, which definitely makes this feature useful. Another characteristic that could be included in the surface is how glossy a leaf is. Some leaves are diffusive reflectors of light while other are not. One definition that we could do of the surface is: The part of the surface that is the same for all parts of the leaf. This is opposite to the inner structure because of the need to the whole leaf.

3.5 Conclusion

We have now studied the four main classes of features of the leaf and their advantages and disadvantages. The shape is easy to obtain, and also a good feature. The innerstructure is not a important descriptor in general but it is possible to group certain tree classes using the inner structure. The surface could be a helping hand when we have similar leaves which is also true for the color if we have additional information such as date and location of the tree. We will focus the analyze on the shape because it is the most stable and general feature but we will also try to obtain a good description of the inner structure of the leaf.

Chapter 4

Descriptors, and preprocessing procedures

4.1 Introduction

In order to succeed in our task to build a system that classify leaves we should build descriptors for certain features from the leaf. The ideal descriptor would be one that always has the same value for leaves that belongs to the same class of tree, and always has distinct values for leaves of different classes of trees. Unfortunately, it is rather difficult to find a descriptor like that, but there is another way. If we can find a set of descriptors that together separate all the tree classes that would give us the same final result.
Or more easily, if we have one descriptor that gives the value 5 for class A and class B, and value 3 for class C . And another descriptor that gives us 1000 for class A and class C, and 300 for class B. Then would it be very easy, even with out a computer to see that if we get the values 5 and 1000 from the descriptors, we have class A.
We could see this as a tree with nodes, where there are one descriptor in every nod. Every nod decides which next nod we should pass on to, and if we follow through we will end up at the end where there are, hopefully, only one answer left if not we need more descriptors in our tree graph.
In this chapter will we study different descriptors and try to understand if they are useful and how we could use them. But first should we study the way that we should pre-process the image before we could implement our descriptors.

4.2 Image adjustments and Preprocessing methods

Different descriptors need different Image adjustments and preprocessing procedures we will therefore study different methods. First we have the color image of the leaf, the next is the gray level image, and the third is the black and white image. The last one, is the sample on the border of the leaf. These methods that have been mentioned above, are just different ways of representing the image, and we will therefore call them image adjustments. We will mention some

Figure 4.1: This is grayscale image from the RGB-image of a leaf.

Figure 4.2: This is the black and white image of the leaf.

methods that change the image content and call the preprocessing methods. Symmetry filters will be mentioned but will not be implemented in the classification system. A method of how to find the leaf in the image will also be described. These are just short descriptions, for further reading [21]

4.2.1 Grayscale

To obtain the grayscale image we calculate the length of the RGB-vector for each position in the image. The result is what we can see in figure 4.1

4.2.2 Black and white image

There are many ways of producing a black and white image from a gray scale image. Typically we start with the gray scale image and we choose a threshold level. All the points above the threshold point will be white and all below will be black. This type of image is referred to as a black and white image or simply, B/W-image. It is normally not an easy task to choose the threshold level, but in this application there is a white background and a leaf that is rather dark which makes it possible to chose a threshold level within the range 0.4-0.8. This

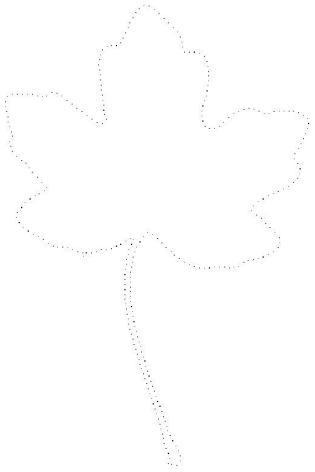

Figure 4.3: Sample points from the shape of a leaf

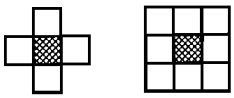

Figure 4.4: To the left we have 4-connectivity. We have the pixel in the middle and the pixels it is connected to with the distance one. To the right we have 8-connectivity. With the pixel in question in the middle and the pixel it is connected to around it.

gives us a good final result as we can see in figure 4.2. More advanced, and sophisticated techniques for producing B/W-images are described in [11].

4.2.3 Sample points from the shape of the leaf

Sample points from the shape of the B/W-image are easy to obtain by going around the edge of the B/W picture and that is showed in figure 4.3. We take the image, and calculate the number of points on the edge, inside within the leaf and decide how many sample points we want and find the distance between the points. If we want 200 sample points and we have 800 points as a total from the edge of the B/W image , we take each fourth point on the border of the B/W image and call them our sample points. One thing that we need to decide is the connectivity. The connectivity is how we chose which points are connected and not connected. 4-Connectivity is when only the points to the left, right, up and down are connected with the distance one. 8-Connectivity is the points in 4-connectivity connected, but also the points to left-up, left-down, right-up and right-down are connected with the distance one as the figure 4.4.

4.2.4 Rotational symmtryfilters

This is a set of filters that could be applied to the gray image to find different type of local orientation within the leaf. For further reading, see [13].

4.2.5 Identification of the leaf

In our scanned image of the leaf will there be other objects too, such as dust particles, dirt and other objects. Therefore we need to isolate the leaf and erase all other objects.
If we presume that the leaf will be the largest object in the picture we can measure the size of all objects in the image and then pick the largest object.

From the B/W-image we numerate all objects by giving all the pixels in every object the same number but different numbers for all the objects, the background pixels will get zero as a number. Then we count all the numbers and the biggest object will have the largest quantity of numbers so the only thing we need to do is to keep that object. For further reading, see [11].

4.3 Descriptors

We will here do a small survey of the descriptors that could be useful and descriptive when we want to classify the leaves. In table 4.1 we have some of the basic data about each descriptor.

Name	In picture	Nr. of output values	
Curvature Scale Space	Samples	3	
Incremental Circle-transform	Samples	2	
Circularity	B/W	1	
Area	B/W	1	
Modified Fourier descriptors	Samples	1	
Hu-Moments	B/W	7	
Flusser-moment	B/W	4	
Eccentricity	B/W	1	

Table 4.1: A summary of the descriptors and their main properties

4.4 Circularity

Introduction

This descriptor measures the ratio between the perimeter and the area of an object, and it defined according to the following equation.

$$circularity = \frac{perimeter^2}{Area} \tag{4.1}$$

A somewhat unstable parameter because the area is stable and the perimeter is unstable depending on the rotation, which makes the formula rather unstable. For further study [21].

Different changes and parameters that could be changed

The obvious parameter is the metric. We can chose between 8connectivity and 4connectivity, as are described in 4.4, but we also have honeycomb and other metrics that could be used depending on the object. For further study about different metrics, see [11].

4.5 The Curvature Scale Space method

The Curvature Scale Space method (CSS) that is described in [17] uses the shape of the object to distinguish different objects. The method uses the maximums of curvature zero crossing contours of Curvature Scale Space image as a feature vector to represent the shapes of object boundary contours. We can see the CSS curve in figure 4.5 were the method uses the maximas to classify objects. The method is robust to noise and variations in scale and orientation of the object and is therefore suitable for our task. We chose the CSS model from [17] to explain the basic idea behind the method.

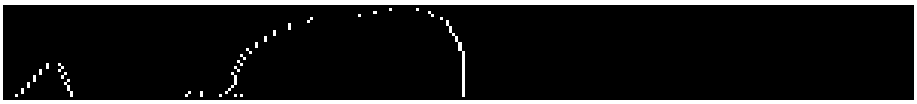

Figure 4.5: This is the curvature from the scale-space function with position in the x-axis and $\boldsymbol{\sigma}$ in the y-axis

Introduction

It is possible to divide the method into three parts.
In the first part we extract the sample values from the interior boundary of the B/W image.
In the second part we calculate the curvature and use curve evolution, by using the Gaussian kernel, to calculate the curvature. Curve evolution is when we use different Gaussin kernels to get the derivative when we want to calculate the derivative for the curvature formula. We then calculate the zero crossing of the curvature.
The third and last part, is the matching. This part of this specific method wills we not use and will therefore not be presented here. We will instead use the maximas as in values in the classification part.

The Curvature

The planar curve, that we use, is a set of points whose position vectors are the values of a continuos, vector valued function. We will represent it like:

$$r(u) = (x(u), y(u)) \tag{4.2}$$

where r(u) is the representation of the curve where the x and y are the coordinates for the position u in the sample vector r. Curvature has many desirable computational and perceptual characteristics.

The definition is:

$$\kappa(u) = \frac{y'(u)x''(u) - x'(u)y''(u)}{(x'(u)^2 + y'(u)^2)^{\frac{3}{2}}} \tag{4.3}$$

The curvature can define a planar curve, but together with torsion defines the two of them a curve.

We will look at, a short description, how to obtain the curvature theoretically. If you want a more detailed description, see [12] [1][18] [25]. We can now look at the tangent of the curve,

$$t(u) = \frac{r'(u)}{|r'(u)|} = (\frac{x'}{\sqrt{x'^2 + y'^2}}, \frac{y'}{\sqrt{x'^2 + y'^2}}) \tag{4.4}$$

and the normal to the same curve,

$$n(u) = (\frac{-y'}{\sqrt{x'^2 + y'^2}}, \frac{x'}{\sqrt{x'^2 + y'^2}}) \tag{4.5}$$

These equations must satisfy the simplified Serret-Frenet vector equation that is defined in [1].

$$t' = \kappa n \tag{4.6}$$

$$n' = -\kappa t \tag{4.7}$$

where the curvature is defined as:

$$\kappa = \lim_{h \to 0} \frac{angle(t(u), t(u+h))}{h} \tag{4.8}$$

And where the denominator is the angle between two parts of t(u). This leads us, together with the chain rule to:

$$\frac{d}{du}t = \frac{dr}{du}\kappa n = |r'|\kappa n \tag{4.9}$$

and we can now see that this gives us the formula (4.3).

Gauss kernels and the derivative

We will use gauss kernels to estimate the derivatives as have been described in (4.3). Below is a short description how it is done.

$$X_u(u, \tau) = \frac{d}{du}(x(u) * g(u, \tau)) = x(u) * g_u(u, \tau) \tag{4.10}$$

$$Y_u(u,\tau) = \frac{d}{du}(y(u) * g(u,\tau)) = y(u) * g_u(u,\tau) \tag{4.11}$$

Where $*$ is the convolution sign.

$$\kappa(u,\tau) = \frac{X_u(u,\tau)Y_u u(u,\tau) - X_u u(u,\tau)Y_u(u,\tau)}{(X_u(u,\tau)^2 + Y_u(u,\tau)^{2)}1.5} \tag{4.12}$$

The Method

First we get the sample points from the image.
With the samplepoints we calculate the curvature which we do with use of zero crossing.
To calculate the curvature with zero crossing we chose σ, that is we see when the curve is cutting the x-axel, i.e. when the curve is zero and uses that as a position in the Curvature Scale Space (CSS). After we have found all zero crossings, we add one to the σ and calculate the curvature and the zero crossing all over again for that σ. We continue until there are no zero crossing. We now create a CSS as the showed in 4.5 with the σ values as the y-axel and the different positions on the x-axel. Then we find the position of all local maximums in the CSS. The descriptors for this algorithm are the local maximums of the CSS.

Changes and adaptations

There are a few changes that could be made. First we have the number of samples per object which could be chosen differently. This must be done carefully because of the different sizes the objects have.

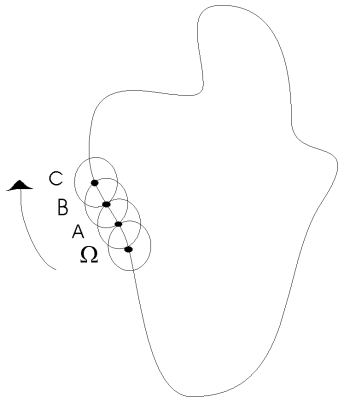

Figure 4.6: This is an illustration how to calculate the circletransform. First we map a circle A, save the values from that point. Then we map another circle, B, on the border of the first circle and save the values from that point. Then we take the difference between the center coordinates from the circle A and B. We save this value in the vector $\Delta\alpha$. The we start all over again and create a new circle C on the border of circle B. We continue until we have come back to circle A and do the difference between circle A and circle Ω. And know we have the full $\Delta\alpha$ or as we call it, the ICT

4.6 The Incremental Circle Transform, with eigenvalue analyze

Introduction

The method that will be described below will be divided into two stages.
In the first stage, the shape of an object is represented by the use of Incremental Circle Transform, which maps the boundary of an object into a circle, as showed in 4.6.
In the second stage, the eigenvalues of the variance and covariance matrix of the computed ICT-vector is calculated and they are used to represent the object. The method is both rotational and translation invariant. For a more detailed description, see [14].

The ICT algorithm

We will first describe the main equation for the ICT. It maps every point on the boundary curve of our object onto circle with radius r, in the x-y plane. And the result will be a 2 * n- matrix, where n depends on how large circle we decides to choose and how large object we have.

First, let us assume that we want to map the curve $\alpha(l)$, which is the border curve of our object(in our case a leaf). The $\Delta\alpha(l)$ is sometimes said to be the ICT of $\alpha(l)$ for $l \in [0, L]$ with the distance $0 \leq \Delta l \leq L$.

We start with the curve from the border and with the two constraints below fulfilled. We have to chose the size of r to decide which size the circle will have.

$$\Delta x^2(l) + \Delta y^2(l) = r^2 \tag{4.13}$$

$$\alpha(l + \Delta l) = \alpha(l) + \Delta\alpha(l) \tag{4.14}$$

$$\Delta\alpha(l) = (\Delta x(l), \Delta y(l)) \tag{4.15}$$

And when we have 'walked' around the α -curve then we will have the ICT transform of the object as showed in 4.6.

The Combined Algorithm

This is the three steps that are needed to uses this algorithm.

- Calculate the ICT
- Construct the variance-matrix of the ICT
- Calculate the eigenvalues of the variance matrix

and then use the eigenvalues as descriptors.

Adjustments

We could extract some other value instead of the eigenvalues. We also have to chose r, that is the radius of the circle that we map our object.

4.7 Template matching

Introduction

We have our picture $\boldsymbol{\Omega}$ that we want to classify and we have some pictures, $\boldsymbol{\Theta_i}$ that we want to match with our image $\boldsymbol{\Omega}$.

The Algorithm

We will here describe an algorithm that compares two black and white images with each other. But we start of with a gray scale image $\boldsymbol{\Omega}$. The first thing that we do is to convert the size to some standard size, for example 50 × 50. Then we transform $\boldsymbol{\Omega}$ to a B/W picture. Then we use the formula in 4.24 to know how we should rotate the picture. Now we want to compare $\boldsymbol{\Omega}$ with $\boldsymbol{\Theta_i}$. The pictures are here B/W pictures were white is represented with 1 and black is represented with 0. We do this with the operation,

$$R_i = XOR(\Omega, \Theta) = Inv(\Omega) \times \Theta_i + \Omega \times Inv(\Theta_i) \tag{4.16}$$

Where the function Inv() invert the image.
We compare all images, i, with our image and picks the one that is most similar to our own image and that is the one with the lowest $\boldsymbol{R_i}$ value.

Changes

This was just an example of how we could do Template matching. There is a variety of different implementation.

4.8 The Wavelet-transform

Introduction

This is one of the newest areas in transform theory and it is very interesting because of its combination of the spatial and frequency domains. We will not describe this method in detail but we believe it is an interesting method because of what it can extract from the image. It is also interesting if we want to use a larger image library because the method is rather fast. We will use the method described in [8]. The method can be divided into a few steps:

- Perform Haar-wavelet decomposition of the image.
- Store the overall average color, find the wavelet coefficients for all indices and the signs m largest-magnitude wavelet coefficients.
- Organized the indices into a single data structure in the program that optimizes searching.
- Perform the same wavelet decomposition for each query.
- The score for each target image is then computed by the evaluation expression, explained below.

The Evaluation Expression

This is the distance expression of the method. Q and T are the Wavelet parameters of the image that we want to identify and the image that we want to compare with. And $\tilde{Q}$ and $\tilde{Q}$ are the truncated, quantified wavelet coefficients of Q an T. These values are either -1, 0, or +1.

$$\|Q,T\| = w_{0,0}|Q[0,0] - T[0,0]| + \sum_{i,j} w_{i,j}|\tilde{Q}[i,j] - \tilde{T}[i,j]| \tag{4.17}$$

Where do we go from here?

There are a lot of changes that we could do. The first and probably the most obvious change would be to change the mother wavelet. We could save a different number of parameters for the Evaluation Expression. We could preprocess the image, using for example symmetry filters to get the inner structure of the leaf. There are many changes that could be made but we will not study them in this paper any further.

4.9 Modified Fourier Descriptors

Introduction

This is a method that uses Fourier descriptors to describe the object. It is described in [26]. This method is invariant to translation, rotation, and scale.

The Algorithm

We have the shape boundary represented as a complex valued function.

$$z(n) = x(n) + jy(n) \quad , \quad n = [0, N_B - 1] \tag{4.18}$$

Where x and y are sample values, and N_B the number of samples. The Discrete Fourier Transform of *z(n)* is:

$$Z(k) = \sum_{n=0}^{N_B-1} z(n) e^{-j\frac{2\pi nk}{N_B}} = M(k) e^{j\eta(k)} \quad , \quad k = [0, N_B - 1] \quad , \quad M(k) \in \Re \tag{4.19}$$

where we see that for $k \neq 0$, is *Z(k)* invariant to translation. If we rotate, translate and scale z(n) we get $z'(n)$. It is related to z(n) according to:

$$z'(n) = \alpha z(n - l) e^{j\phi} \tag{4.20}$$

It can be shoved that their DFT's are related to each other as:

$$M'(k) = \alpha M(k) \tag{4.21}$$

$$\theta'(k) = \phi + \theta(k) - \frac{2\pi lk}{N_B} \tag{4.22}$$

Where θ is the rotation angle of the object.

$$\phi = \theta - \theta_0 \tag{4.23}$$

We can calculate the last variable by using central moments, μ from 4.32:

$$\theta_0 = \frac{1}{2} \arctan \frac{2\mu_{11}}{\mu_{20} - \mu_{02}} \tag{4.24}$$

Based on this we construct two sequences:

$$ratio(k) = \frac{M'(k)}{M(k)} \tag{4.25}$$

$$shift(k) = \frac{\theta(k) - \theta'(k) + \phi}{k} \tag{4.26}$$

If the two sequences, M and M' are equal, then the function ratio, 4.25 must be constant, otherwise will we have a more or less high variation. Therefore will we use the distance measure,

$$D_m = \sigma[ratio] \tag{4.27}$$

$$D_p = \sigma[shift] \tag{4.28}$$

where the σ is the standard deviation and the overall distance measure is,

$$D = \psi_m D_m + \psi_p D_p \tag{4.29}$$

ψ_m and ψ_p are constants, deciding the ratio between 4.27 and 4.28.

Changes and adaptations

The only thing that is needed to do, is to choose the two constants in the matching equations, ψ_m and ψ_p for this MFD method. There are many other methods using Fourier descriptors which leads us to conclude that here are many changes possible to do, and still maintain the basic concept!

4.10 Area

Description

This is the simplest descriptor that will be presented here. But it is also one of the most useful descriptors. It is very robust and fast to calculate; we just have to count all the pixels that belong to the leaf.

$$Area = \sum_{i,j=Allpositions} f(i,j) \tag{4.30}$$

where f (i,j) is the value of the image, in the position height = i and width = j.

4.11 Moments

Introduction

These are the same moments as those used in mechanics, maybe with the exception that we use moments of higher orders to achieve invariance with respect of size, orientation, etc. We calculate the moments on the B/W image that we obtain as described above. Something that also is worth mentioning is that the functions that we study here are just a few of the vast number of descriptors that uses moments. The reason why these have been chosen is that they are well studied and they are invariant to position, rotation and size. For further reading, see [24].

The Algorithm

The basic algorithm for computing a moment of order (p, q) is,

$$m_{pq} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} i^p j^q f(i,j) \tag{4.31}$$

where f(,j) is the value of the image in position height=i and width=j. If the image is a B/W-image will *f(i,j)* be either one or zero. In most cases are the central moments are used. It is the same equations except that the origin of the coordinate system is translated to the center of gravity for both x_c and y_c. The definition for the central moments is:

$$\mu_{pq} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} (i - x_c)^p (j - y_c)^q f(i,j) \tag{4.32}$$

where $x_c = \frac{m_{10}}{m00}$ and $y_c = \frac{m_{01}}{m00}$.

There are a large number of descriptors that use moments. We will try to cover the most useful descriptors for Pattern recognition of leaves. The following is a presentation of moment based descriptors that are useful for recognition of leaves.

4.11.1 Hu's Descriptors

Hu [24] has derived seven descriptors that are invariant with respect to rotation, translation, and scale. The definition of these seven is as follow. As we see are the descriptors based on central moments that are normalized with respect to the μ_{00} that is the same as the area.

$$\vartheta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{\gamma}} \tag{4.33}$$

where $\boldsymbol{\gamma}$ is normally given the value 1. The descriptors ϕ_1 to ϕ_7 are defined according to,

$$\phi_1 = \vartheta_{20} + \vartheta_{02} \tag{4.34}$$

$$\phi_2 = (\vartheta_{20} + \vartheta_{02})^2 + 4\vartheta_{11}^2 \tag{4.35}$$

$$\phi_3 = (\vartheta_{30} - 3\vartheta_{12})^2 + (3\vartheta_{21} - \vartheta_{03})^2 \tag{4.36}$$

$$\phi_4 = (\vartheta_{30} + \vartheta_{12})^2 + (\vartheta_{03} + \vartheta_{21})^2 \tag{4.37}$$

$$\begin{aligned}\phi_5 = (\vartheta_{30} - 3\vartheta_{12})(\vartheta_{30} + \vartheta_{12})[(\vartheta_{30} + \vartheta_{12})^2 - 3(\vartheta_{03} + \vartheta_{21})^2] \\ +(\vartheta_{30} - 3\vartheta_{12})(\vartheta_{30} + \vartheta_{12})[3(\vartheta_{30} + \vartheta_{12})^2 - (\vartheta_{03} + \vartheta_{21})^2]\end{aligned} \tag{4.38}$$

$$\phi_6 = (\vartheta_{02} - \vartheta_{20})[(\vartheta_{30} + \vartheta_{12})^2 - (\vartheta_{03} + \vartheta_{21})^2] + 4\vartheta_{11}(\vartheta_{30} + \vartheta_{12})(\vartheta_{03} + \vartheta_{21}) \tag{4.39}$$

$$\begin{aligned}\phi_5 = (3\vartheta_{21} - \vartheta_{03})(\vartheta_{30} + \vartheta_{12})[(\vartheta_{30} + \vartheta_{12})^2 - 3(\vartheta_{03} + \vartheta_{21})^2] \\ -(\vartheta_{30} - 3\vartheta_{12})(\vartheta_{30} + \vartheta_{12})[3(\vartheta_{30} + \vartheta_{12})^2 - (\vartheta_{03} + \vartheta_{21})^2]\end{aligned} \tag{4.40}$$

4.11.2 Flusser and Suk's descriptors

Flusser and Suk descriptors are invariant to translation, rotation and scaling. They are presented below.

$$I_1 = \frac{\mu_{20}\mu_{02} - \mu_{11}^2}{\mu_{00}^4} \tag{4.41}$$

$$I_2 = \frac{\mu_{30}^2\mu_{03}^2 - 6\mu_{30}\mu_{21}\mu_{12}\mu_{03} + 4\mu_{30}\mu_{12}^3 + 4\mu_{21}\mu_{03}^3 - 3\mu_{21}^2\mu_{12}^2}{\mu_{00}^1 0} \tag{4.42}$$

$$I_3 = \frac{\mu_{20}(\mu_{21}\mu_{03} - \mu_{12}^2) - \mu_{11}(\mu_{30}\mu_{03} - \mu_{21}\mu_{12}) + \mu_{02}(\mu_{30}\mu_{12} - \mu_{21}^2)}{\mu_{00}^7} \tag{4.43}$$

$$\begin{aligned}I_4 = (\mu_{20}^3\mu_{03}^2 - 6\mu_{20}^2\mu_{11}\mu_{13}\mu_{03} - 6\mu_{20}^2\mu_{02}\mu_{21}\mu_{03} \\ +9\mu_{20}^2\mu_{02}\mu_{12}^2 + 12\mu_{20}\mu_{11}^2\mu_{21}\mu_{03} + 6\mu_{20}^2\mu_{11}\mu_{02}\mu_{30}\mu_{03} \\ -18\mu_{20}\mu_{11}\mu_{02}\mu_{21}\mu_{12} - 8\mu_{11}^3\mu_{30}\mu_{03} - 6\mu_{20}\mu_{02}^2\mu_{30}\mu_{12} + 9\mu_{20}\mu_{02}^2\mu_{21}^2 \\ +12\mu_{11}^2\mu_{02}\mu_{30}\mu_{12} - 6\mu_{11}\mu_{02}^2\mu_{30}\mu_{21} + \mu_{02}^3\mu_{30}^2)/\mu_{00}^{11}\end{aligned} \tag{4.44}$$

4.11.3 Eccentricity

This descriptor measures how circular the object is, it is the ratio between, the longest path of the object border to the other side of the border, and the shortest path, as showed in figure 4.7. The eccentricity ranges from one, which uniquely refers to a perfect circular object, to zero, which indicates a line, shaped object.

One way to calculate eccentricity from the momentum, as defined in [21] , as:

$$\varepsilon = \frac{(m_{2,0} - m_{2,0})^2 + 4m_{1,1}^2}{Area} \tag{4.45}$$

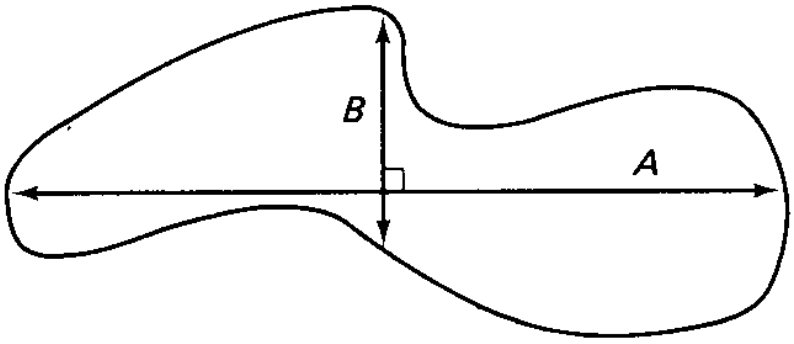

Figure 4.7: The eccentricity is the ratio between A and B i.e. the longest and shortest way from one side of the object to the other.

Chapter 5

Different Learning Structures

5.1 Introduction

The question is now: How should we use these descriptors that we have seen in chapter 4 to build a single system that work for all the tree classes with a satisfactory result? Computer vision classification problems are normally non-linear. We will begin to study some of the main structures. We will only try to study methods that are closely connected to computer vision, some methods will be mentioned only briefly, and some, will be more deeply studied and we will implement and test them. After this we will be able to build a computer vision classifier system for leaf classification.
First we should pre-process the data before we feed it to the classification model.
We will then study some of the more common classification models
Last are the final presentation discussed.
We can see in figure 5.1 how we will handle the data were we first get the data from the descriptor until we present the data from the classifier.

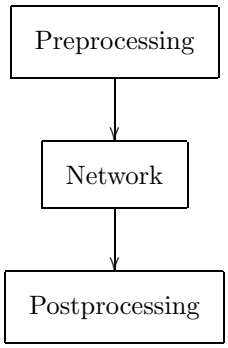

Figure 5.1: This is the Schedule that we will use when we classify the data from the descriptors

5.2 Pre-processing

The most simplest pre-processing method is called rescaling which just makes all the input values similar in size and shape i.e. give them an average of zero and a standard deviation value of one. To read more about this topic, see [6].

Rescaling

To do rescaling, we rescale all the input values separately. For each input-value, x_i, We calculate the mean-value, $\bar{x}_i$, and variance, σ_i^2 from the training data.

$$\bar{x}_i = \frac{1}{N}\sum_{n=1}^{N} x_i^n \tag{5.1}$$

$$\sigma_i^2 = \frac{1}{N-1}\sum_{n=1}^{N} (x_i^n - \bar{x}_i)^2 \tag{5.2}$$

where N, is the number of training-samples and i, is the number of the descriptor. And now we define the rescaled variable,

$$\tilde{x}_i^n = \frac{x_i^n - \bar{x}_i}{\sigma_i} \tag{5.3}$$

which has zero mean and unit standard deviation. There are also other normalization methods such as whitening, KLT, the PCA transform and canonical correlation. For further reading, see [7] and [6].

5.3 Classification systems

In this section we will study some of the most common classification methods. We will not try to cover the vast area of artificial learning. Instead we will try to study some of the main approaches and when they can be used. For further reading, see [6].

5.4 The Tree-structure

The idea is to start with one nod that will use one descriptor. Separate all leaves into sub-groups depending on the value that the descriptor give them and send them to the next underlying nod in the tree and repeat the same thing in the new nod with the descriptor that have been chosen for this nod. This is a static tree with pre-decided descriptors and its a fast way to do the recognition part. But it is difficult to choose the descriptors for each nod and its even more difficult to choose the values those are the borders between different sub nodes. It is a rather simple structure, and a way of analyzing how well the descriptors divide the leaves into different groups and it is easier to see if there are some sorts of leaves that cannot be distinguished. This is of course also a way to get

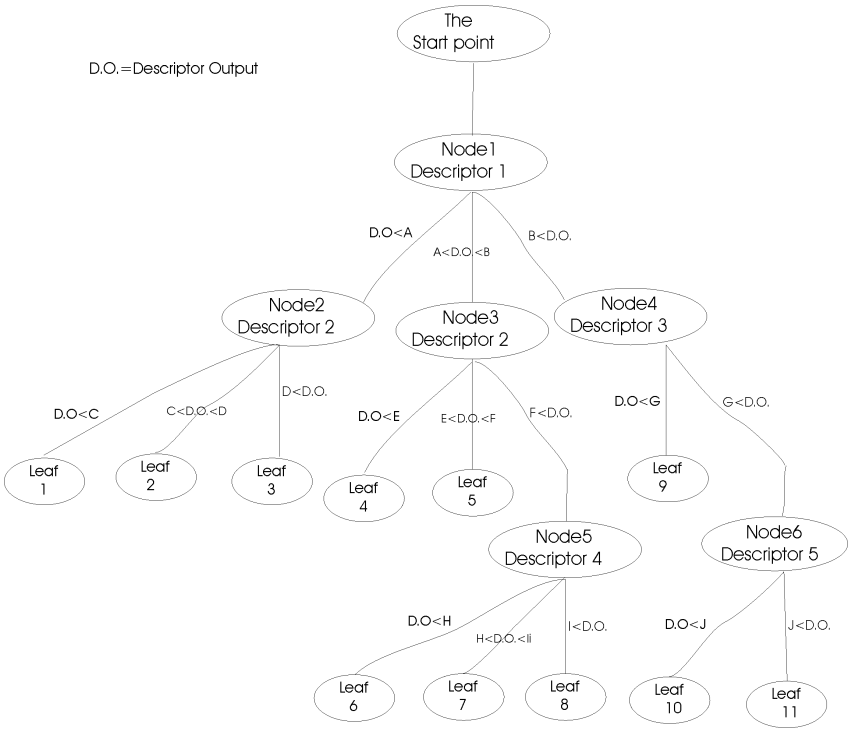

Figure 5.2: This is tree structure were we use the different descriptors to classify the leaves.

control of the system from outside if we would like to use an expert of any sort that could help us with the borders of the descriptors when we choose sub nod. This is illustrated in figure 5.2, we can there follow the tree path. We begin at the starting point, go to the first node. Then our image passes through the first descriptor. Let say the descriptor output gives us a value between a and b, then we go too Nod 3. Here will descriptor number two process our image. Let's say that the value from the descriptor is bellow E, then we can conclude that we have Leaf 4.

5.5 Artificial Neural Networks

The overall name is Artificial Neural Networks (ANN). It is called Networks because it is build of nodes that are connected with each other in a network. Neural because of the weights between all the nodes(each connection has a weight) is updated (or at least tried to be updated) in the same way as the cells in the brain are updated when the brain is activated. There are a vast number of different ANN's so therefor will we just study two of the most common ANN's, backpropagation and Radial Basis Functions. But we will begin with some easier structures just to understand the main idea.

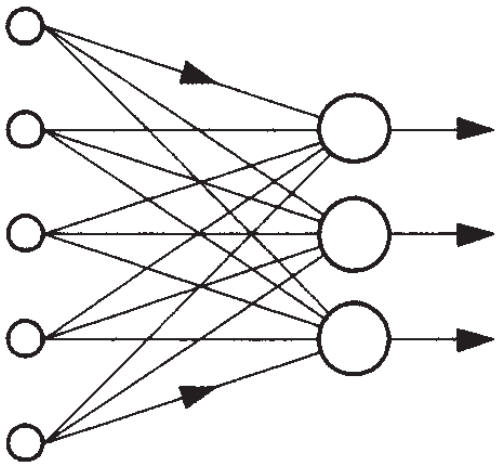

Figure 5.3: This is a image of the nodes and branches of a single layer network.

5.5.1 Feed Forward Single-layer Network

Figure 5.3 shows a Single-Feed-Forward-network, which is characterized by having only one layer of connections, i.e., the input values from the first layer are propagated directly to the output layer. There is a large number of training methods for this kind of network. We will study the Perceptron Learning Algorithm, just to get an idea how it works. For further reading, see [5].

The Perceptron Learning Algorithm

This method is suitable for linear separable binary functions. If the function is not linear we cannot be sure that the learning converge which may cause problems. The basic equations are :

$$u_i = \sum_{j=o}^{N} \omega_{ij} x_j \tag{5.4}$$

$$y_i = \begin{cases} -1 & \text{if } u_i \leq 0 \\ +1 & \text{if } u_i \leq 0 \end{cases} \tag{5.5}$$

where x is the input-vector, y is the output vector and ω is the weight matrix. The weights can be updated using a large number of training rules. We will study only one method that uses linear supervised learning to understand the main idea. To train a network we have: x - the input, $y(x)$ - the output from the net as given by formula 5.4 and formula 5.5 and $t(x)$ - the output that we want to have.

$$\Delta w_{ij} = \begin{cases} 2\eta t_i x_j & \text{if } t_i \neq y_i \\ 0 & \text{otherwise} \end{cases} = \eta(1 - t_i y_i) t_i x_j \text{ because } y_i \in \{-1, +1\} \tag{5.6}$$

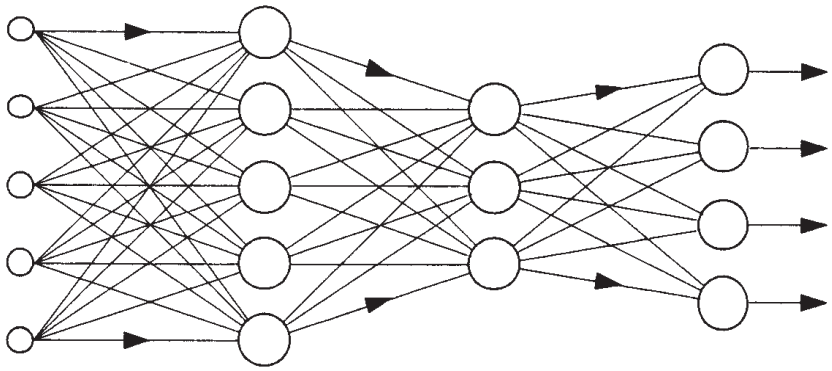

Figure 5.4: This is a image of the nodes and branches of a multi layer network.

The weight matrix is updated according to

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \Delta\omega_{ij}(t) \tag{5.7}$$

where $\omega_{ij}(t+1)$ is the new matrix, and the $\omega ij(t)$ is the old matrix. We also not that we normally chose the inlearning constant within $0 \leq \eta \leq 1$. We see that there no change when the classification is correct and when it is not, each element of the weight matrix is changed to minimizes the error between what we get from our network and what we should get from it, with 2η. This works well if the function, that we want to have from our network, is linear. Otherwise we need to use a Multi Layer Network.

5.5.2 Feed Forward Multi-layer Network

A multi-layer feed forward network, illustrated in figure 5.4, which in comparison with a single layer network has layers in between the input and output layer. There are many different training methods also for this kind of network. We will study Backpropagation and Radial Basis Functions. These are the two main training procedures but there are many other different techniques. Most of them are just variations within the two techniques, some of them will be mentioned here.

5.5.3 Backpropagation

The Backpropagation works as a non-linear, parallel computer. The advantages of the Backpropagation in computer vision are:

- Non-linearity
- Robustness

We will study some of the different approaches below. We will just do a short surveys of the main parts of the network and will not try to study these in detail. For a more detailed description, see [5] [6] and [20].

There are a number of different methods that train a MLP, the most known, and one of the first ones is backpropagation. It has a large number of followers and there are many methods that are related. We will therefore first study backpropagation, and then study some of the better offsprings to this method. The system are built with a number of layers with a certain number of nodes, as can be seen in figure 5.4. It needs to have an input layer and an output layer. And then it as a number of layers between those two layers called hidden layers.

How to calculate the value from the input layer to the next proceeding layer

We start with the input values that is the first layer and calculate the values for the first hidden layer. We do this as,

$$a_i = \sum_{j=1}^{N} \omega_{ij} x_j \tag{5.8}$$

$$y_i = f(a_i) \tag{5.9}$$

where x is the input vector from the input layer and y is the output vector from the nodes of the first hidden layer. ω_{ij} is one element of the W-matrix that is the connection layer between the input layer and the first hidden layer.

The squash function

The function, (5.9), is called the squash function and is nonlinear for Multi layer networks. Normally the sigmoid function, (5.10) is chosen.

$$f(u) = \frac{1}{1 + e^{-u}} \tag{5.10}$$

This function has an easily calculated derivative, which makes it easy to implement.

$$f'(u) = f(u)(1 - f(u)) \tag{5.11}$$

How to calculate the output values from our input values

First we start to calculate the first layer as we did above, when we calculated the value from the input layer to the next proceeding layer. The next step that we will take, is to move from the first hidden layer to the next hidden layer in exactly the same way as in (5.8) and (5.9), when we went from the input layer to the first output layer and we will continue until we will reach the output layer and then we have the output values for the output layer. And then we will have the values that we want.

Back propagation

We will study the basic learning equations. The idea is to is to minimize the error function, E. We will chose it as,

$$E = (t_i - y_i)^2 \tag{5.12}$$

where $\boldsymbol{t_i}$ is the out value we want, and $\boldsymbol{y_i}$ is the out value we get.
The question is now how do we do this? Our multi layer neural network is build on functions with different weights as we have seen in (5.8) and (5.9). There for we want to minimize the error with respect to those weights. As we can see below, we have $\Delta\omega_{ij}$ that is the update that we use as we have seen in (5.7).

$$\Delta\omega_{ij} = -\eta\frac{\partial E}{\partial\omega_{ij}} \tag{5.13}$$

$\boldsymbol{\eta}$ is the learning rate, that is the length that we step towards the error everytime. Typical values are, $\mathbf{0.05 \leq \eta \leq 0.9}$. If we chose a too small step then it takes us a lot of time, and if we chose a too large number then there we will not reach the minimum at all because we will pass over it every time. There are many methods that try to solve this problem, and we will look at one later.
If we calculate the derivative we get,

$$\frac{\partial E}{\partial\omega_{ij}} = \delta_i y_j \tag{5.14}$$

Where $\boldsymbol{\delta_i}$ is the value of node i, from the layer closest to the output layer, and the output value $\boldsymbol{y_j}$ of node j, that is closest to the input layer.

$$\delta_i = \begin{cases} -(d_{pi} - y_{pi})f_i' & \text{for output nodes} \\ f_i'\sum_k \omega_{ki}\delta_k & \text{for hidden nodes} \end{cases} \tag{5.15}$$

And $\boldsymbol{f'}$ is easy to calculate if we use 5.10 as we see in 5.12.

Resilient propagation

This is one of the better ANN models. We will just study the small changes that have been made in comparison with the backpropagation algorithm. Resilient propagation (Rprop) has some learning-rate adjustments and uses only the sign of the gradient terms, and not the magnitude. The length of the step that we will take when we minimize the error is decided by Δ_{ij} and is calculated as below,

$$\Delta_{ij} = \begin{cases} \eta^+\Delta_{ij}(t-1) & \text{if } \frac{\partial E}{\partial\omega_{ij}}(t-1)\cdot\frac{\partial E}{\partial\omega_{ij}}(t) > 0 \\ \eta^-\Delta_{ij}(t-1) & \text{if } \frac{\partial E}{\partial\omega_{ij}}(t-1)\cdot\frac{\partial E}{\partial\omega_{ij}}(t) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.16}$$

where $\mathbf{0 < \eta^- < 1 < \eta^+}$

$$\Delta\omega_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & \text{if } \frac{\partial E}{\partial \omega_{ij}}(t) > 0 \\ +\Delta_{ij}(t) & \text{if } \frac{\partial E}{\partial \omega_{ij}}(t) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.17}$$

Finally we have a special case if the partial derivative changes sign which indicates that a minimum has been passed and the previous weight update should be canceled.

$$\Delta\omega_{ij}(t) = -\Delta\omega_{ij}(t-1) \text{ If } \frac{\partial E}{\partial \omega_{ij}}(t-1)\frac{\partial E}{\partial \omega_{ij}}(t) < 0 \tag{5.18}$$

According to [20] this is the fastest Backpropagation that has been constructed so far.

Quick-propagation

This method is based on two assumptions:

- E($\boldsymbol{\omega}$) for each weight can be approximated with a parabola that has a minimum.
- The change in the slope of E($\boldsymbol{\omega}$) for each weight is not affected by all the other weights that change at the same time.

The update rule is:

$$\Delta\omega_{ij}(t) = \frac{S_{ij}(t)}{S_{ij}(t-1) - S_{ij}(t)}\Delta\omega_{ij}(t-1) \tag{5.19}$$

where $\boldsymbol{S(t) = \frac{\partial E}{\partial \omega_{ij}}}$

This method is based on Newton's method from numerical analysis, and it is one of the faster and only Rprop seems to be consistently faster.

5.5.4 Pruning

The main idea behind pruning is to train a network that is oversized. Oversized is a network that has a large quantity of nodes and therefor get slow, in comparison with what it could be if the network was smaller, and we will try to remove all the nodes that are redundant. We will learn faster with a large network and with less sensitivity to initial conditions. A straight way, called sensitivity calculations, of solving this is to, for every weight, set the weight to zero and evaluate the change in the error function. If it increases too much we keep the weight, otherwise it is removed. An alternative approach is based on penalty term methods, which penalize complex solutions. For further reading, see [20].

5.5.5 Size versus deep and other options

It is difficult to choose a sufficient number of layer and number of nodes for each layer in advance. There is not a straightforward answer to how we should do it, just by knowing the data that we want our classifier to work with. We see the pruning as on way of defining the size of the network. There are a number of formulas that calculate a lower boundary for how many nodes we got to have. But there are no good estimates for the upper boundary which in practice may be impossible to determine because of the different type of the input data that we might have. What we can do is to chose a small number of nodes and increase them until we have a sufficient answer, or we can start large number of nodes and decrease them until we see when we reach the limit that we have decided.

To choose how many layers that we need is an even more complex issue. There have been proven that if we have an infinite number of nodes. Then we can approximate any function with only one hidden layer and with two hidden layers we can construct any function with the ANN. But its not so common with a infinite number of nodes. So which guidelines could we use? One rule is that more than two hidden layers are of little use because of the time it takes for the network to learn and because of poor convergence. And to use more nodes than $N - 1$, where N is the number of input values, is not necessary either. But as we saw when we studied pruning, it could be useful to start with more nodes because of faster convergence. There have been many articles and books written about this subject, for further reading, see [20].

5.5.6 Radial-Basis Functions

The main difference with this method, in comparison with for example back-propagation, is that it determines the activation of a hidden unit by the distance between the input vector and normally some average vector.

There are a number of advantages if we use RBF (Radial Basis Functions) instead of backpropagation techniques e.g. the training procedure is faster because of the internal representation form.

- First we train the parameters of the basis function, using unsupervised methods i.e. methods that use only the input data and not the target data i.e. the data that we want to have. Examples are the k-means algorithm and vector quantization.
- Finally we determine the output layer weights. This requires the solution of a linear problem, and therefore is it possible to solve it fast.

The basic equations are,

$$y_k(x) = \sum_{j=1}^{M} \omega_{kj}\phi_j(x) + \omega_{k0} \tag{5.20}$$

where the ω_{k0} is the bias. We chose Gaussian basis functions which gives us,

$$\phi_j(x_j) = exp(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}) \tag{5.21}$$

where x is the input vector μ_j is the vector determining the center of the RBF, ϕ_j .

The RBF method origins from the idea of interpolating a function but most functions are too oscillating to motivate an exact interpolation in all data points, if we chose the number of basis functions by how complex the function is will normally be a good approximation. To chose the number of basis functions is something we must do with care! There are a number of rules that we should consider.

- The number of basis functions does not need to be equal to the number of input values. There are usually much fewer basis functions than input values.
- We determine the centers, μ_j, of the basis functions from the training process instead.

Networking training

This is a two-stage procedure.

- First we train the input layer, or more correctly, the basis functions, parameters with unsupervised learning with only the input-vectors, $\boldsymbol{x}$. That is, we decides the values of μ_j and σ_j.
- Then we train the second layer as a single layer network while the basis functions are kept fix.

For more details about RBF methods, see [6].

5.6 Recurrent Network

There is an alternative to feed forward networks, which is named recurrent networks. This is a network that not only sends the signal forward but also to nodes, or states, that have been processed before, as illustrated in figure 5.5. We will not go into details how these networks work, but just mention that they exists.

5.7 Post Processing

This is where the data from the network is transferred to understandable data for the user of the learning system. This can be done in a number of ways, with graphs to make it easier for the user, or with use of other parameters such as certainty measures together with our output from the network to analyze if it is reliable. This is a broad area but will not be discussed in this thesis except a small explanation when the system is presented in chapter 7.

5.8 Concluding remarks

When we choose methods, we should first think about what we want from the method and what our resources are. How much knowledge of the objects that

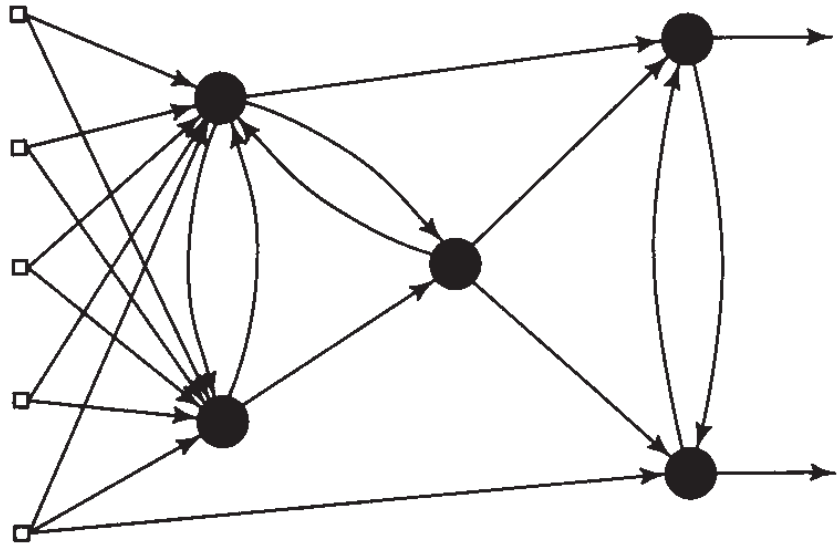

Figure 5.5: This is a recurrent network.

we want to classify do we have? How much time, of our own, could we spare to build the system? How fast is our computer? How many descriptors do we have? How many samples of each class of objects do we have? These are the questions we should answer before we can choose method. We will use a feed-forward network. The network will have one hidden layer because its most likely that the classification will be non-linear, but we still want a network that converge fast and that is why we will choose a one hidden layer network instead of two hidden layers. We will also hope it will be enough with one hidden layer because the descriptors will do lot of the work that we otherwise would have if we used the images as input-values, otherwise we could be forest to use two layers due to higher complexity.
In [20] is there a rule that says that it is sufficient with the number of input values minus one. But if we increase the number of nodes, the lesser the number of iterations are needed to train the network. We will therefor try different number of nodes to see which one is optimal. We will choose the sum of squared errors(SSE) as an error function, and the sigmoid function to be the squash function because they are standard functions.

Chapter 6

Implementation

6.1 Description of the implementation and results

The implementation has been done in MatLab, on a Sun station with a Unix operation system. To pre-process we locate the leaf in the picture and remove all other objects from the picture. We use the techniques from chapter 4 to get data that will be valuable to classify different tre classes from each other. We will discuss the methods that have been implemented and problems that have occurred during implementation in this chapter. A small presentation is made of the results and comments from each descriptor.

6.1.1 Samples

The test has been done with 50 samples from each tree class, and there are 15 different tree classes, as can be seen in figure 6.1 to 6.5. The names in the figures are in latin. The image samples are in the form of digital images, obtained with a 300 dpi color scanner. The scanning has not been done to make the information be easier obtained by the descriptors. The collecting of the leaves has been done so that large variations of leaves have been included. The leafs that have been used are the leaves showed in figure 6.1 to 6.5.

6.1.2 Different ways of evaluating and describe the descriptors

There are several measures that we can use to analyze the descriptor and evaluate if they describe the different classes of leaves well or not. We will study and use nine measures to analyze the descriptors that are implemented in our computer vision classifier.
First we will define some variables that we will use later on.
M_h is the average of the descriptor output for the class of leaves that has the closest average higher than the specified class. And M_l is the average for the class of leaves that has the closest average lower than the specified class. Both M_h and M_l will also be presented in the diagrams below.

Figure 6.1: These are images of leaves from tree class 1 to 3. As seen, tree class 1 is Ulmus carpinifolia. tree class 2 is Acer platanoides, and tree class 3 is Ulmus.

Figure 6.2: These are images of leaves from tree class 4 to 6. As seen, tree class 4 is Quercus robur. tree class 5 is Alnus incana, and tree class 6 is Tilia.

M is the average value from the descriptor.

- S_u and S_d are used to find out if the group of leaves is described well by

Figure 6.3: These are images of leaveas from tree class 7 to 9. As seen, tree class 7 is Salix fragilis, tree class 8 is Populus tremula, and tree class 9 is Corylus avellana.

Figure 6.4: These are images of leaveas from tree class 10 to 12. As seen, tree class 10 is Sorbus aucuparia, tree class 11 is Prunus padus, and tree class 12 is Tilia.

the descriptor. The formulas are described by 6.1 and 6.2.

$$S_h = \frac{M_h - M}{M} \tag{6.1}$$

Figure 6.5: These are images of leaveas from tree class 13 to 15. As seen, tree class 13 is Populus, tree class 14 is Sorbus hybrida , and tree class 15 is Fagus silvatica.

and,

$$S_l = \frac{M - M_l}{M} \tag{6.2}$$

- M is the average value from the descriptor.
- $Width$, The width of a class of leaves from a descriptor will be presented. I.e. highest value minus the lowest value from a specific descriptor and group.
- $Low - High$, Highest and lowest values from the samples for each specific group of leaves will be presented. For these two values will we also show a diagram for each descriptor, to better illustrate how the different values vary between different descriptors.
- σ, The standard deviation of the treeclass

We will also look at a table with the number of the treeclass on the y-axel and the range of values that the descriptor give each treeclass on the x-axel.

6.2 CSS

Implementation

There are some changes that we could do while using this algorithm, if we compare to the original implementation described earlier.
The first is when we find the maximums of the scale-space curve since the curve

oscillates close to a maximums. Because of these oscillations, and because we want to make the algorithm fast, will we find the maximums in another way than explained in the article. The maximums are found by the assumption that the curve is cutting the x-axis an even number of times. When it has lowered the number of cuts with two, we presume that we have found a new maximum and we try to find where it is. One other difficulty is which features we will extract from the Scale Space. Some leaves have only two maximums. The parameter that we will use is the average height of the two highest maximums, the distance between them and average number of maximums with two certain gauss kernels (σ=8 and σ =9).

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.18	0.03	14.55	24	2-26	5.90
Leaf 2	0.03	0.08	14.09	36	3-39	9.96
Leaf 3	0.10	0.12	11.71	21	1-22	5.52
Leaf 4	0.05	0.22	8.92	36	1-37	8.34
Leaf 5	0.05	0.05	9.37	28	1-29	6.67
Leaf 6	0.03	0.02	10.04	28	1-29	5.74
Leaf 7	10.8	-	0.59	5	0-5	0.77
Leaf 8	0.03	0.21	21.89	37	3-40	10.24
Leaf 9	0.27	0.15	17.21	22	4-26	5.37
Leaf 10	0.08	0.006	13.0	49	1-50	12.03
Leaf 11	0.006	0.09	12.92	18.0	5-23	3.54
Leaf 12	0.02	0.05	9.85	20	1-21	4.94
Leaf 13	-	0.03	22.53	34	6-40	8.87
Leaf 14	0.13	0.03	10.36	41	1-42	8.98
Leaf 15	0.29	0.92	6.92	14	1-15	3.52

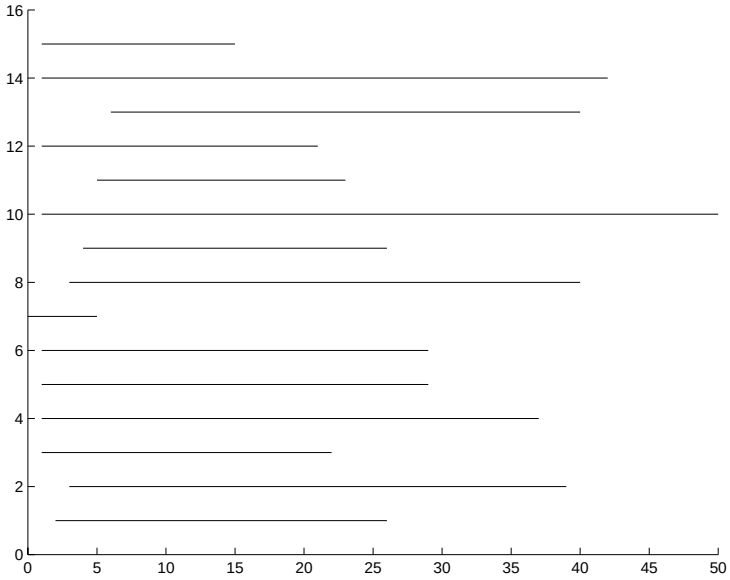

Table 6.1: The average height of the two highest maximums for CSS.

Conclusions from table 6.1.

We notice that the S_h and S_h are rather small. This may indicate that this descriptor does not describe the tree classes very well and as we see in the diagram the values are not so good. The avrages, M, are similiar too, except for tree class 7. We can conclude that this is not a very useful descriptor, but will still be somewhat useful.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.005	0.02	10.21	16.98	4.75-21.72	3.80
Leaf 2	0.02	0.10	6.80	21.75	1.05-22.80	3.38
Leaf 3	-	0.007	11.18	16.53	5.90-22.43	4.85
Leaf 4	0.04	0.02	8.84	15.68	2.25-17.93	2.62
Leaf 5	0.02	0.02	8.65	14.95	6.80-21.75	3.04
Leaf 6	0.007	0.07	11.10	17.43	5.2-22.63	5.84
Leaf 7	0.11	-	6.14	22.78	0-22.78	6.24
Leaf 8	0.02	0.05	8.44	18.33	3.2-21.5	3.37
Leaf 9	0.003	0.005	10.26	14.33	6.55-20.88	3.02
Leaf 10	0.03	0.02	6.97	11.83	0.90-12.73	2.11
Leaf 11	0.02	0.08	9.98	16.85	5.23-22.08	5.42
Leaf 12	0.05	0.10	8.03	14.73	6.75-21.48	1.84
Leaf 13	0.08	0.04	9.20	15.8	6.1-21.9	3.86
Leaf 14	0.08	0.003	10.29	13.43	4.15-17.58	3.68
Leaf 15	0.12	0.03	7.19	6.90	5.18-12.08	1.02

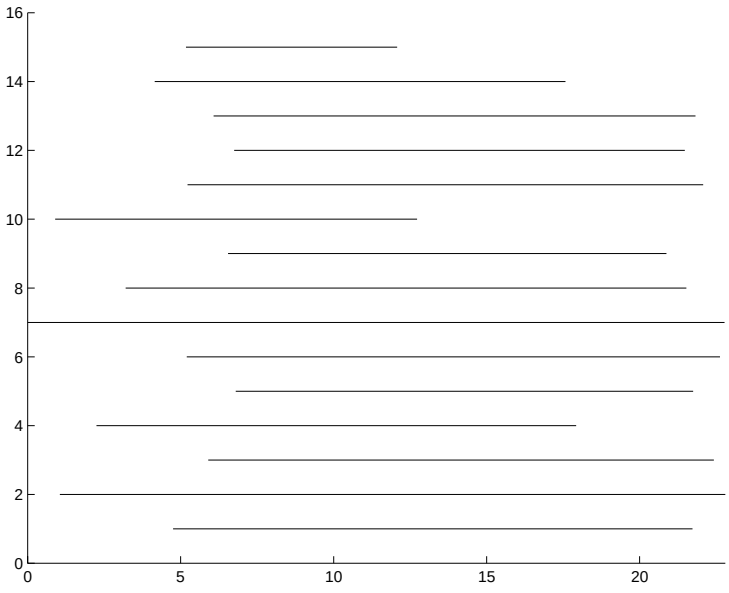

Table 6.2: The length between the two highest maximums for CSS.

Conclusions from table 6.2.

We notice that the S_h and S_h are rather small. This may indicate that this descriptor does not describe the tree classes very well and as we see in the diagram the values are not so good. The avrages, M, are similiar too. We conclude that this is a bad descriptor.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.01	0.15	2.44	3.0	2-5	0.78
Leaf 2	0.92	0.03	8.39	11	0-11	1.71
Leaf 3	0.20	0.01	2.47	3	2-5	0.84
Leaf 4	0.03	0.09	8.16	7	5-12	1.80
Leaf 5	0.003	0.003	4.13	2	4-6	0.91
Leaf 6	0.02	0.23	3.85	4	2-6	0.91
Leaf 7	77.00	-	0.03	2	0-2	0.23
Leaf 8	0.09	0.003	4.15	3	4-7	0.51
Leaf 9	0.17	0.99	2.08	2	2-4	0.36
Leaf 10	-	0.48	16.13	14	10-24	2.72
Leaf 11	0.30	0.17	2.96	2	2-4	0.96
Leaf 12	0.03	0.02	4.03	2	3-5	0.23
Leaf 13	0.65	0.08	4.51	3	4-7	0.86
Leaf 14	0.10	0.39	7.44	8	4-12	1.71
Leaf 15	0.02	0.02	3.93	3	2-5	0.38

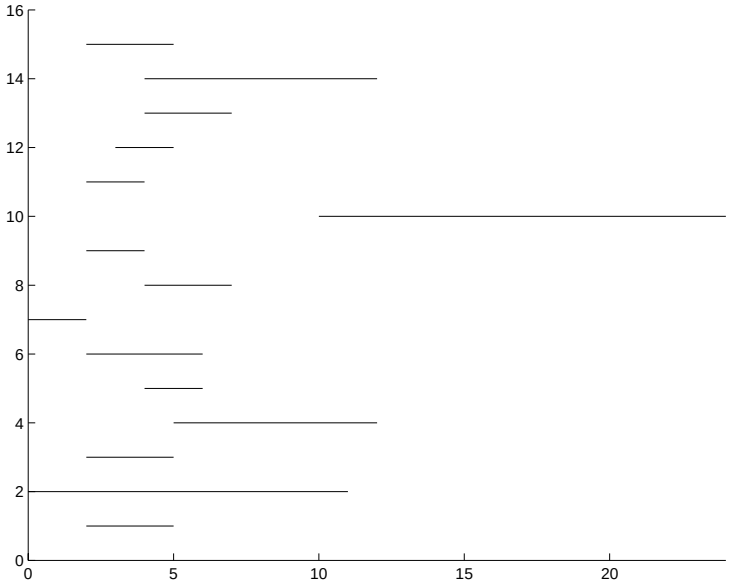

Table 6.3: This is the average number of maximums between $\sigma = 8$ and $\sigma = 9$ for CSS.

Conclusions from table 6.3

This descriptor has average values for the indicators S_h and S_l. And as we can see in the diagram can this descriptor distinguish between, for example, tree class 7, 8 and 10. As we see in the diagram this descriptor is useful for the purpose of tree classification.

6.3 Area

Implementation

When we have the black and white image is this method straightforward. We just count the number of white pixels in the image 6.4.

Conclusions from table 6.4

This descriptor has rather low values for S_h and S_l, but we can that we can separate, for example, 2, 3 and 7. in the diagram we see that tree class 2 and 10 have significantly higher values this descriptor will therefor be useful for those classes. We can also see that leaf class 7 has significantly lower values.

6.4 Eccentricity

Implementation

When implementing this method we can only choose if we use a black and white image or a grey image and we will choose the black and white image as the input value.

Conclusions from table 6.5

S_h and S_l are small for all descriptors and when we study the min-max values in the diagram we see that most of the classes ranges from 0.4 to 0.8. There are three classes, class 2, 6, and 12, which have significantly lower min-max values and also lower average, M, values and for those classes could this descriptor be useful.

6.5 Moment features

For this descriptor has a black and white image been used as the input value been used.

6.5.1 Flusser moments

Implementation

No changes has been done to this descriptor.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.05	0.06	5.90	11.23	2.16-13.39	2.54
Leaf 2	-	0.38	35.77	56.97	11.75-68.72	13.95
Leaf 3	0.07	0.07	5.52	4.84	3.09-7.93	1.19
Leaf 4	0.11	0.26	4.61	5.79	2.26-8.04	1.21
Leaf 5	0.27	0.92	2.70	3.33	1.24-4.57	0.67
Leaf 6	0.22	0.05	6.19	9.86	1.83-11.70	2.65
Leaf 7	11.90	-	0.21	1.79	0.14-1.93	0.20
Leaf 8	0.06	0.04	8.52	11.74	3.15-14.89	2.59
Leaf 9	0.06	0.02	7.75	11.29	3.38-14.68	2.07
Leaf 10	0.61	0.60	22.19	30.63	7.85-38.48	6.58
Leaf 11	0.35	0.21	3.41	3.26	1.77-5.04	0.84
Leaf 12	1.46	0.05	9.01	12.69	2.88-15.57	3.10
Leaf 13	0.04	0.05	8.19	11.60	3.65-15.24	2.73
Leaf 14	0.02	0.18	7.57	12.03	3.88-15.91	2.42
Leaf 15	0.08	0.10	5.12	7.28	1.70-8.99	1.56

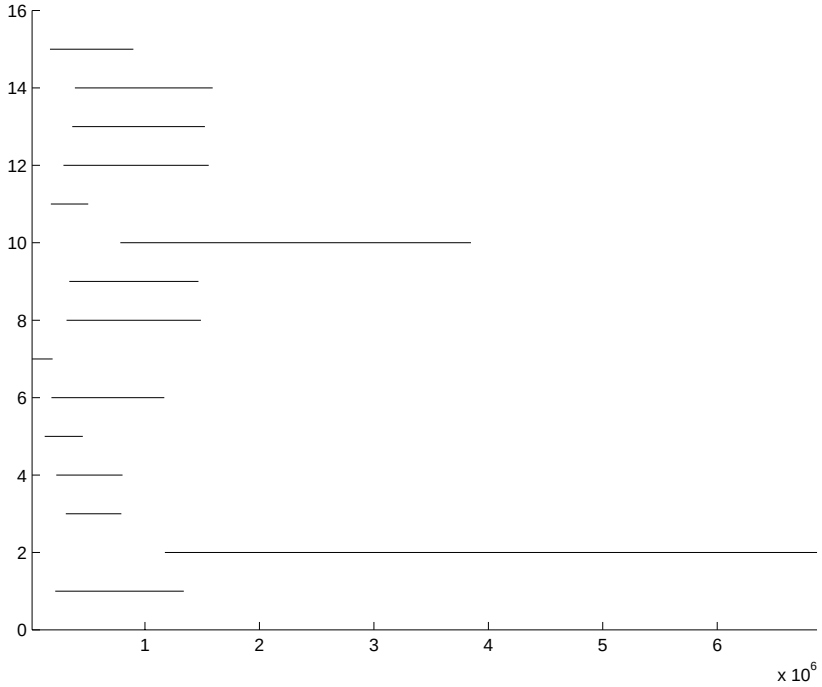

Table 6.4: A table of the results from the Area. All values, except S_h and S_l, should be divided with 10^5

Conclusions from table 6.6

This descriptor have high values from S_h and S_l, and if we look at the min-max values we see that the some classes can be well separated, which makes this a good descriptor. The two classes 2 and 10 have exceptionally large values. we can also see that the class 7 has exceptionally low values which makes it easy to classify also.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.01	0.0012	0.58	0.50	0.27-0.77	0.10
Leaf 2	0.16	-	0.36	0.72	0.0005-0.72	0.20
Leaf 3	0.007	0.05	0.63	0.41	0.37-0.86	0.14
Leaf 4	0.01	0.007	0.63	0.41	0.36-0.77	0.09
Leaf 5	0.05	0.02	0.60	0.47	0.31-0.78	0.15
Leaf 6	0.07	0.14	0.42	0.75	0.009-0.76	0.19
Leaf 7	-	0.23	0.83	0.37	0.52-0.89	0.07
Leaf 8	0.11	0.10	0.50	0.58	0.18-0.75	0.14
Leaf 9	0.02	0.01	0.59	0.48	0.29-0.78	0.10
Leaf 10	0.005	0.009	0.64	0.41	0.37-0.77	0.10
Leaf 11	0.001	0.002	0.58	0.41	0.36-0.78	0.16
Leaf 12	0.11	0.07	0.45	0.66	0.07-0.73	0.16
Leaf 13	0.002	0.04	0.58	0.42	0.36-0.78	0.14
Leaf 14	0.05	0.10	0.55	0.50	0.27-0.77	0.12
Leaf 15	0.30	0.005	0.64	0.41	0.37-0.78	0.09

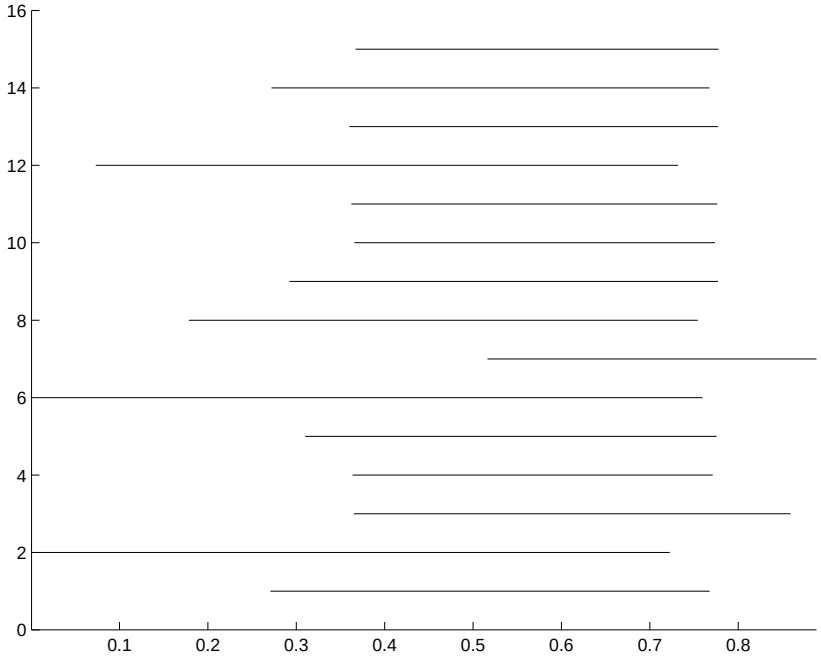

Table 6.5: A table of the results from the Eccentricity. All values, except S_h and S_l, should be divided with 10^5

Conclusions from table 6.7

We see that the values S_h and S_l are very high, but if we look at the descriptors min-max values we see that the tree classes 2 and 10 have exceptionally high values which should make them easy to distinguish with this descriptor. We also see that leaf class 5 and 7 have exceptionally low values which makes them easy to distinguish. We see that most tree classes have a rather large width between lowest and highest values, so we may conclude that for the majority of

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.15	0.31	1.19	4.23	0.14-4.37	0.99
Leaf 2	0.15	0.87	10.58	27.94	1.32-29.26	6.28
Leaf 3	0.06	0.23	0.66	2.65	0.05-2.70	0.60
Leaf 4	0.29	0.35	0.34	0.84	0.06-0.91	0.19
Leaf 5	1.95	0.93	0.07	0.19	0.01-0.20	0.04
Leaf 6	0.44	0.15	0.83	2.60	0.09-2.69	0.58
Leaf 7	13.70	-	0.005	0.04	0.0006-0.04	0.006
Leaf 8	0.30	0.16	0.51	1.27	0.05-1.32	0.27
Leaf 9	6.68	0.006	1.38	8.85	0.17-9.02	1.40
Leaf 10	-	0.13	12.20	51.63	1.32-52.95	8.64
Leaf 11	0.53	0.66	0.22	0.62	0.02-0.64	0.16
Leaf 12	0.001	0.13	1.37	4.62	0.13-4.75	0.85
Leaf 13	0.17	0.06	0.70	2.09	0.05-2.14	0.48
Leaf 14	0.006	0.001	1.37	7.20	0.04-7.23	1.19
Leaf 15	0.19	0.22	0.43	1.16	0.05-1.21	0.24

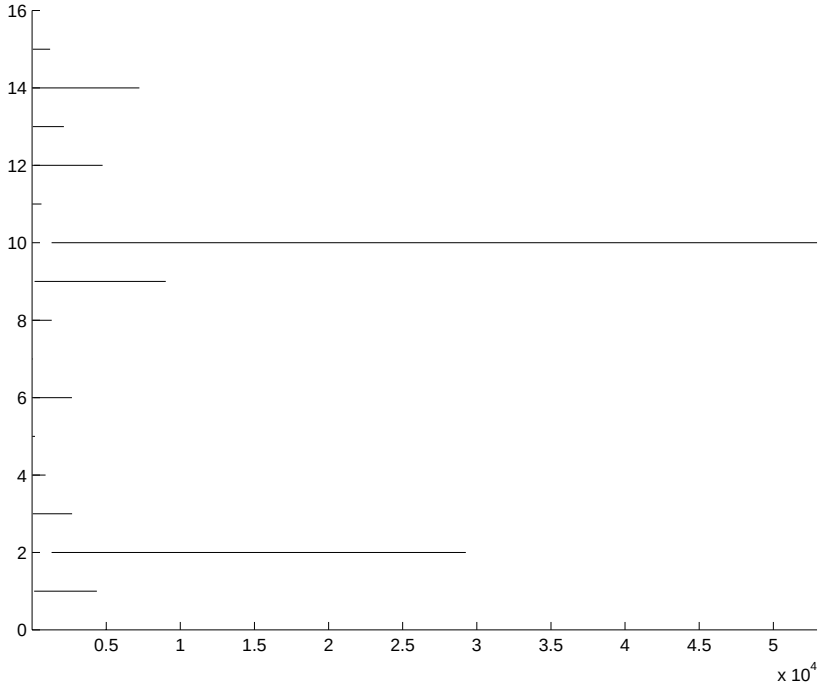

Table 6.6: A table of the results from the Flusser1. All values, except S_h and S_l, should be divided with 10^3

the tree classes is this a not so good descriptor.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.69	0.70	1.67	18.74	0-18.74	3.79
Leaf 2	-	0.47	30560	484110	0-484110	71940
Leaf 3	5.45	0.48	0.08	1.42	0-1.42	0.22
Leaf 4	4.05	0.79	0.0008	0.001	0-0.001	0.0002
Leaf 5	227.31	1.00	0.0000007	0.000001	0-0.000001	0.000002
Leaf 6	2.32	0.84	0.50	17.45	0-17.45	2.15
Leaf 7	560.12	-	0.0000001	0.0000001	0-0.000001	0.0000001
Leaf 8	2.47	0.68	0.01	0.20	0-0.20	0.03
Leaf 9	669.66	0.73	24.0	1492.2	0-1492.2	172.7
Leaf 10	0.90	1.0	16080	341120	0-341120	54130
Leaf 11	3.75	1.0	0.0002	0.003	0-0.0025	0.0004
Leaf 12	1.30	0.41	2.83	114.21	0-114.21	13.75
Leaf 13	0.91	0.71	0.04	0.86	0-0.86	0.13
Leaf 14	2.69	0.56	6.51	325.97	0-325.97	39.17
Leaf 15	2.08	0.80	0.004	0.06	0-0.06	0.01

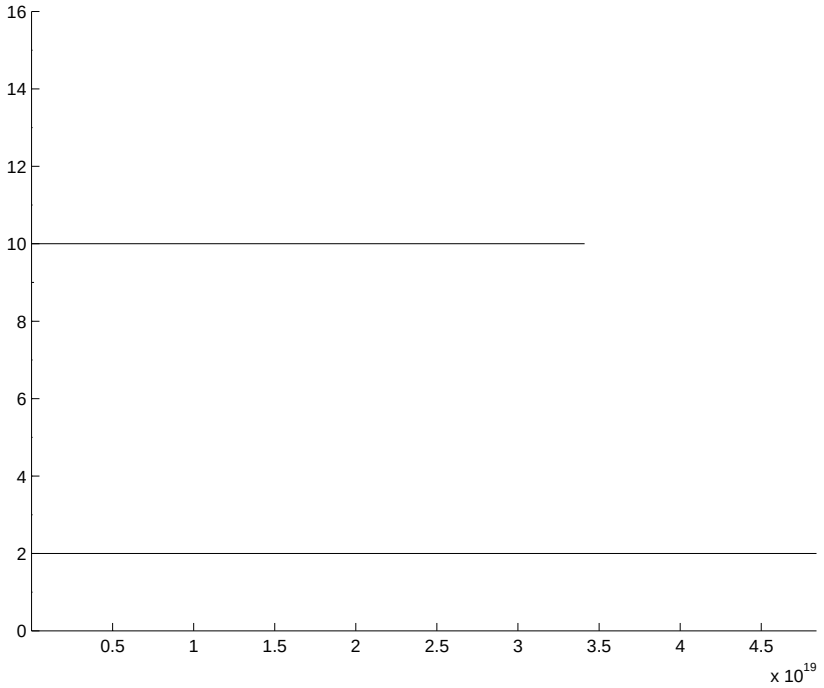

Table 6.7: A table of the results from the Flusser2. All values, except S_h and S_l, should be multiplied with 10^{14}

Conclusions from table 6.8

We see that the values S_h and S_l are very high, but if we look at the descriptors min-max values we see that all the descriptors have many values in common. If we study the average, M, we see that the two classes 2 and 10 has far higher values than the rest of the classes why this descriptor could be very good for those two tree classes. We also see that the average for the tree classes 5, 7

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.12	0.52	0.34	2.31	0.003-2.31	0.49
Leaf 2	1.16	0.96	15.32	78.26	-6.38-71.89	16.95
Leaf 3	0.44	0.28	0.11	0.96	0.0005-0.96	0.17
Leaf 4	1.11	0.36	0.02	0.10	0.0006-0.10	0.02
Leaf 5	10.44	0.98	0.001	0.006	0-0.006	0.001
Leaf 6	1.09	0.31	0.16	1.36	0.001-1.36	0.23
Leaf 7	59.49	-	0.001	0.0003	0-0.0003	0.0001
Leaf 8	0.82	0.21	0.04	0.26	0.0003-0.26	0.05
Leaf 9	0.10	0.27	0.52	10.80	0.005-10.81	1.38
Leaf 10	-	0.54	33.10	244.40	0.28-244.68	41.96
Leaf 11	0.57	0.91	0.01	0.06	0-0.06	0.01
Leaf 12	0.38	0.11	0.38	3.58	0.003-3.58	0.56
Leaf 13	0.40	0.45	0.08	0.65	-0.0001-0.65	0.12
Leaf 14	25.76	0.09	0.57	9.91	0.0003-9.91	1.32
Leaf 15	0.27	0.53	0.03	0.20	0.0003-0.20	0.04

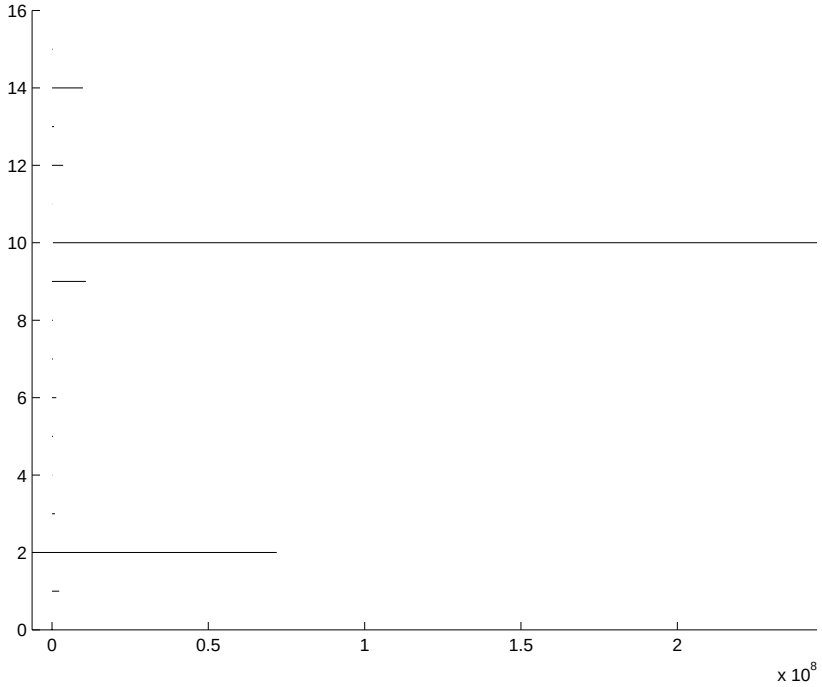

Table 6.8: A table of the results from the Flusser3. All values, except S_h and S_l, should be divided with 10^6

and 11, are much smaller than for rest of the classes which makes this a good descriptor for those classes.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	-0.56	-0.45	-0.03	0.22	-0.22-0	0.05
Leaf 2	-0.09	-	-19.57	189.06	-189.07-(-0.009)	29.73
Leaf 3	-0.80	-0.21	-0.006	0.06	-0.06-0	0.01
Leaf 4	-0.66	-0.94	-0.0004	0.004	-0.004-0	0.0007
Leaf 5	-1.00	-21.78	0	0.0001	-0.0001-0	0
Leaf 6	-0.52	-1.26	0.01	0.21	-0.21-0	0.03
Leaf 7	-	-2186.3	0	0	0-0	0
Leaf 8	-0.29	-3.89	-0.001	0.01	-0.01-0	0.002
Leaf 9	-0.42	-192.93	-0.09	3.53	-3.53-0	0.42
Leaf 10	-0.99	-0.10	-17.87	214.84	-214.85-(-0.01)	35.42
Leaf 11	-0.96	-1.97	-0.0001	0.001	-0.001-0	0.0002
Leaf 12	-0.31	-0.17	-0.05	0.87	-0.87-0	0.12
Leaf 13	-0.17	-1.10	-0.007	0.08	-0.08-0	0.01
Leaf 14	-0.14	-0.73	-0.05	1.70	-1.70-0	0.21
Leaf 15	-0.48	-0.42	-0.0008	0.007	-0.007-0	0.001

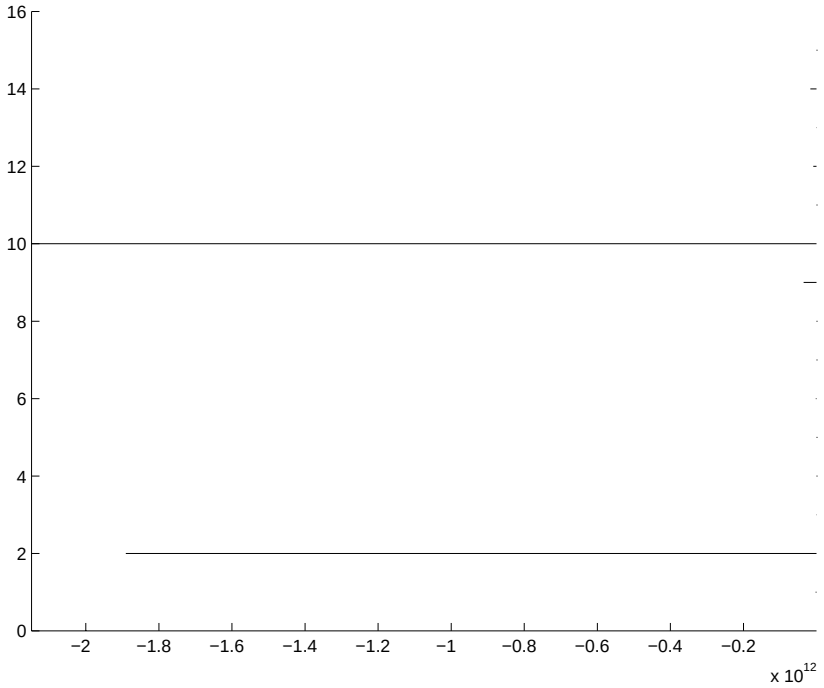

Table 6.9: A table of the results from the Flusser4. All values, except S_h and S_l, should be multiplied with 10^{26}

Conclusions from table 6.9

We see that the values S_h and S_l are very high, but if we look at the descriptors min-max values we see that all the descriptors have many values in common why this descriptor is not so good. Tree classes 2 and 10 have significantly higher values which makes this descriptor useful.

6.5.2 Hu moments

$Name$	S_h	S_l	M	$Width$	$Low-high$	σ
Leaf 1	0.04	0.02	0.77	0.77	0.29-1.06	0.15
Leaf 2	0.04	0.04	0.81	0.71	0.53-1.25	0.15
Leaf 3	0.04	0.28	0.59	0.85	0.24-1.10	0.30
Leaf 4	0.02	0.03	0.76	0.64	0.52-1.16	0.17
Leaf 5	0.28	0.04	0.84	0.89	0.45-1.34	1.18
Leaf 6	0.04	0.06	0.67	0.74	0.33-1.07	0.17
Leaf 7	3.41	-	0.10	0.16	0.09-0.26	0.02
Leaf 8	0.39	0.77	0.42	0.34	0.33-0.68	0.06
Leaf 9	0.03	0.03	0.61	0.82	0.26-1.07	0.29
Leaf 10	-	0.04	1.13	1.22	0.73-1.95	0.26
Leaf 11	0.03	0.04	0.74	0.93	0.28-1.21	0.26
Leaf 12	0.04	0.01	0.71	0.47	0.44-0.91	0.08
Leaf 13	0.04	0.22	1.08	1.50	0.57-2.09	0.35
Leaf 14	0.01	0.04	0.70	0.76	0.28-1.04	0.15
Leaf 15	0.06	0.03	0.63	0.37	0.43-0.80	0.07

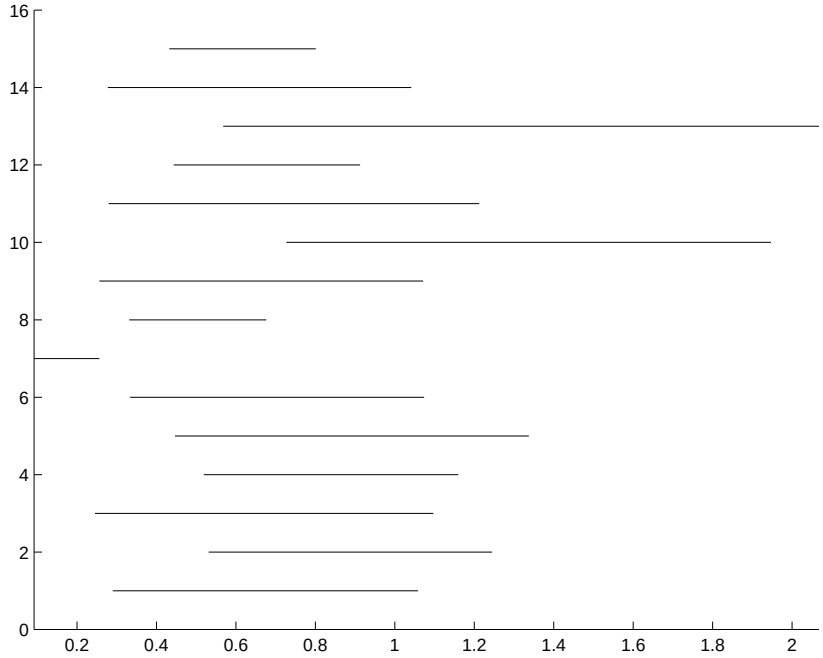

Table 6.10: A table of the results from the Hu1.

Conclusions from table 6.10

This descriptor gives low values for S_h and S_l and the separation between the tree classes is not good, even though it can separate between some of the classes as we can see in the diagram. Tree class 7 has significantly lower values and will therefor be easy to classify.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.03	0.01	0.77	1.18	0.12-1.30	0.24
Leaf 2	0.47	0.15	1.16	2.24	0.46-2.70	0.47
Leaf 3	0.01	0.43	0.53	1.34	0.08-1.42	0.43
Leaf 4	0.25	0.03	0.79	1.48	0.35-1.83	0.36
Leaf 5	0.17	0.20	0.99	1.87	0.30-2.18	0.37
Leaf 6	0.08	0.01	0.67	1.41	0.17-1.58	0.29
Leaf 7	27.54	-	0.01	0.06	0.009-0.07	0.007
Leaf 8	0.76	0.97	0.30	0.54	0.18-0.72	0.08
Leaf 9	0.19	0.05	0.56	1.32	0.09-1.41	0.42
Leaf 10	0.03	0.32	1.71	3.73	0.70-4.43	0.78
Leaf 11	0.04	0.08	0.73	1.51	0.11-1.62	0.38
Leaf 12	0.01	0.04	0.76	0.81	0.33-1.14	0.15
Leaf 13	-	0.02	1.75	4.82	0.53-5.34	1.03
Leaf 14	0.01	0.16	0.67	1.15	0.10-1.26	0.24
Leaf 15	0.05	0.01	0.53	0.54	0.26-0.80	0.11

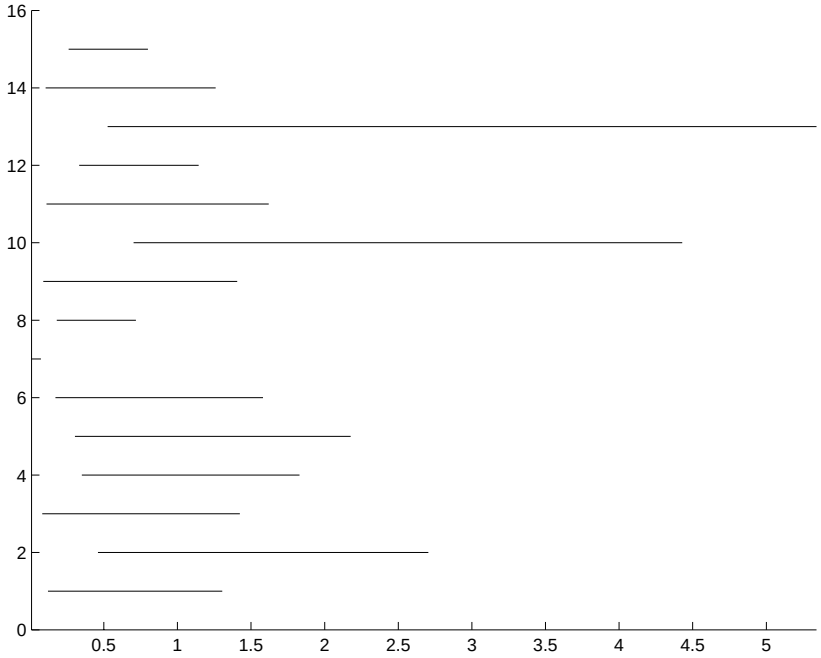

Table 6.11: A table of the results from the Hu2.

Conclusions from table 6.11

This descriptor gives low values for S_h and S_l and the separation between the tree classes is not good, but for tree classes 10 and 13 we get very high values which indicates that these to would benefit from using this descriptor. Tree class 7 can be classified with this descriptor alone.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.02	0.03	4.67	10.75	0.22-10.97	2.65
Leaf 2	0.05	0.17	5.61	18.22	0.56-18.79	3.35
Leaf 3	0.24	0.12	3.66	11.10	0.09-11.19	3.84
Leaf 4	0.03	0.08	2.32	9.61	0.48-10.09	1.84
Leaf 5	0.15	0.78	1.22	5.24	0.06-5.29	0.93
Leaf 6	0.09	0.03	2.13	6.36	0.09-6.45	1.66
Leaf 7	39.00	-	0.007	0.07	0.003-0.07	0.008
Leaf 8	3.50	0.98	0.27	0.90	0.09-0.99	0.16
Leaf 9	0.03	0.19	4.52	17.31	0.18-17.50	4.77
Leaf 10	-	0.81	30.9	142.2	3.04-145.27	25.44
Leaf 11	0.35	0.03	2.39	6.20	0.06-6.27	1.63
Leaf 12	0.03	0.32	2.06	3.68	0.44-4.11	0.82
Leaf 13	4.23	0.05	5.91	25.79	0.27-26.06	6.12
Leaf 14	0.14	0.26	3.22	11.30	0.11-11.41	2.13
Leaf 15	0.46	0.13	1.41	4.10	0.33-4.42	0.66

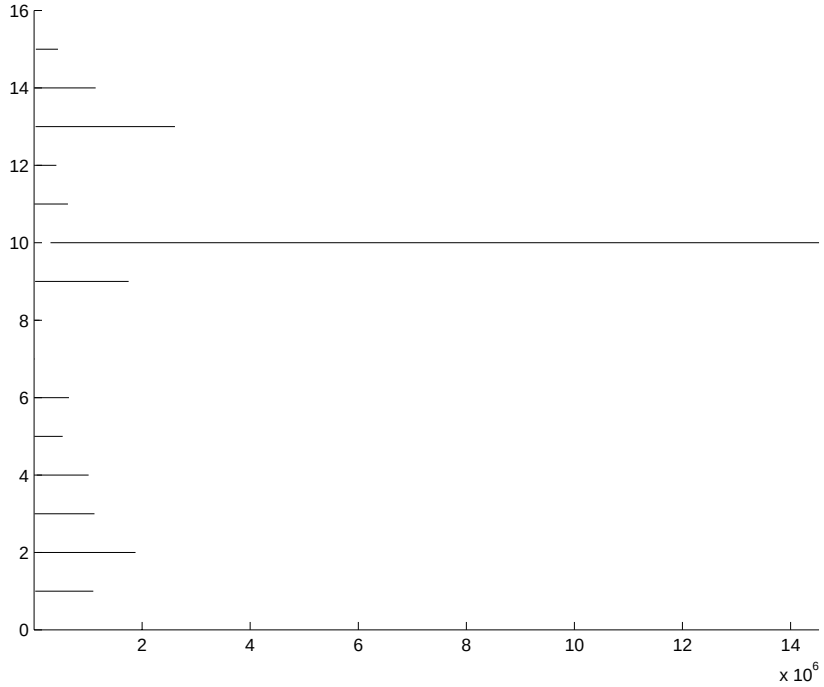

Table 6.12: A table of the results from the Hu3. All values, except S_h and S_l, should be multiplied with 10^5

Conclusions from table 6.12

Hu3 has fairly good values for S_h and S_l but the separation between the tree classes is not good. Tree class 7 can be classified with only this descriptor because it has lower values than any one else. Tree class 10 and 13 get higher values than most of the others and will there for be useful when we classify those two.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.39	0.04	5.38	11.99	0.29-12.27	2.93
Leaf 2	2.82	0.23	9.70	28.09	1.34-29.43	5.14
Leaf 3	0.26	0.04	4.10	12.71	0.11-12.82	4.25
Leaf 4	0.09	0.005	2.76	10.47	0.64-11.11	2.04
Leaf 5	0.18	0.64	1.48	5.82	0.10-5.92	1.03
Leaf 6	0.005	0.02	2.75	7.43	0.21-7.63	1.95
Leaf 7	70.98	-	0.007	0.08	0.003-0.08	0.009
Leaf 8	1.77	0.99	0.54	1.50	0.20-1.70	0.24
Leaf 9	0.04	0.21	5.17	19.30	0.24-19.55	5.32
Leaf 10	-	0.74	37.05	156.0	4.23-160.3	28.92
Leaf 11	0.02	0.35	2.69	6.83	0.08-6.91	1.81
Leaf 12	0.32	0.08	3.00	4.78	0.79-5.56	1.10
Leaf 13	0.30	0.28	7.45	28.31	0.53-28.84	6.89
Leaf 14	0.04	0.24	3.95	12.38	0.13-12.51	
Leaf 15	0.53	0.16	1.76	4.43	0.52-4.95	0.75

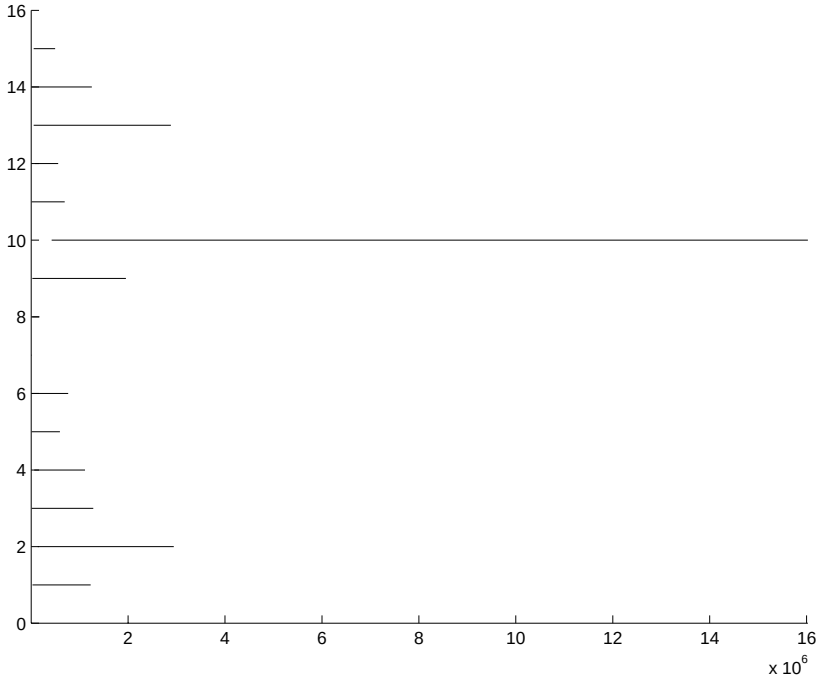

Table 6.13: A table of the results from the Hu4. All values, except S_h and S_l, should be multiplied with 10^5

Conclusions from table 6.13

This descriptor has rather high values for S_h and S_l. Tree class 10 gets very high values, tree class 7 gets very low values and the rest of the classes gets more or less the same values. This is therefor a poor descriptor.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.47	0.06	0.35	1.42	0.007-1.42	0.33
Leaf 2	0.01	0.44	0.92	6.91	0.012-6.92	1.09
Leaf 3	0.07	0.04	0.33	1.54	0.0001-1.54	0.43
Leaf 4	0.77	0.07	0.11	1.17	0.004-1.18	0.19
Leaf 5	0.10	0.92	0.03	0.33	0.0001-0.33	0.05
Leaf 6	0.08	0.02	0.10	0.54	0.0003-0.54	0.12
Leaf 7	1984	-	0.00005	0.0001	0.00005-0.0001	0.00005
Leaf 8	11.0	1.00	0.003	0.02	0.0003-0.02	0.003
Leaf 9	0.80	0.32	0.51	3.62	0.0005-3.62	0.78
Leaf 10	-	0.95	20.25	244.38	0.15-244.53	36.77
Leaf 11	0.02	0.14	0.10	0.45	0.0001-0.45	0.10
Leaf 12	0.17	0.61	0.08	0.26	0.005-0.27	0.06
Leaf 13	20.65	0.01	0.94	7.90	0.002-7.90	1.74
Leaf 14	0.71	0.43	0.19	1.50	0.0002-1.50	0.26
Leaf 15	1.17	0.09	0.03	0.23	0.002-0.23	0.03

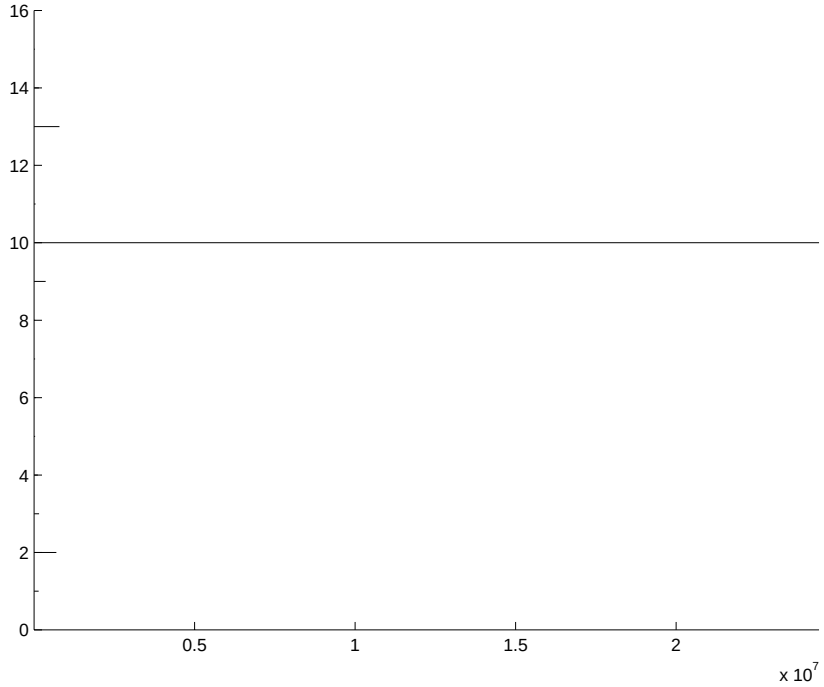

Table 6.14: A table of the Results from the Hu5. All values, except S_h and S_l, should be multiplied with 10^5

Conclusions from table 6.14

This descriptor has rather high values for S_h and S_l. Tree class 10 gets very high values, tree class 7 gets very low values and the rest of the classes gets more or less the same values. This is therefor a poor descriptor.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.03	0.17	4.20	10.62	0.08-10.70	2.60
Leaf 2	0.31	0.41	7.34	31.96	0.58-32.54	5.43
Leaf 3	0.21	0.19	3.48	12.44	0.03-12.46	3.90
Leaf 4	0.01	0.13	2.25	11.87	0.32-12.19	2.31
Leaf 5	0.47	0.20	1.32	7.21	0.04-7.24	1.29
Leaf 6	0.15	0.003	1.95	6.27	0.06-6.33	1.66
Leaf 7	237.06	-	0.0008	0.02	0.0003-0.02	0.0002
Leaf 8	4.34	1.00	0.20	0.78	0.06-0.84	0.13
Leaf 9	0.69	0.03	4.34	20.10	0.06-20.17	5.05
Leaf 10	-	0.79	45.32	297.43	2.73-300.16	49.87
Leaf 11	0.24	0.01	2.28	7.18	0.02-7.20	1.74
Leaf 12	0.003	0.32	1.95	3.73	0.30-4.02	0.84
Leaf 13	3.72	0.24	9.59	53.06	0.26-53.32	12.52
Leaf 14	0.23	0.19	2.82	12.19	0.04-12.22	2.35
Leaf 15	0.25	0.81	1.06	3.61	0.19-3.80	0.56

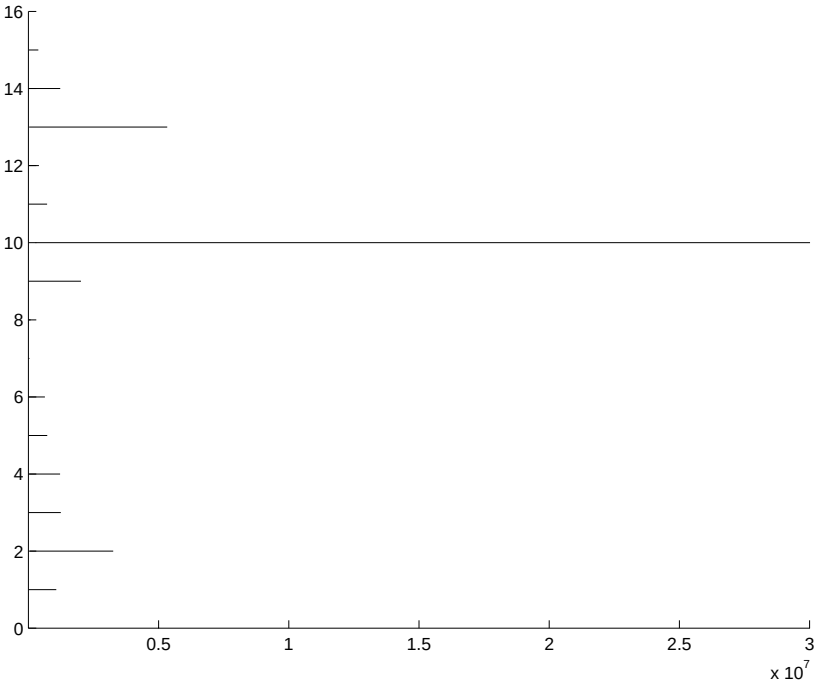

Table 6.15: A table of the results from the Hu6. All values, except S_h and S_l, should be multiplied with 10^5

Conclusions from table 6.15

This descriptor has rather high values for S_h and S_l. Tree class 10 gets very high values and tree class 7 gets very low values. the rest of the classes can not be classified very well with this descriptor but they do not get the same averages, M, which indicates that this descriptor may be useful.

$Name$	S_h	S_l	M	$Width$	$Low-high$	σ
Leaf 1	0.35	0.06	0.40	1.42	0.002-1.43	0.35
Leaf 2	-1.00	-	-1.01	8.23	-5.89-2.34	1.46
Leaf 3	0.16	0.30	0.32	2.23	0.0002-2.23	0.49
Leaf 4	0.44	0.43	0.23	1.48	0.01-1.50	0.28
Leaf 5	0.84	0.93	0.04	0.22	0.0007-0.22	0.04
Leaf 6	0.75	0.20	0.13	0.51	-0.002-0.50	0.12
Leaf 7	1217.9	423420.9	0.00005	0.0001	0-0.0001	0.00005
Leaf 8	12.65	1.00	0.003	0.07	-0.02-0.06	0.009
Leaf 9	0.71	0.26	0.54	3.29	0.0008-3.29	0.77
Leaf 10	-	0.98	40.71	353.05	0.74-353.79	61.35
Leaf 11	0.03	0.27	0.10	0.34	0.0002-0.34	0.09
Leaf 12	0.24	0.03	0.10	0.50	-0.02-0.34	0.08
Leaf 13	43.26	0.42	0.92	7.54	-0.77-6.77	1.38
Leaf 14	0.06	0.14	0.37	2.20	0.0003-2.20	0.38
Leaf 15	0.38	0.46	0.07	0.23	0.01-0.24	0.05

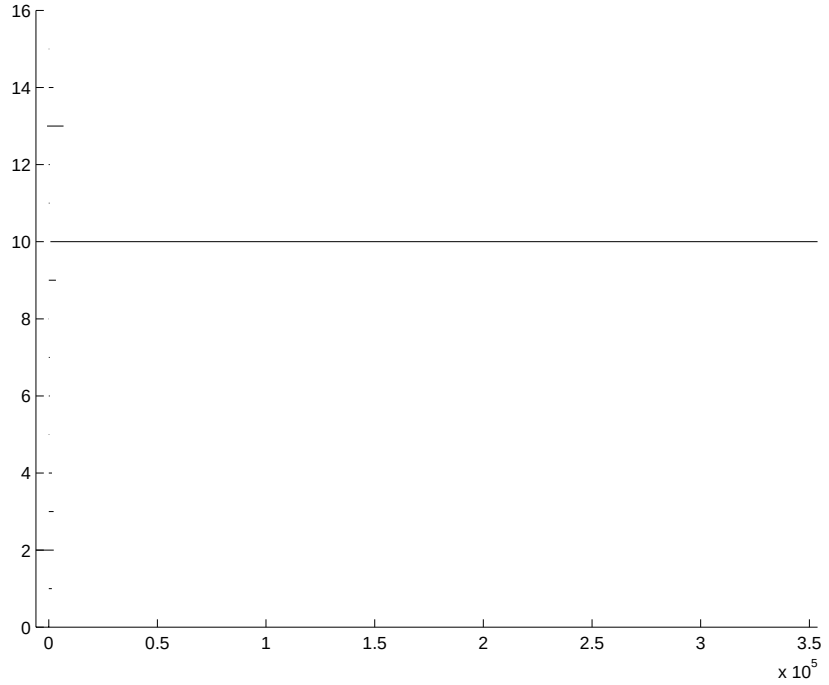

Table 6.16: A table of the results from the Hu7. All values, except S_h and S_l, should be multiplied with 10^3

Conclusions from table 6.16

This descriptor has average values for S_h and S_l. If we look at the min-max values we see that the classes have different values, the average value, M, is different for most of the classes. This indicates that this is a good descriptor.

6.6 Circularity

Implementation

8-connectivity was chosen when the perimeter was calculated. Except from this, no change in the descriptor has been done.

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.05	0.009	33.76	18.31	24.66-42.97	3.49
Leaf 2	0.41	0.03	36.49	54.33	0-54.33	7.08
Leaf 3	0.03	0.04	35.33	14.82	28.16-42.98	3.12
Leaf 4	0.16	0.04	28.13	20.01	19.78-39.78	4.58
Leaf 5	0.07	-	15.60	6.70	12.37-19.08	1.29
Leaf 6	0.02	0.14	32.66	28.22	24.03-52.24	4.31
Leaf 7	-	0.72	726.76	736.66	218.57-955.22	105.18
Leaf 8	0.17	0.13	19.83	8.72	16.40-25.12	1.80
Leaf 9	0.009	0.02	33.47	13.91	26.48-40.39	3.20
Leaf 10	2.56	0.75	203.99	165.07	127.25-292.32	31.37
Leaf 11	0.17	0.14	23.18	13.10	17.53-30.62	3.41
Leaf 12	0.04	0.14	27.04	11.13	23.52-34.65	2.12
Leaf 13	0.16	0.03	17.16	6.70	14.54-21.24	1.28
Leaf 14	2.95	0.29	51.63	57.56	25.35-82.91	12.23
Leaf 15	0.03	0.07	16.73	6.63	13.82-20.46	1.07

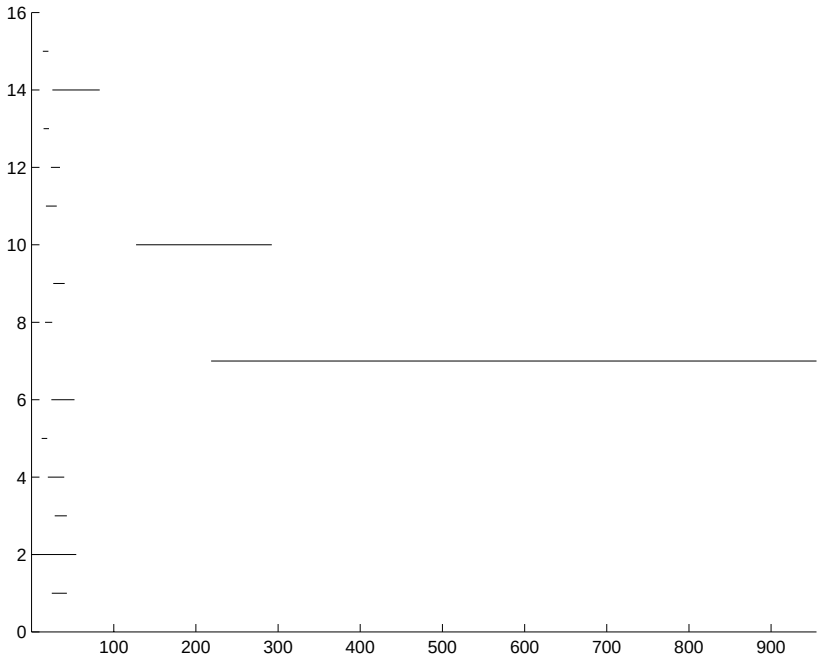

Table 6.17: This is the average height of the circularity.

Conclusions from table 6.17

This descriptor gives us low values S_h and S_l from most of the tree classes, but we can see that it gives us a good separation between the different tree classes if we look at the min-max values. The width is also low for most classes, and all this indicates that this is a good descriptor.

6.7 The Incremental Circle Transform

Implementation

For the ICT has 8-connectivity been choosen. The radie, r, in the circle transform has been chosen to 10.

Conclusions from table 6.18

This descriptor gets average values for S_h and S_l but as we see when we study the Lowest-highest values. The average values are different for most of the descriptors why we may conclude that this descriptor may be useful.

Conclusions from table 6.19

This descriptor gets medium values from S_h and S_l but as we see when we analyze the lowest-highest values, we see that the descriptor could be useful for most classes.

6.8 The structure of the object recognition

6.8.1 Tree

This is of course a very fast and good way if we know that the limits for each group are correct. It is a very good method if we have a small number of objects that we compare between, for example if we want to compare two or three leaves. But it is a very difficult method when the number of groups expands and to optimize the tree classification borders We will therefore turn our interest to feed-forward networks for a self-adapting approach.

6.8.2 Feedforward networks

A feed forward network with one hidden layer, trained with backpropagaition with a sigmoid function as a squash function and the standard error function was chosen. Different number of nodes in the hidden layers have been tried out. As can be seen in figure 6.20, where different number of hidden nodes has been compared can we see that it is not very important to have a large number of nodes, the only system that differs is the system with five nodes. This system shows a significantly worse result than the rest of the system that have 25, 45, 65, 85 and 105 nodes. As can be seen in figure 6.21, the procentage of accurate answers is increasing for all the systems , and the only system that shows worse results than the others are the system with five nodes. We could therefore conclude that it is enough with 25 nodes, but if we look at it more

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0.04	0.008	131.71	293.97	39.80-333.17	64.02
Leaf 2	0.65	0.78	933.9	1773.1	0-1773.1	410.6
Leaf 3	0.23	0.04	137.54	138.22	61.81-200.03	31.28
Leaf 4	0.21	0.16	96.54	125.81	42.08-167.88	31.21
Leaf 5	0.78	-	37.38	51.60	16.90-68.50	10.11
Leaf 6	0.04	0.07	125.18	212.42	29.45-241.88	52.23
Leaf 7	0.008	0.003	130.71	342.47	69.92-412.39	43.87
Leaf 8	0.003	0.04	130.30	192.87	42.36-235.23	45.16
Leaf 9	0.11	0.01	187.05	256.52	78.30-334.82	51.62
Leaf 10	-	0.39	1542.8	2613.4	419.1-3032.5	598.7
Leaf 11	0.21	0.44	66.69	73.36	31.87-105.23	18.50
Leaf 12	0.11	0.19	168.95	256.38	57.03-313.41	55.45
Leaf 13	0.07	0.17	116.94	167.44	57.39-224.83	41.33
Leaf 14	3.51	0.10	207.28	366.23	90.75-456.98	78.78
Leaf 15	0.19	0.17	80.82	121.38	24.49-145.87	25.94

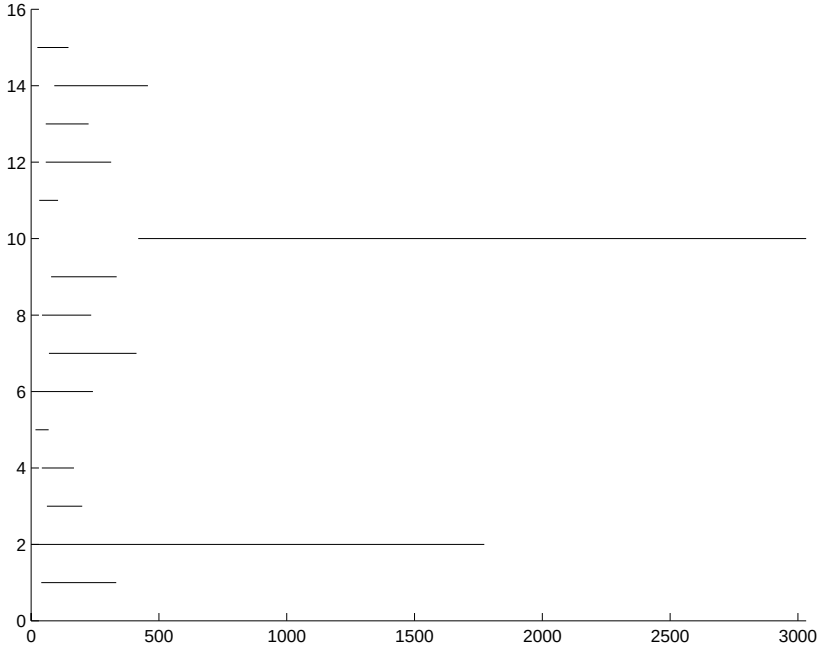

Table 6.18: A table of the results from the first eigenvalue of the ICT

closely we can see that it converge faster if we use a larger number of nodes. The Quickprop method have been tried out but it converge very slow.

6.9 Conclusion about the implementation

The implementation has worked well. Some of the descriptors were not so good; The moments did not, in general, work very well. But the CSS method and the ICT looked OK, and the test-result as a whole looks reasonable. The problem

$Name$	S_h	S_l	M	$Width$	$Low - high$	σ
Leaf 1	0	0.007	76.8	161.6	27.9-189.5	33.9
Leaf 2	2.90	0.67	635	1242.8	0-1242.8	261.0
Leaf 3	0.007	0.29	76.2	82.2	35.7-117.9	17.2
Leaf 4	0.09	0	76.8	134.2	27.8-162.0	30.1
Leaf 5	0.47	-	17.2	21.0	8.4-29.4	5.2
Leaf 6	0.04	0.05	87.7	156.8	25.2-182.0	35.1
Leaf 7	0.37	0.04	26.3	45.0	11.8-56.7	9.0
Leaf 8	0.05	0.08	83.5	144.6	25.1-169.8	30.1
Leaf 9	1.28	0.02	93.4	165.9	47.0-213.0	31.8
Leaf 10	-	0.74	2475.6	3986.0	558.0-4550	952.9
Leaf 11	0.04	0.32	25.3	38.1	7.2-45.3	8.6
Leaf 12	0.02	0.04	91.3	130.8	30.4-161.2	27.1
Leaf 13	0.04	0.34	54.2	84.1	18.0-102.1	20.2
Leaf 14	1.99	0.56	212.6	548.5	67.8-616.3	108.8
Leaf 15	0.51	0.27	36.0	50.2	10.5-60.7	10.6

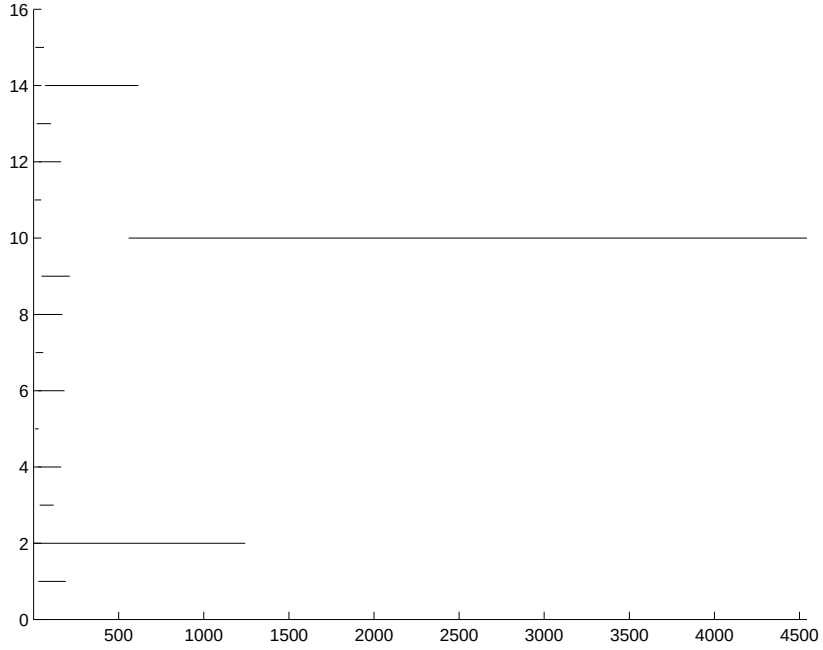

Table 6.19: A table of the results from the second eigenvalue of the ICT

is to see how far this can take us.

If we look at each of the descriptors, each descriptor can divide the groups of tree classes in three groups (in average). If we have 20 descriptors we would get an upper bound of the number of different groups of tree classes that we are allowed to have:

N_{groups} = Number of groups that each descriptor gives ≈ 3

N_{desc} = Number of descriptors=20

$N_{max} = {N_{group}}^{N_{desc}} = 3^{20} \approx 3.5 * 10^9$

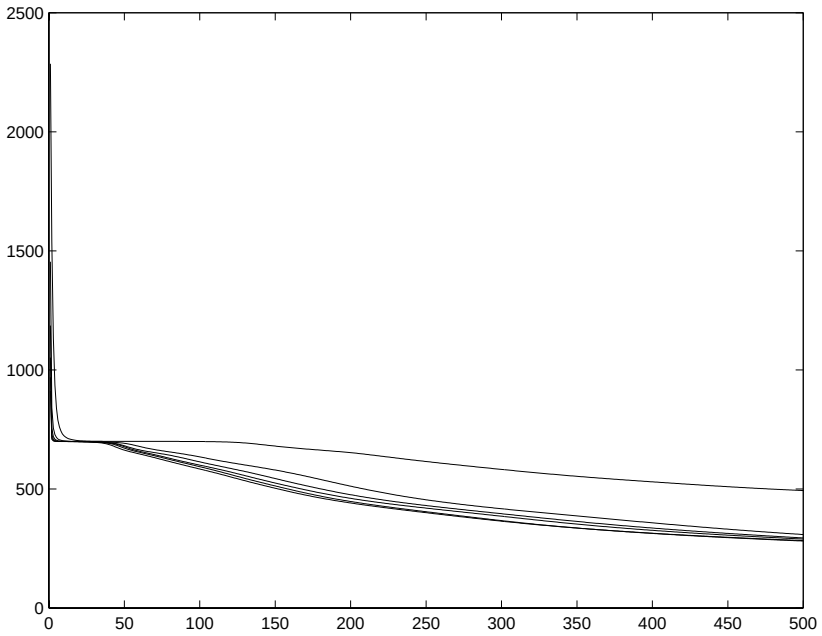

Table 6.20: This diagram shows the errorcurve for the backpropagation. The horizontal axis is the number of iteration and the vertical the error.

As everyone understand, this is just something that we are not going to even come close to. Why?
Even though the leaves are different, some of them come for the same group. And they will, as we can see, have the same values, or close, from each descriptor but their will always be groups that are closely related that will receive the same values because of the way that leaves evolve. Because of this the leaves will have closely related values from the descriptors. The task for the ones that will continue this exercise will be to separate all the subgroups from each other i.e. builds descriptors that can. And always make sure that there are not two tree classes that will give the same value for all descriptors.
Problem will occur when the different subgroups of leaves get too big and two classes are indistinguishable, then it will be up to the biologist and/or the engineer to find a descriptor that can obtain results that will divide the group in question.

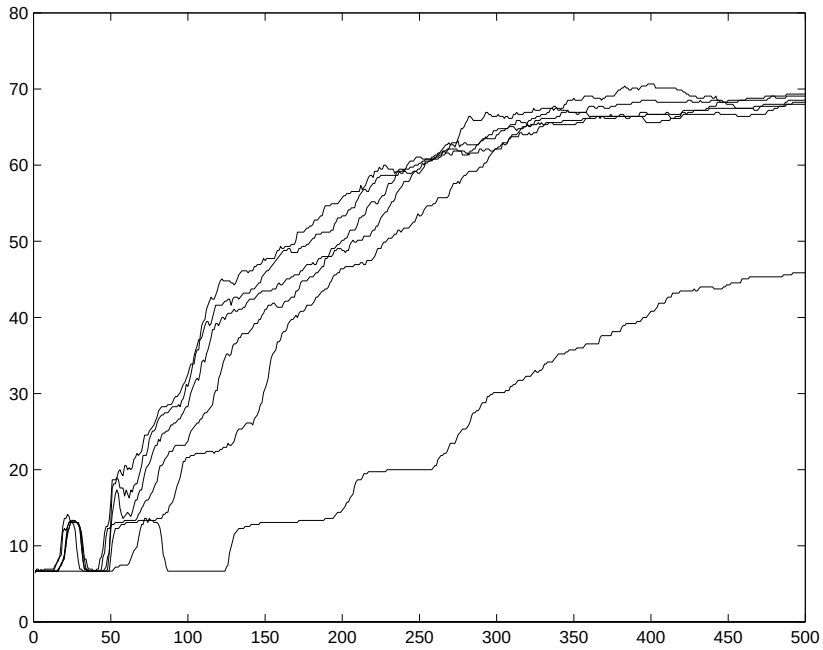

Table 6.21: This diagram shows the procentage of correct answers that we get from the system with the samples that we have not trained the system with as input values.

Chapter 7

Presentation of the Final System, Summary and Conclusion

7.1 Execution time and memory usage

The time demand is extremely dependent on the size of the image that we use as input value. The time demand varies therefore between a few seconds up to some minutes. This can probably be lowered if the code was implemented in c. And it is also very dependent on what how much RAM memory that we have, because if we start to use some slower memory then speed will drop very fast. It is also dependent on how many nodes we use and other similar variations. The memory storage demand is dependent on which method we use (Backpropagation, R-prop, tree, etc). But it will never be above 100*100 bytes, because of the descriptors that just leave a small (<10) feature vector that describes the leaf, instead of, for example, sending the whole image.

7.2 The System structure

To build a system we first need to build the structure. We will therefore first go through the system, section by section. Then finally study how the system works when running. The system has four parts, which we have studied before. We will see here which parts that have been implemented in the system and how they have been implemented. The four parts are:

- Preprocessing of the image
- Describe the leaf with the descriptors
- Preprocessing, for the classification, of the descriptors values
- The classification system

There are not distinct lines between all of these parts. But we need the structure to get an idea how this is built.

7.2.1 Preprocessing of the Image

The preprocessing that has been used can be divided into six steps.

- Threshold the image to get a B/W-image
- Identify the leaf in the image
- Remove all other objects than the leaf
- Remove all the holes in the leaf
- Crop the picture to a minimal rectangle without removing any part of the leaf
- Calculate the border of the leaf
- Sample the border of the leaf

We will now study the different parts more in detail.

Threshold

The threshold that has been chosen is 0.8 from a scale of 0 to 1. This has been found to work with trial and error rather than with any exact measurements. This maybe one of the weakest links for the whole system because different scanners may use different light and different leaves may affect the light of the picture. This can be studied further in [23].

Identify the leaf and remove all other Objects

This has been done with the technique described in 4.2.5.

Remove all the holes

To remove all the holes we just need to remove all objects that don't have any border to the background. This can be studied further in [11].

Reduce the picture

First we will identify the leafs left, right, top and low borderpoints in the image and then cut the picture exactly above and below those points.

Calculating the border and taking samples from it

First we need to find a point on the leaf surface and then we try to find the edge of the leaf and after that we go around the border and save all the points as border points. Here we chose between 4- and 8- connectivity. In our special case we will use 8 connectivity to get the sampling points.

7.2.2 Describe the leaf with the descriptor

The descriptors are fully explained in the previous chapters so we will just see which descriptor that has been used. The descriptors have been selected because of what output they give. For example the MFD-descriptor compares the leaf with all the leaves that are in its data base and calculates the distance. This gives a large database and also many output-values and we have to compare the leaf with all samples of leaves to find the right tree class. If we compare this with eccentricity that gives us an output value, it is clear which type of descriptors that we prefer. We chose these descriptors :

- Area
- Circularity
- Flusser-moments
- Hu-moments
- Eccentricity
- Curvature Scale Space
- Incremental Circle Transform

7.2.3 Preprocessing of the descriptor values

As a preprocessing for the classification system rescaling has been chosen as described in 5.2.

7.2.4 The Classification System

A feed forward network with one hidden layer was chosen. To build it a sigmoid function of squash type and the standard error function was used. Different types of variations have been tried out on the classification system but one larger system with a large number ($10000 <$) training cycles have been used. This net has one input layer with one node for each descriptor, one hidden layer with 100 nodes and finally 15 nodes in the output layer, one for each tree class. This system was trained with the Backpropagation algorithm.

7.2.5 Teaching and Using the System

Teaching the system

We will here study how we teach the system with one leaf. And we will just do one cycle of training but what we normally do is to train all leaves, one time, in a training cycle and do the training cycle many hundred times until we have reached the limit for the error. The training cycle is illustrated in figure 7.2.5.

What we first do in the training cycle is to preprocess the leaf, as described in section 7.2.1, and then let the descriptors calculate their description of the leaf as described in section 7.2.2, then will we preprocess the descriptor valuesas described in 7.2.3, and give the learning system the values. But we will also give the learning system the output values that we want from the network and this will we do for all the leaves in a cycle.

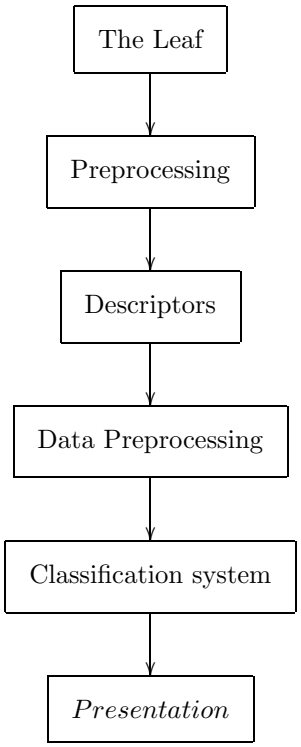

Figure 7.1: This is a figure that shows the main structure of the system where we start with the input, the leaf. Continue with the preprocessing methods, we get the features from the descriptors. We preprocess the valus we get from the descriptor, and finally we classify the leaf

Using the System

The post-process part that presents the values to us in comprehensible and informative way. In this system will the information be presented straight forward by writing the most likely tree class. We will study the result from this presentation below.

7.3 Results from the System

tree class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	52	0	12	0	0	8	0	0	12	0	12	0	0	0	4
2	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0
3	4	0	72	0	0	0	0	0	12	0	12	0	0	0	0
4	0	0	0	88	0	0	0	0	0	0	0	0	0	12	0
5	0	0	0	0	88	4	0	0	0	0	0	4	0	0	4
6	0	0	0	0	4	88	0	0	0	0	0	0	8	0	0
7	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0
8	0	0	0	0	4	8	0	76	0	0	0	12	0	0	0
9	16	0	12	0	0	0	0	0	76	0	4	0	0	0	0
10	0	0	0	0	0	0	0	0	0	88	0	0	0	12	0
11	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0
12	0	0	0	0	4	8	0	0	0	0	0	88	0	0	0
13	0	0	0	0	0	0	4	0	4	0	0	0	92	0	0
14	0	0	0	48	0	12	0	0	0	0	0	0	4	36	0
15	0	0	0	0	0	0	0	0	0	0	8	0	0	0	92

Table 7.1: A table of the results from the system. The vertical axis is numbered from 1 to 15, that is the tree class from which the leaf is taken that we give the system. The horizontal axis is the tree class that the system give us, as an answer.

As we can see in table 7.1, we have a system that doesn't give us a hundred percent correct answer for all the tree classes. For some we do get a hundred percent, and for the worst case, we get below fifty. For tree class 1 and 14 the system is not sufficient. It needs at least one new descriptor that separates them from the other leaves that they are mixed up with. We also see that tree class 1 gets mixed up with tree class 3, 6, 9, 12 and 15, and tree class 14 gets mixed up with 4, 6 and 13. So even though tree class 14 gets a worse result, it may be easier to solve that than for tree class 1.

7.4 Summary

This thesis describes how to chose descriptors and system for the purpose of building a computer vision classification system for leaves. The descriptors are normal computer vision descriptors, such as Moments, Area, curvature etc. A Feed-forward network have been chosen. Each descriptor, and the whole system are evaluated with fifteen sorts of leaves and there are 75 samples from each sort. It is shown that there is no single descriptor that can fulfill the task but instead

if we take all of the descriptors as input values in a Feed-forward network and use the network as a classifier, then all the values together will show that each tree class will be in the same area in the descriptor space.

7.5 Conclusion

From my experience and from the result of my program I conclude that it is possible to build a system that identifies the tree class from a scanned picture of one leaf from that tree classes showed in figur 6.1 to 6.5. This program does classify twelve out of fifteen tree classes, more or less, correctly, but with two or three more descriptors that are 'hand made' to separate the two worst classes, tree class 1 and 14, from the rest of the classes the system would work sufficient for all tree classes that were tested here. It is likely to do this for all of the different 40 tree classes in Sweden, but then we need even more descriptors, and also more samples. If one want to build a larger system with thousands of tree classes it is difficult to predict the result. It could be possible, but then it is important to evaluate the system and see if we have two leaves that give us the same answer for all the descriptors. It could also be interesting to build some sort of tree-structure to make the training phase faster. If we would build a system like that, it would probably be better to present, for example, the tenth most likely tree classes than just a single alternative.

Chapter 8

Future Applications and Developments

To produce better system in the future there are certain changes that could and need to be done. We will begin to list some problems that we may encounter, and afterwards describe possible solutions.

- Improve reliability
- Shorter classification time usage
- Collecting the leaf samples faster

Support Vector Machines

This method is a learning method for a feed forward network. The main difference between Backpropagaition learning and Support vector machine learning is that instead of just being satisfied that it has found a border between the classes of trees it will find the optimal border that is exactly in between the nearest samples from the tree classes.

Expertsystem

This is a system that uses more information than just the input leaf. It could for example ask the user what date the leaf was scanned and with that and the color information improve the system if for example the leaf gets some particular color during autumn. This could increase the precision i.e. the number of correct answers, but also the demanded time for the classification.

Fuzzy rules

Fuzzy rules can be defined in different ways, but here we define it as all systems that use certain types of rules. For example; if the leaf if bigger than $100cm^2$ then it is not a Populus leaf.This could increase the accuracy i.e. the number of correct answers, but also the usage of time for the classification.

Larger amount of leaves

The best way to increase the percentage of correct answers, except maybe to find better descriptors, is to increase the number of leaves used for the training. This statement is just valid up to a point, but because the leaves from the same tree class differs from different trees, different regions and different dates, we can collect a large number of them before we have seen all varieties.

Linear systems

We could save a lot of time if we used linear systems instead of nonlinear. The error if we just used linear system would be too large, but if we used it together with some sort of tree structure or/and together with RBF, it could be a possibility.

Lower the resolution in the image

If we lower the spatial resolution in the image we will save a lot of time but then we will also loose a large amount of information.

New descriptors

To increase the precision i.e. the number of correct answers, we will always have use for more and better descriptors.

Identification of Leaves from Photographic films and Video-cameras

This is probably the only way to do a perfect identification of leaf through computer vision. Partly because its the only way that you could (if we had at least two pictures) do a 3-D model of the leaf. To go from 2-D to 3-D modeling is not a simple task, but is still possible. This would also make it possible to know when and where the leaf has been filmed/photographed, which will make it a lot easier, for example, to use color identification. But the main reason for using photography instead of a scanner is that it is much easier and faster to collect the information.

Bibliography

[1] Farzin Mokhtarian Sadegh Abbasi and Josef Kittler. Robust and efficient shape indexing through curvature scale space. *British Machine Vision Conference*, 1999.

[2] URL: www.cs.bris.ac.uk/Research/Vision/search.html*ACRC*.

[3] Johan Wilkinson Alan Mitchell. *Träd i Europa*. Bonnier Fakta, 1983. ISBN 91-34-50299-8.

[4] URL: www.monetdb.cwi.nl/ acoi/Amis/projects.html*AMIS*.

[5] D. H. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–112, 1981.

[6] Christopher M. Bishop. *Neural networks for Pattern recognition*. Claradendon Press, 1992.

[7] M. Borga and H. Knutsson. Finding Efficient Nonlinear Visual Operators using Canonical Correlation Analysis. In *Proceedings of the SSAB Symposium on Image Analysis*, pages 13–16, Halmstad, March 2000. SSAB.

[8] Adam Finkelstein Charles E. Jacobs and David H. Salesin. Fast multiresolution image querying. Report, Department of C.S. and Engineering, University of Washington.

[9] M. Skytte Christiansen. *Svenska växter i färg*. Raben och Sjögren, 1989. ISBN 91-29-58386-3.

[10] Thomas Karlsson och Ebba Werner Dag Ekholm. *Vilda och Förvildade Träd och Buskar*. SBT-redaktionen, 1991.

[11] Per-Erik Danielsson. opt. In *Bildbehandling*, page 342. Linus o Linnea, 1997.

[12] Gregory Dudek and Johan K. Tsotsos. Shape representation and recognition from multiscale curvature. *Computer vision and Image understanding*, 68(2):170–189, November 1997.

[13] G. H. Granlund and H. Knutsson. *Signal Processing for Computer Vision*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9530-1.

[14] P. Nagabhushan D.S. Guru. Incremental Circle Transform and Eigenvalue Analysis for Object Recognition: an Integrated Approach. *Pattern Recognition Letters*, 21:989–998, 2000.

[15] URL: research.microsoft.com/users/yongrui/html/publication.html *MARS*.

[16] URL: min.ecn.purdue.edu/ jauyuen/webbrowse/webbrowse.cgi*MIDS*.

[17] F. Mokhtarian, S. Abbasi, and J. Kittler. Robust and efficient shape indexing through curvature scale space. In *Proceedings of the 1996 British Machine and Vision Conference BMVC'96*, Edinburgh, Scotland, September 1996.

[18] Farzin Mokhtarian and Alan K. Mackworth. A theory of Multiscale, Curvature-Based Shape Representation for planar curves. *IEEE Transactions on pattern analysis and machine intelligence*, 14(8), Aug 1992.

[19] Gunnar Nilsson. *Lövträd och buskar*. Svenska byggtjänst, 1983. ISBN 91-7332-173-7.

[20] Russel D. Reed and Robert J. Marks II. *Neural Smithing*. The MIT Press, 1999. ISBN 0-262-18190-8.

[21] John C. Russ. *The Image Processing Handbook*. CRC Press, 1999. ISBN 0-8493-2532-3.

[22] URL: www.lems.brown.edu/vision/researchAreas/SIID/index.html*SIID*.

[23] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. International Thomson Publishing Inc., 1999. ISBN 0-534-95393-X.

[24] Sergios Theodoris and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, 1999.

[25] Du-Ming Tsai and Ming-Fong Chen. Curve fitting approach for tangent angle and curvature measurement. *Pattern Recognition*, 27(5):699–711, 1992.

[26] Alfred C. She Yong Rui and Thomas S. Huang. A modified fourier descriptor for shape matching in mars. Report, Image Formation and Processing Lab, Beckman Institute University of Illinois.