Leaf Shape Identification Based Plant Biometrics

By

Javed Hossain

ID. 061 016 040

Under the supervision of

Dr. Ashraful Amin

May 2, 2010

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE IN COMPUTER SCIENCE

North South University

ABSTRACT

This dissertation presents a robust and computationally efficient method for plant species recognition using leaf image. The unique feature of this method is, it can differentiate simple leaves from compound leaves and identify the plant species of both type of leaves with minimum user interference. The method consists of five major parts which are- leaf image acquisition, image preprocessing, feature extraction, system training and system testing. First, images of leaf are acquired with digital camera or scanners. This method works only for the plants with broad flat leaves which are more or less two dimensional in nature. Then the image is filtered to get rid of the noises. After that the user selects the base point of the leaf and a few reference points on the leaf blades. Based on these points the leaf image is converted into a binary image and aligned horizontally with its base point on the left of the image. Then several morphological features are extracted. These features include area, perimeter, solidity, major axis, minor axis, equivalent diameter, convex area, filled area, extent and so on. A set of special features are extracted from simple leaves by slicing across the major axis and parallel to the minor axis. Then the feature pointes are normalized by taking the ratio of the slice lengths and leaf lengths (major axis). These features are used as inputs to the probabilistic neural networks. Two probabilistic neural networks has been used. The first network was trained with 1200 simple leaves from 30 different plant species and used for identifying the plant species of the simple leaves and the second network is trained with 200 compound leaves belonging to 5 plant species and used for identifying the species of the compound leaves. The proposed method has been tested using ten-fold cross-validation technique and the system shows overall 92.42% recognition accuracy.

LETTER FROM CANDIDATE

To

Dr. Miftahur Rahman,

Chairman, Department of Electrical Engineering and Computer Science,

North South University,

Dhaka - 1229,

Bangladesh.

I hereby declare that I know what plagiarism entails, namely to use another's work and to present it as my own without attributing the sources in the correct way. I know that plagiarism is a punishable offence because it constitutes theft. I understand the plagiarism policy of the Department of Electrical Engineering and Computer Science of North South University. I declare that all work presented by me in this dissertation, are my own, and where I have made use of another's work, I have attributed the source in the correct way.

Name: Javed Hossain

Signature _____

Date: April 25, 2010.

Witnessed by

Name: Dr. Ashraful Amin

Signature _____

Date:

DEDICATION

I would like to dedicate my work to

Bear Grylls; he is the biggest inspiration of my life.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Ashraful Amin for his continuous guidance throughout the thesis. It would not have been possible to finish this work without his support and wise supervision. I learnt a lot about the computer vision and image processing techniques from him. He, definitely, is one of the best teachers I have ever had.

I am much grateful to *Wu et al*. for making their dataset publicly available. They have done a tremendous job making this enormous dataset and uploading it on the Internet. I made my first prototype of this plant classification system using this dataset and in the final version this dataset contributed around 85% of the entire leaf image dataset.

Last but not the least I want to thank my parents for encouraging me to do what my heart says. Their support and inspirations have helped me stay focused on my work and not to give up. They are the most precious persons I will ever have.

Table of Contents

List of Figures and Tables

Chapter 1: Introduction

Chapter 2: Literature Review

Chapter 3: Description of the Method

Chapter 4: Implementation

Chapter 5: Testing and Evaluation

Chapter 1

Introduction

Plants play a critical role in preserving the delicate balance of the environment. Unfortunately, the overwhelming development of human civilization has disrupted this balance to a greater extent than we realize. It is one of our biggest responsibilities to save the plants from various threats, restore the diverseness of the plant community and put everything back to balance. A computerized plant identification system can be very helpful in botanical garden or natural reserve park management, new plant species discovery, plant taxonomy, exotic plant detection, edible/poisonous plant identification and so on. A computer based plant identification or classification system can use different characteristics of the flora, starting at very simple level such as: shape and color of the leaf, flower and fruit type, branching style, root type, seasonality, outlook, to very complex such as cell and tissue structure, DNA and genetic structure. However, a simplified approach which requires very little work by the user to identify the plant is our concern. Presently the cell phones are capable of acquiring high quality images with their integrated digital camera, which makes the usability of this system even wider. Adventurers, campers, trekkers or people in survival situation might also be able to make use of such a system.

The main objective of the project is to research and develop an intelligent system that can recognize a plant species when provided with an image of the leaf of that plant. The block diagram in Figure-1.1 shows the input and output of the system.

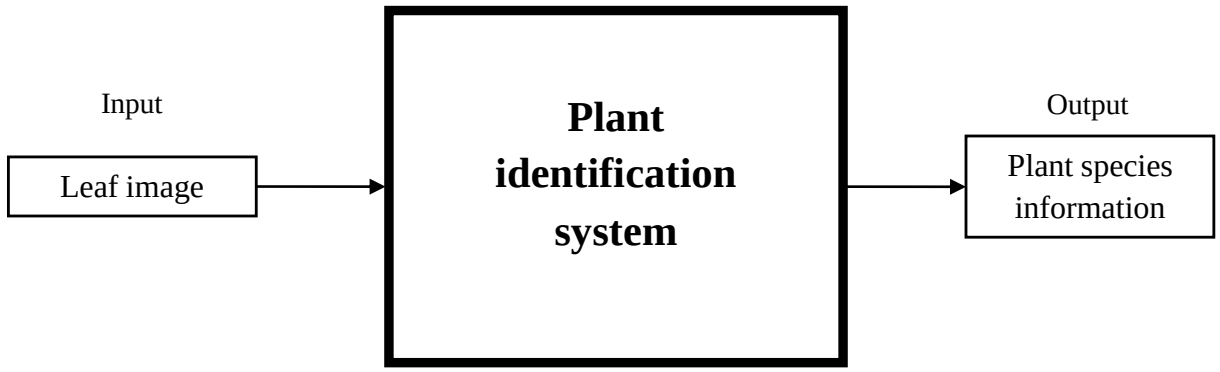


Figure 1.1 A simple block diagram of the system.

1.1 Plant Leaves

A major percentage of the plants are broadleaved, that is, having flat wide leaves which are almost two dimensional in nature. This property makes these leaves suitable for machine processing. Based on the structure of the blade these leaves can be categorized into the two following groups.

1. Simples leaves
2. Compound leaves

Simple leaves

A simple leaf has an undivided blade. However, the leaf shape may be formed of lobes, but the gaps between lobes do not reach to the main vein. Figure-1.2 shows different parts of a simple leaf.

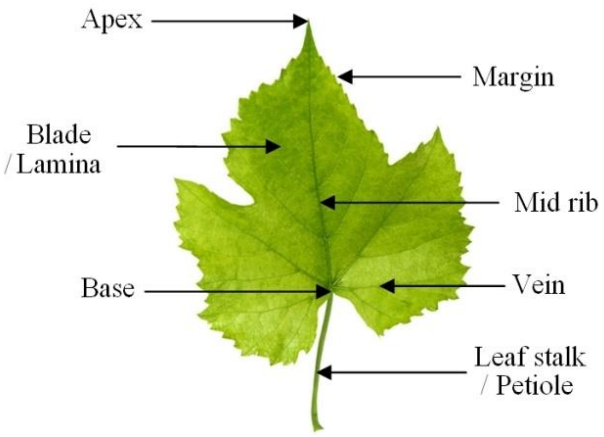


Figure 1.2 Parts of a simple leaf.

Compound leaves

A compound leaf has a fully subdivided blade where each leaflet (shown in Figure-1.3) of the blade separated along a main or secondary vein. The middle vein of a compound leaf, to which the leaflets are attached to, is called rachis.

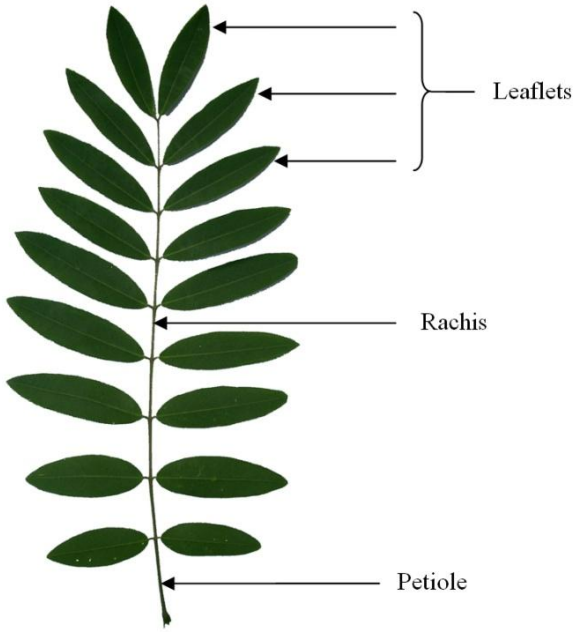


Figure 1.3 Parts of a compound leaf.

The major contribution of this work is a robust and computationally efficient method for plant species recognition from leaf image. The unique capability of our system is- it can differentiate simple leaves from compound leaves and identify their species. It can also identify the type of plant from a partially damaged or broken leaf as well. The method consists of five major parts which are- leaf image acquisition, image preprocessing, feature extraction, system training and system testing. First, images of leaf are acquired with digital camera or scanners. Then the system requires the user to select the base point of the leaf and a few reference points on the leaf blades. After preprocessing is complete, several morphological features, such as area, perimeter, solidity, major axis, minor axis, equivalent diameter, convex area, filled area, extent and so on were extracted from the leaves. A set of special features called Laminal Width Factor is extracted from simple leaves by slicing across the laminal major axis and parallel to the minor axis. Then the feature pointes are normalized by taking the ratio of the slice lengths and leaf lengths (major axis). These features are used as inputs to the probabilistic neural networks. 1200 simple leaves from 30 types of plants and 200 compound leaves from 5 types of plants have been used to train our system.

1.2 Structure of the Dissertation

Chapter 2 contains discussion on previous works done on plant identification or classification from leaf image. Different approaches to solve commonly faced problems are compared and analyzed. Techniques that have been adopted by other people are discussed. Limitations of various methods are also pointed out as well. Analysis of techniques which might be suitable for our purpose is also a major part of this chapter. Chapter 3 describes the method that we developed and applied to build our plant identification system. This chapter is subdivided

in to five mains sections. These sections describe, in detail, their corresponding steps followed by our method. Theories behind each technique or concept used in the method are discussed with necessary formulae and figures. Even alternative ways of solving particular problems are also explained and compared. The limitations of this method are mentioned along with the new features. Detailed information about the implementation of the proposed method is discussed in chapter 4. It reveals which of the techniques and solutions discussed in earlier chapter has been used for our system and which ones were not. The reasons behind giving preference to one solution over another are also discussed. Implementations of the techniques or solutions that have been applied to attain the objectives of the project are discussed in detail. Chapter 5 analyzes the performance of the system and presents them using tables, graphs and equations. Chapter 6 contains the conclusions drawn from the analysis and the work done in this project.

Chapter 2

Literature Review

A substantial amount of work has been done on leaf shape based plant classification and recognition. Many different approaches has been taken by different researchers. However, all the works dealt with simple leaves only. This chapter discusses and compares previously done works on leaf shape extraction based plant species recognition.

Wu et al. [1] extracted the 5 basic geometric features. They are diameter, physiological length, physiological width, leaf area and leaf perimeter. Some of these features are shown in Figure-2.1. Then, 7 commonly used digital morphological features (DMFs) were derived from these 5 basic features. In addition to that, 5 vein features were extracted. The DMFs are- smooth factor, aspect ratio, form factor, rectangularity, narrow factor, perimeter ratio of diameter, perimeter ratio of physiological length and physiological width. They used 1800 leaves to classify 32 kinds of plants.

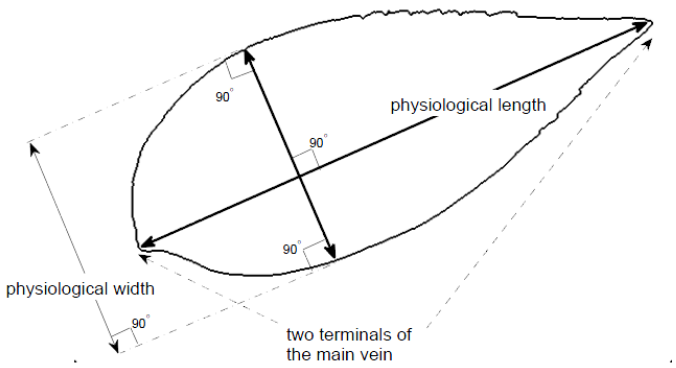


Figure 2.1 Basic geometric features of a leaf. [1]

Their experimental result indicates that the average accuracy of their algorithm is 90.312%. *Wang et. al.* [2] employed centroid-contour distance (CCD) curve, eccentricity and angle code histogram (ACH) for feature extraction. For leaf end-points detection they SUSAN algorithm.

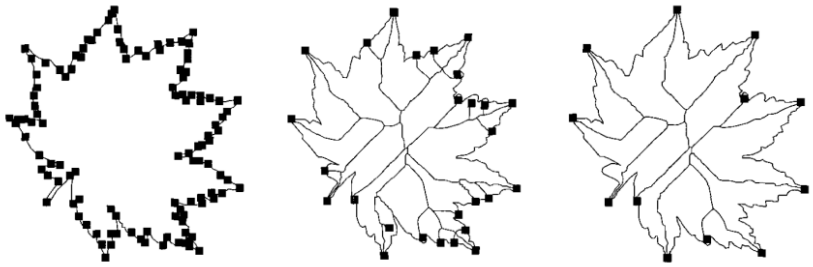


Figure 2.2 End-points of the leaf contour. [2]

Figure-2.2 shows end-points extraction of the leaf contour using SUSAN algorithm. To compare two CCD curves, the curves were first normalized by down-sampling the CCD curve which has more sample points. A down-sampling example is shown in Figure-2.3. Their experimental results on 1400 leaf images from 140 plants show that the proposed approach can achieve a better retrieval performance than both the curvature scale space (CSS) method and the modified Fourier descriptor (MFD) method.

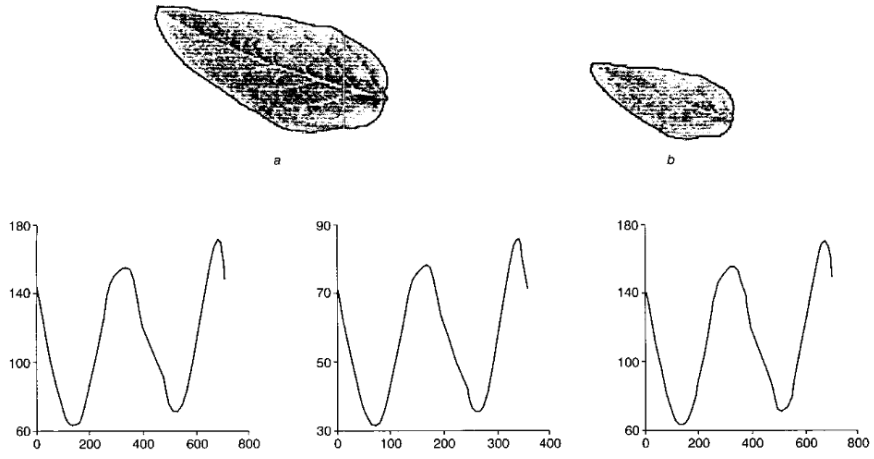


Figure 2.3 Down sampling CCD curves of two different sized leaves of the same plant. [2]

Fu et al. [3] also used centroid-contour distance curve to represent leaf shapes. They used 6 types of leaves, each type having 35 samples. They reached an average accuracy of 94.2%. *Wang et al*. [4] used an automatic marker-controlled watershed segmentation method combined with pre-segmentation and morphological operation to segment leaf images with complicated background based on the prior shape information. An example is given in Figure-2.4. Seven Hu geometric moments and sixteen Zernike moments were extracted as shape features from segmented binary images. They have used 1200 leaf samples corresponding to 20 classes of plants and their average classification rate is up to 92.6%

Figure 2.4 Segmentation of leaf image using automatic marker-controlled watershed method. [4]

Du et. al. [5] employed accelerated Douglas-Peucker approximation algorithm to extract leaf shape features. This algorithm finds the polygonal approximation of the contour, after which a leaf shape can be represented as a sequence of vertices (as shown in Figure-2.5). Using 2170 leaves from 25 types of plants their method showed an average recall rate of 92%.

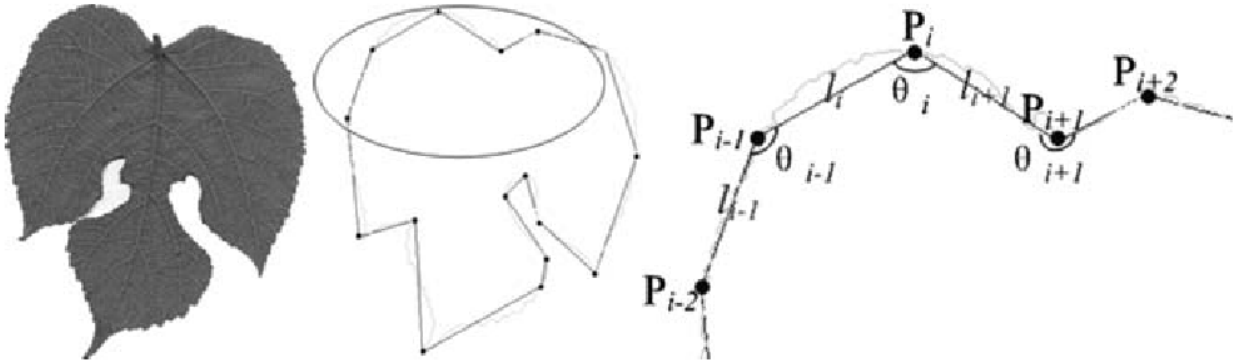


Figure 2.5 The polygonal approximation of a leaf contour. [5]

Ye et al. [6] used CCD (centroid-contour distance) curve to represent leaf shapes. *Li et al.* [7] applied snakes technique with cellular neural networks (CNN). *Gu et al.* [8] used the result of segmentation of leaf's skeleton based on the combination of wavelet transform (WT) and Gaussian interpolation, shown in Figure-2.6-7.

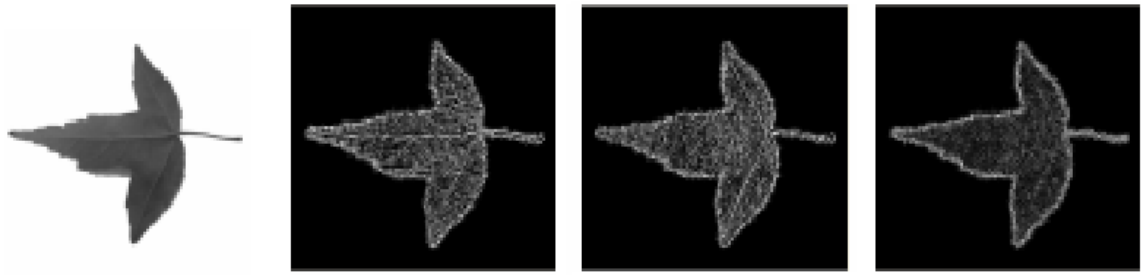


Figure 2.6 Decomposition of leaf image using wavelet transformation. [8]

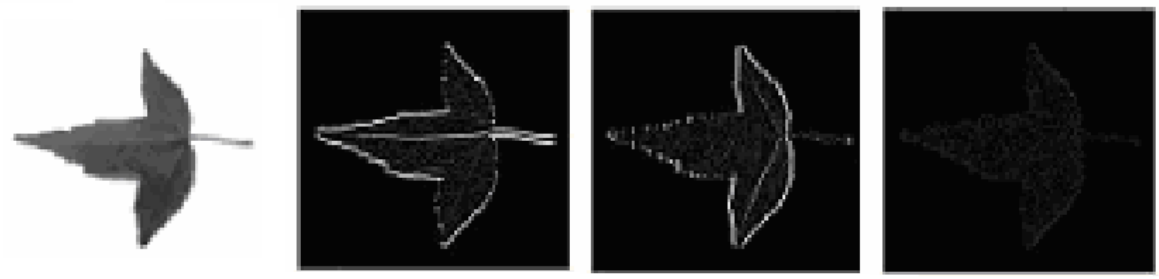


Figure 2.7 Adding Gaussian interpolation to wavelet decomposition. [8]

Wang et al. [9] also used marker-controlled watershed segmentation method for preprocessing, shown in Figure-2.8. After preprocessing several geometric features like rectangularity, circularity, eccentricity and seven moment invariants were extracted for classification. They used a dataset of 800 leaf images belonging to 20 different plant species. The average accuracy of their method had reached 92.2%.

Figure 2.8 Watershed segmentation and leafstalk removal. [9]

Some [1], [3], [10] approaches employed artificial neural network for its fast performance. Others [8], [9] employed k-nearest neighbor (k-NN) classifier to classify plants. *Du et al.* [10] introduced shape recognition based on radial basis probabilistic neural network which is trained by orthogonal least square algorithm (OLSA) and optimized by recursive OLSA. It performs plant recognition through modified Fourier descriptors of leaf shape. Mokhtarian and Abbasi [11] used curvature scale space image to represent leaf shapes and applied it to leaf classification with self-intersection.

Chapter 3

Description of the method

According to our method, the system will work its way through the steps shown in the flowchart in Figure-3.1. Each of these steps can have profound impact on the performance and behavior of the system and thus requires very meticulous analysis.

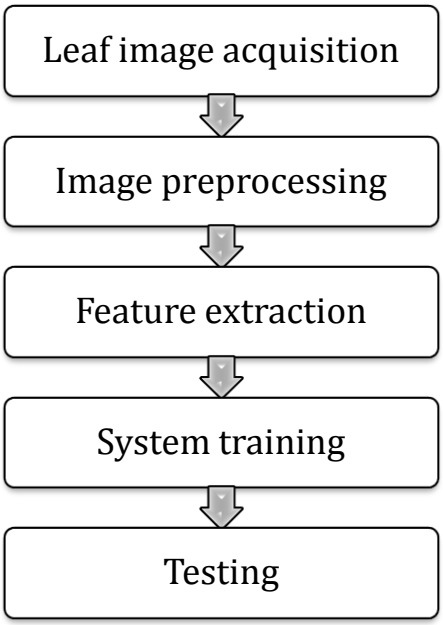


Figure 3.1 System flowchart.

The leaf image acquisition cannot be done by the system, it is the user of the system who will acquire the images and give the images to the system as inputs. The system will preprocess the images with the help of the user and extract necessary features. A set of features, extracted from

leaf images, will be used to train the system. Then, another set of features, extracted from a new set of leaf images, will used to test the system.

3.1 Leaf Image Acquisition

Leaf image acquisition is the first and one of the most essential tasks. Without a substantial number of leaf images of various plant species the proposed method would not be much useful. The easiest option for acquiring images of plant leaves is using digital cameras which, nowadays, have become integrated in most of the cell phones available today. Another potential option is scanning. However, scanners are not as portable as a digital camera. In addition to that, scanners are much slower compared to a typical digital camera. For our purpose, high quality image is not necessary at all. Instead, proper acquisition of the image is much more important.

3.1.1 Leaf selection

Appropriate selection of leaf is very crucial. Our method strictly requires images of broad leaves. A broad-leaved tree is any tree which has wide leaves, rather than slim, needle-like leaves as found in conifers. Broadleaves are suitable for our system because most of the broadleaves are two dimensional in nature. Figure-3.2 shows a set of broadleaves.

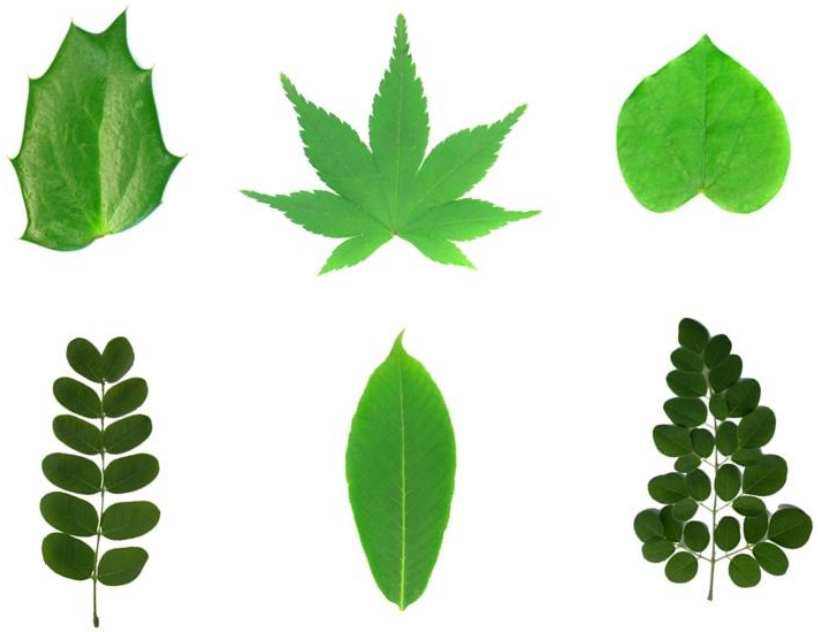


Figure 3.2 Broadleaf samples.

If a leaf is partially broken, deformed or damaged, it can still be used for image acquisition. Leaves which are too tender or too old should be avoided but there is no harm to try. The basic rule is to pick leaves, if available, which will represent the majority of the leaves of a certain plant.

3.1.2 Image acquisition

Proper image acquisition is very important. There are several things that should be kept in mind while taking photo of a leaf using digital cameras.

- The leaf should to be placed on a clean solid background and the color of the background must be different than the colors present in the leaf. For example, a piece of clean white paper can work as an excellent background.

- Use of flash should be avoided at all costs since the bright flash will most likely cause the image to have dark shadows underneath the leaf blade(s) and will certainly result in incorrect shape extraction in later phases of the method.
- The camera needs to be kept parallel to the leaf or the background. If the image is taken in a slanted manner the shape of the leaf will appear skewed in the captured image. This can significantly reduce the performance of our method.
- When taking photo of simple leaves the stem or leafstalk sticking out of the leaf blade should be removed. This will greatly improve the performance.
- Sometimes the leaf might not be flat enough and some parts of it can remain raised from the ground. In such cases using a scanner can be very helpful. The weight of the lid of the scanner will flatten the leaf and the shape of the leaf will be captured as it should be.
- Resolution of the image is not a big factor. However, if the resolution is too low, for example, less than 600 x 800 pixels with leaf occupying less than 50% of the pixels the chances are the method will not perform as expected. On the other hand, if the captured image is too big, the system will take much longer to process the image, but it will not reduce the chance of a correct recognition.

Figure-3.3 shows an example of image captured with a typical digital camera using a white paper background.

Figure 3.3 Example of an image acquired using digital camera.

3.2 Image Preprocessing

Image preprocessing is the second major step of our method. The term *preprocessing* refers to the task of manipulating and modifying the images using various image processing techniques in order to make the images suitable for next stage of processing which is *feature extraction*. In our method proper preprocessing is highly essential.

3.2.1 Overview of image and filter

A typical digital image is represented as a two dimensional matrix array where each value of the matrix is called a picture element or pixel. The value of these pixels can range between 0 to 255.

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0, \mathrm{N}-1) \\ f(1,0) & f(1,1) & \dots & f(1, \mathrm{N}-1) \\ \vdots & \vdots & & \vdots \\ f(\mathrm{M}-1,0) & f(\mathrm{M}-1,1) & \dots & f(\mathrm{M}-1, N-1) \end{bmatrix} \quad (3.1)\ [22]$$

The right side of Equation-3.1 is by definition an $M \times N$ digital image. For representing a colored image three such matrices are required, first one for red, second one for green and the third one for blue. These matrices forms a three dimensional matrix of size $M \times N \times 3$.

For our purpose colors are not important since we are interested only in the shape of the leaf. This is because the colors of the leaves of a particular plant can widely vary based on age of the leaves, environmental condition, season, sun, camera, time of day and so on. However, the images acquired by a digital camera or scanner are by default colored or *RGB* (Red Green Blue). Therefore, we need to convert the *RGB* image into a two dimensional image called *grayscale* image. There are many possible ways of calculating the value of each pixel in the grayscale image. For example, from the previous $M \times N \times 3$ color image f, we can calculate the $M \times N$ grayscale image g as following

$$g(x, y) = 0.2989 \times f(x, y, 0) + 0.5870 \times f(x, y, 1) + 0.1140 \times f(x, y, 2) \qquad (3.2)$$

Another possible way of converting an RGB image to a grayscale image is by simply taking the arithmetic mean of the R, G and B values, as is shown in Formula-3.3.

$$g(x, y) = \frac{1}{3} \times \sum_{z=0}^{2} f(\mathrm{x}, \mathrm{y}, \mathrm{z}) \qquad (3.3)$$

The image acquired by digital camera or scanners are likely to contain unwanted values called *noises*. Image noise is the random variation of brightness or color information in images produced by the sensor and circuitry of a scanner or digital camera. These *noises* need to be removed in order to enhance the image. We can apply different filters or masks to the grayscale image to get rid of the noises. A representation of a general 3×3 filter mask is shown in Figure-3.4.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figure 3.4 A general 3x3 filter. [22]

The response, *R*, of applying an $m \times n$ mask to a grayscale image at any point (x, y) is given by the following notation.

$$R = \sum_{i=1}^{mn} w_i z_i \qquad (3.3)\ [22]$$

Here the *w*'s are mask coefficients, *z*'s are the values of the image gray levels corresponding to those coefficients, and *mn* is the total number of coefficients in the mask.

3.2.2 Leaf extraction

The main task of preprocessing is to identify the leaf in an image and discarding all other information other than the leaf shape. This can be done with a little help from the user. The user can help identify the end of the leafstalk and some other points belonging to the leaf. Rest of the tasks should be carried out by the computer.

After the user selects the end point and some other parts of a leaf, the computer needs to identify the entire leaf or all the leaflets as accurately as possible. According to our method, this could be done by finding out the pixels that have similar value and connected to the reference pixels selected by the user. *Connectivity* defines which pixels are connected to other pixels. A set of pixels in a an image that form a connected group is called an object or a connected component. Determining which pixels create a connected component depends on how pixel

connectivity is defined. There are two types of connectivity in two dimensional images; 4-*connectivity* and 8-*connectivity*, show in Figure-3.5.

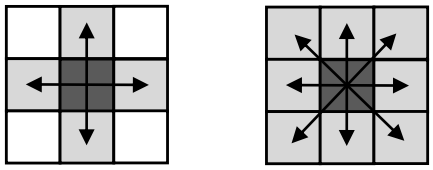


Figure 3.5 4 and 8-connectivity of pixels.

Pixels are 4-connected if their edges touch. This means that a pair of adjoining pixels is part of the same object only if they are both on and are connected along the horizontal or vertical direction. Pixels are 8-connected if their edges or corners touch. This means that if two adjoining pixels are on, they are part of the same object, regardless of whether they are connected along the horizontal, vertical, or diagonal direction. Figure-3.6 shows a 4-connected component drawn with white pixels and 8-connected component drawn with gray pixels.

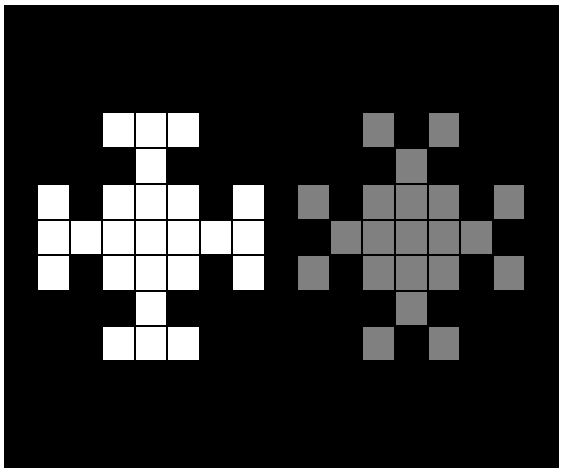
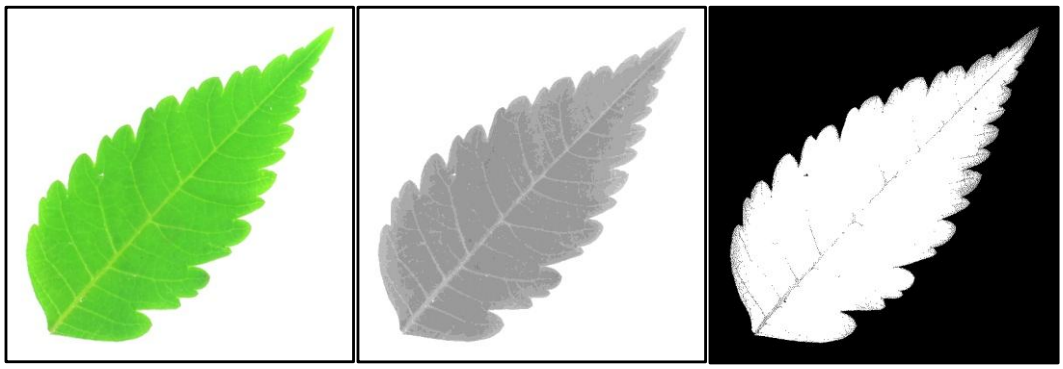


Figure 3.6 4-connected and 8-connected components.

In a properly taken leaf image the largest connected component is supposed to be the leaf. After detecting the largest connected component the grayscale image is converted into a *binary* image. A binary image is a digital image that has only two possible values for each pixel. Typically the two colors used for a binary image are black and white; 1 is used to represent white and 0 is used for black. The RGB, grayscale and binary version of an image can be seen in Figure-3.7.



(a) (b) (c)

Figure 3.7 a) RGB, b) Grayscale, c) Binary.

3.2.3 Aligning the leaf

Boundary tracing of an object, which is a connected component, in a binary image is done by crawling along the perimeter of the object towards a specific direction starting from a particular pixel. Tracing completes when the starting point is reached. Tracing boundary produces an array of pixels that forms the boundary. Using the endpoint of the leaf selected by the user it is possible to find out the tip of a leaf. Usually the tip of a simple leaf is located at the furthest point from the base. Therefore, by calculating the distances of each pixels of the boundary from the endpoint of the leaf the tip of leaf can be identified. Finding the tip is necessary because our method requires the simple leaves to be aligned horizontally having its endpoint on the left and

the tip on the right. One possible way of finding the tip of the leaf is to rotate the leaf by one degree at a time and the take horizontal length, then at some point the leaf would have the longest horizontal length and then the length will start decreasing. Finally rotating the original leaf by the angle at which the leaf has the longest horizontal length will most likely make the leaf horizontally aligned having the tip either on left or right. Then using the endpoint user selected the leaf base can be moved to right and the tip to left by flipping the image appropriately. A pictorial representation of this technique is shown in Figure-3.8. However, image rotation is a very expensive operation and should be avoided as long as there are other options available.

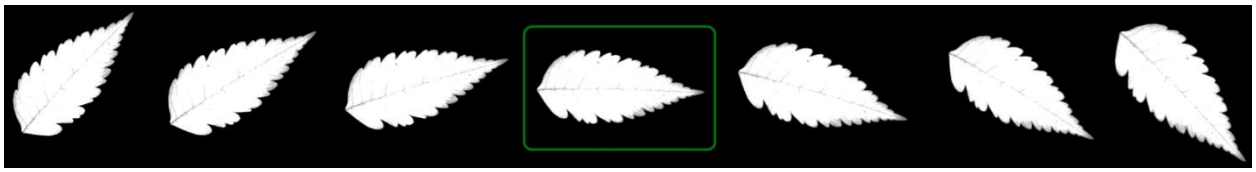


Figure 3.8 Aligning leaf using image rotation.

An efficient approach to align a leaf, provided the coordinate of the base-point of the leaf is known, would be to find out the angle (shown in Figure-3.9) between the horizontal axis and the leaf's major axis, which is the line connecting the base and tip of the leaf, and rotate the image by that angle. That is, to align the leaf horizontally with its base point on the left of the image, a couple of points are required. One of them is the base point which will be provided by the user and the other is the tip of the leaf. The tip of the leaf can found out by calculating the distance of all the points on the boundary of the leaf from the base point and then taking the point which is the furthest.

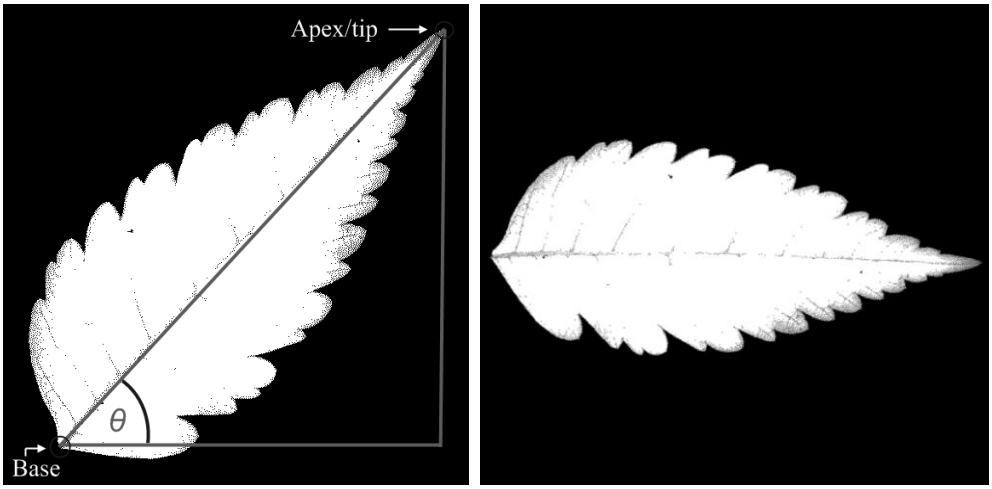


Figure 3.9 Angle between horizontal axis and major axis of a leaf (left), Aligned leaf (right).

The distance, D between two points, $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ is calculated using the simple formula,

$$D = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)} \tag{3.4}$$

After the tip of the leaf is located, the slope of the line that connects these two point is calculated using Formula-3.5 and the image is rotated by θ degrees to align the leaf.

$$\theta = \arctan((y_1 - y_2) / (x_1 - x_2)) \tag{3.5}$$

This will significantly reduce the amount of computation than the technique discussed earlier (Figure-3.8).The last step in preprocessing is to crop the aligned leaf. This can be done be finding out the leftmost, rightmost, topmost and bottommost white pixels of the image and then create a new image by taking pixels only inside the rectangle formed by those four points.

3.3 Feature Extraction

Feature extraction is another crucial part of our method. Since we are taking into account only the shape of the leaves, the goal of feature extraction would be to represent different leaf shapes with sets of values which can be used to distinguish one type of leaf from another. These sets of values extracted from the leaves are called features.

There are numerous ways of extracting shape of an object. In other words, the shape of an object can be represented in many different ways using many different techniques. However, which technique will be more effective depends on what sort of object is being dealt with. Extracting shape features of leaves can be very challenging. Unlike circles, rectangles, triangles, lines or other regular shapes of plant leaves are very complicated. Shapes of different leaves of the same plant can unbelievably different and shapes of the leaves of different plants can be surprisingly similar. This is why shape extraction of plant leaves are quite tricky. For our method we have taken in to account several features of the leaf shape. These features are listed below followed by explanations and diagrams.

- Area
- Convex area
- Filled area
- Solidity
- Perimeter
- Extent
- Equivalent diameter
- Discrete Fourier Transformation (DFT)
- Laminal Width Factor (LWF)

3.3.1 Area

Area is the actual number of pixels in the region. The area of leaf in a preprocessed image is the number of white or '1' pixels. For example, the area of the region in the image segment, shown in Figure-3.10, is 56 pixels because it contains 56 white pixels.

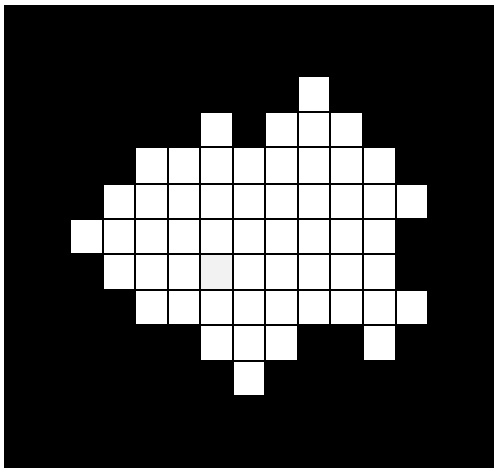


Figure 3.10 Area of the region is 56.

3.3.2 Convex area

Convex area specifies the number of white pixels in the 'Convex Image'. A convex image is a binary image that specifies the smallest convex polygon that can contain the region, with all pixels within the polygon filled in (i.e., set to 1). Figure-3.11 shows the convex image with the pixels filled in (shown in gray pixels).

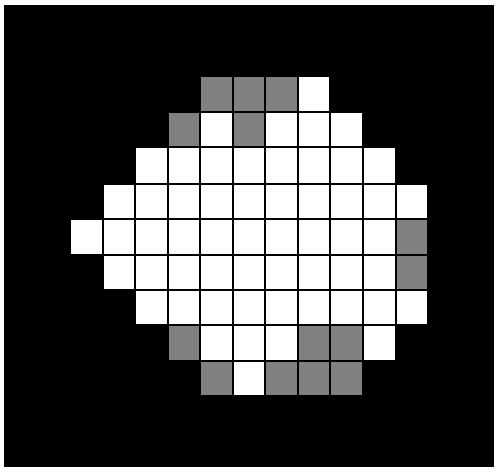


Figure 3.11 Convex area of the region is 70 pixels.

3.3.3 Filled area

Filled area specifies the area of the 'Filled Image'. A filled image is the binary image with all holes in the original region filled in (i.e., set to 1). On the left of Figure-3.12 shows a region with holes and on the right the filled image with the holes filled in (shown in gray pixels).

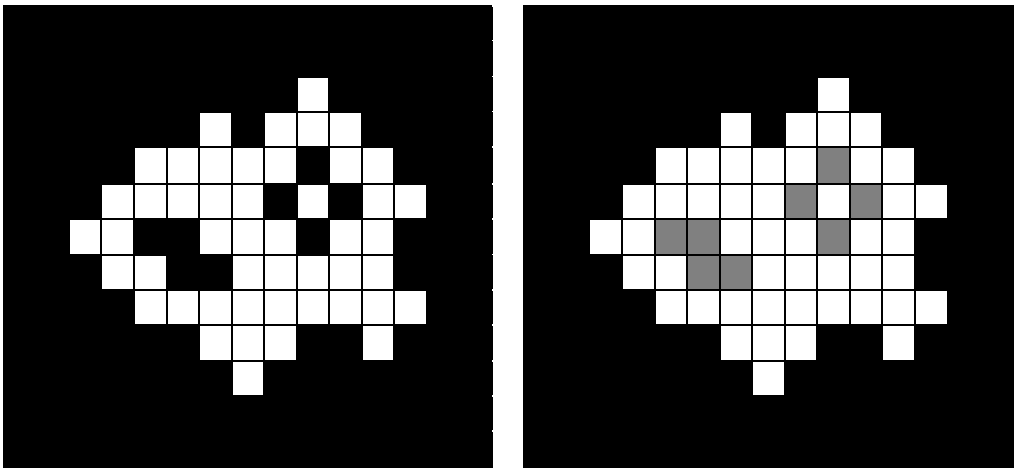


Figure 3.12 Region with holes (left) and the filled image (right).

3.3.4 Solidity

Solidity is the proportion of the area of the original region and convex area. The solidity of the region in Figure-3.13 is 56/70.

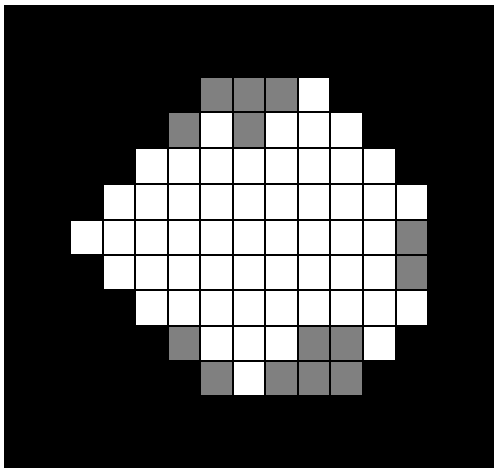


Figure 3.13 Solidity of a region is the ratio of its area and convex area.

3.3.5 Extent

Extent of a region specifies the ratio of pixels in the region to pixels in the smallest rectangle containing the region. In Figure-3.14, the area of the smallest rectangle (shown using gray pixels) containing the region is 99 and the area of the region is 56. Thus, solidity of the region is 56/99.

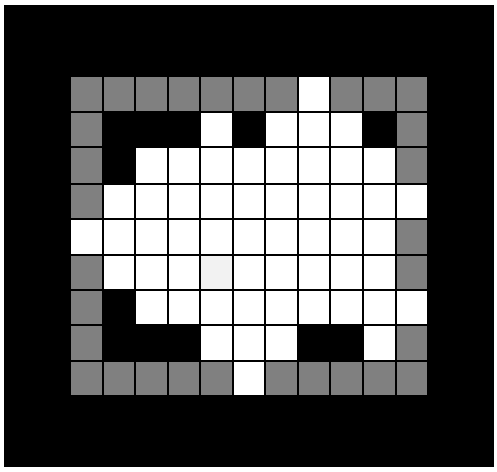


Figure 3.14 Smallest rectangle containing the region.

3.3.6 Perimeter

Perimeter of a region is the summation of the distances between each adjoining pair of pixels around the border of the region. The perimeter of a region is shown in gray pixels in Figure-3.15.

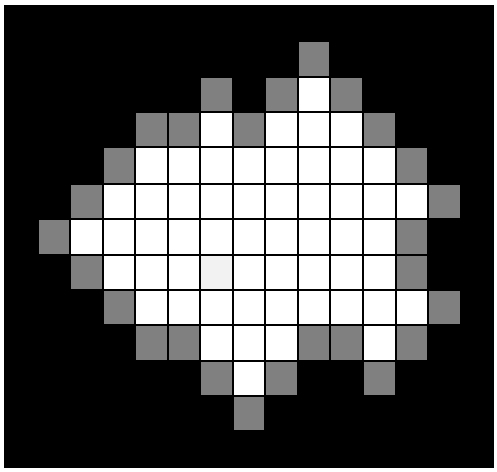


Figure 3.15 Perimeter of a region.

3.3.7 Equivalent diameter

Equivalent diameter specifies the diameter of a circle with the same area as the region. A region's equivalent diameter, D_E can be calculated using the formula,

$$D_E = \sqrt{(4 \times \text{Area} / \pi)} \quad (3.6)$$

3.3.8 Laminal Width Factor (LWF)

The line connecting the furthest two points of the preprocessed leaf is the laminal major axis. And the maximum width, which is perpendicular to the major axis, is considered the minor axis. These two axes of a leaf are shown in Figure-3.16.

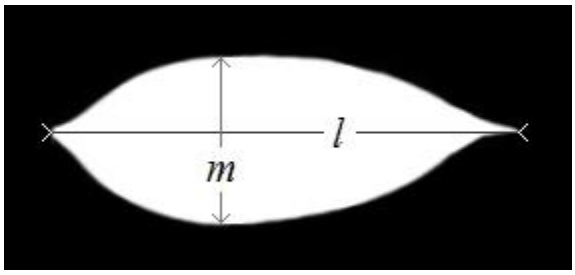


Figure 3.16 Laminal major axis *l* and laminal minor axis *m.*

Having major and minor axis of a leaf determined, the Laminal width factor (LWF) of the leaf in hand is measured by slicing across the major axis and parallel to the minor axis (Figure-3.17). Then the feature pointes are normalized by taking the ratio of the slice lengths and leaf lengths (major axis). The leaf is sliced, perpendicular to the major axis, into a number of vertical strips. Then for each strip, the ratio of length of strip and the length of the entire leaf is calculated. The ratio *R*, at column *c* can be computed by the following formula,

$$R_c = \frac{W_c}{l} \quad (3.7)$$

Here W_c is the width of the leaf at column c and l is the length of the entire leaf.

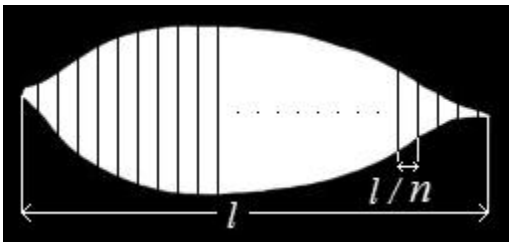


Figure 3.17 Laminal width factor extraction.

3.3.9 Discrete Fourier Transformation (DFT)

In mathematics, the Fourier transform (often abbreviated FT) is an operation that transforms one complex-valued function of a real variable into another. It decomposes a function into a set of oscillatory functions. A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. A DFT decomposes a sequence of discrete values into components of different frequencies. Let x_0, ..., x_{N-1} be complex numbers. The DFT is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \qquad ;k = 0,1,\ldots, N-1. \tag{3.8}$$

Evaluating this definition directly requires $O(N^2)$ operations: there are N outputs X_k, and each output requires a sum of N terms. An FFT is any method to compute the same results in O(Nlog$N$) operations. More precisely, all known FFT algorithms require Θ(Nlog$N$) operations, although there is no proof that better complexity is impossible.

FFT can be applied on the base-point contour distance (BCD) curve of a leaf. The base-point contour distance curve can be obtained by computing the distances of all the boundary pixels of a leaf from its base-point selected by the user.

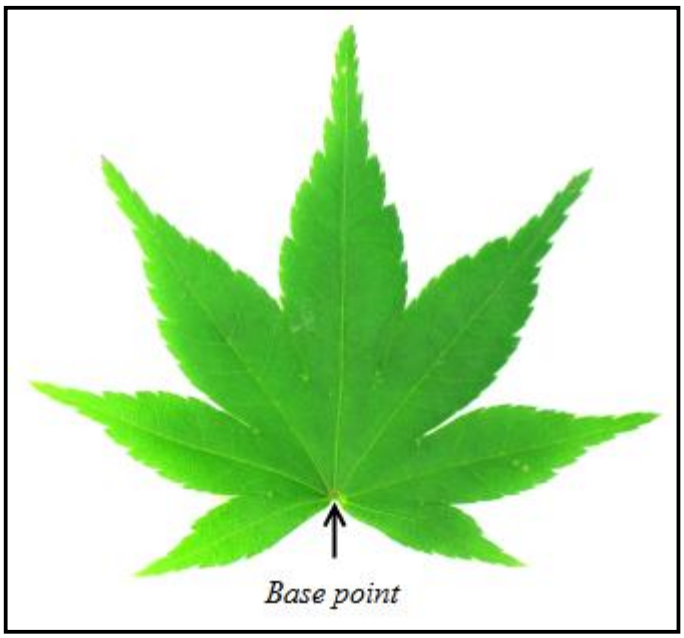


Figure 3.18 Base-point of a Japanese maple leaf.

Figure-3.18 shows the base-point of a Japanese maple leaf and Figure-3.19 shows its BCD curve. Figure-3.22-23 shows different BCD curves of different types of leaves. These curves contain many vital information about the shapes of the leaves. A BCD can be used to find out the following facts about a leaf.

1. Type of leaf: Simple, compound
2. Leaf margin: lobed, plain, toothed, smooth
3. Lobe size
4. Frequency of tooth occurrence
5. Tooth size

However, the number of points in the curve can widely vary depending on the size of the leaf image. This makes the comparison between two BCD curves a bit tricky. This is where the FFT comes in. Applying FFT on the BCD curve of a leaf produces a sequence of complex numbers, each of which represents the magnitude and phase of a particular sinusoidal wave. By picking a threshold and keeping all complex number having higher magnitude than the threshold value and setting 0 to rest of the complex number produces a filtered sequence of complex number. Now, applying inverse FFT on this filtered sequence we can get the BCD curve back but not the same one. Depending on the threshold value that had been picked the new BCD curve will contain lesser details than it previously had. Figure-3.19 shows the original BCD curve, Figure-3.20 show the magnitudes of the sinusoidal waves which were produced by applying FFT on the BCD curve and Figure-3.21 shows the reconstructed smooth BCD curve after FFT filtering.

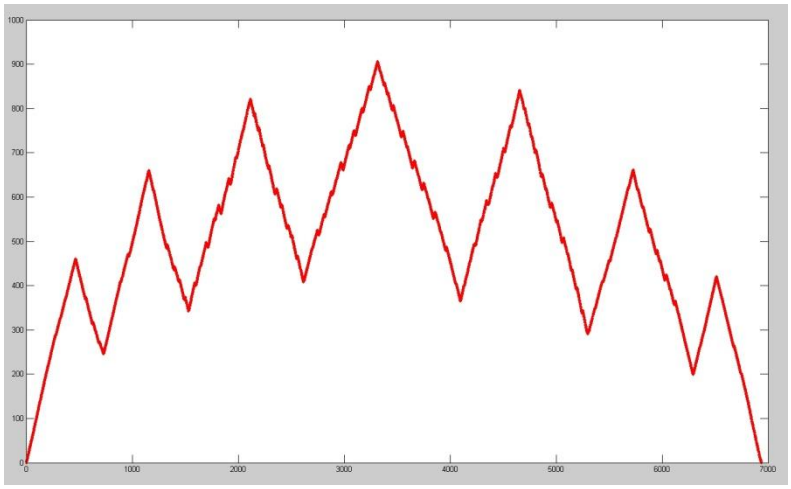


Figure 3.19 BCD curve of the a Japanese maple leaf.

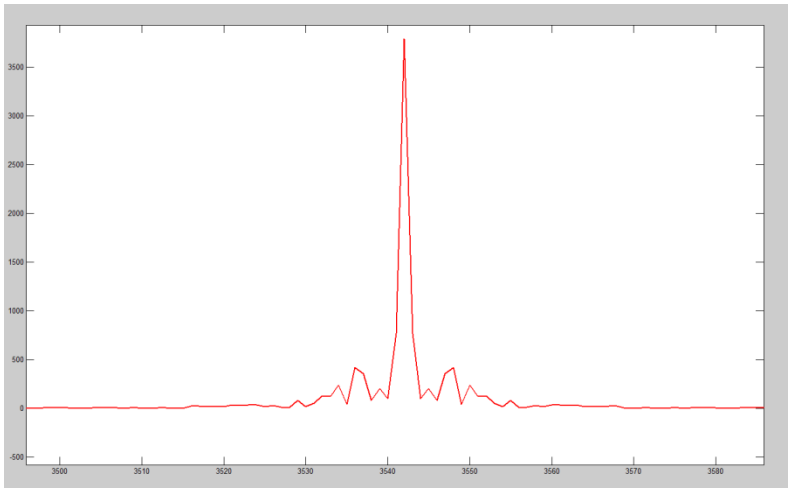


Figure 3.20 The magnitudes of sinusoidal waves produced by FFT on the BCD curve.

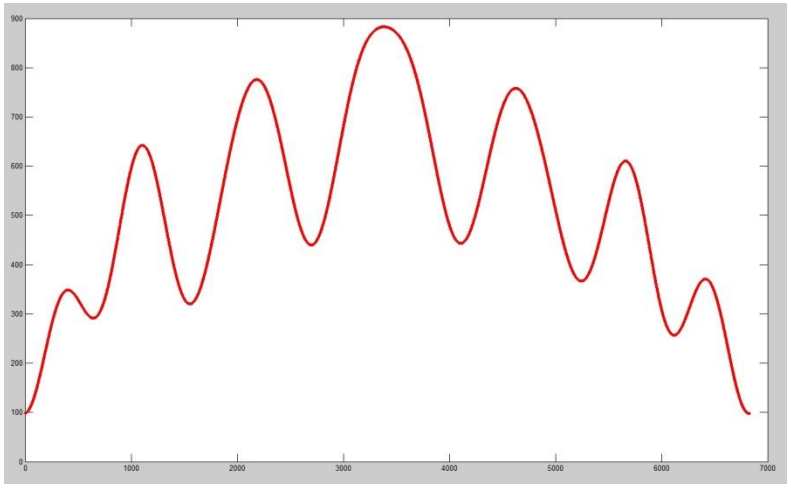


Figure 3.21 Reconstructed BCD curve after FFT filtering.

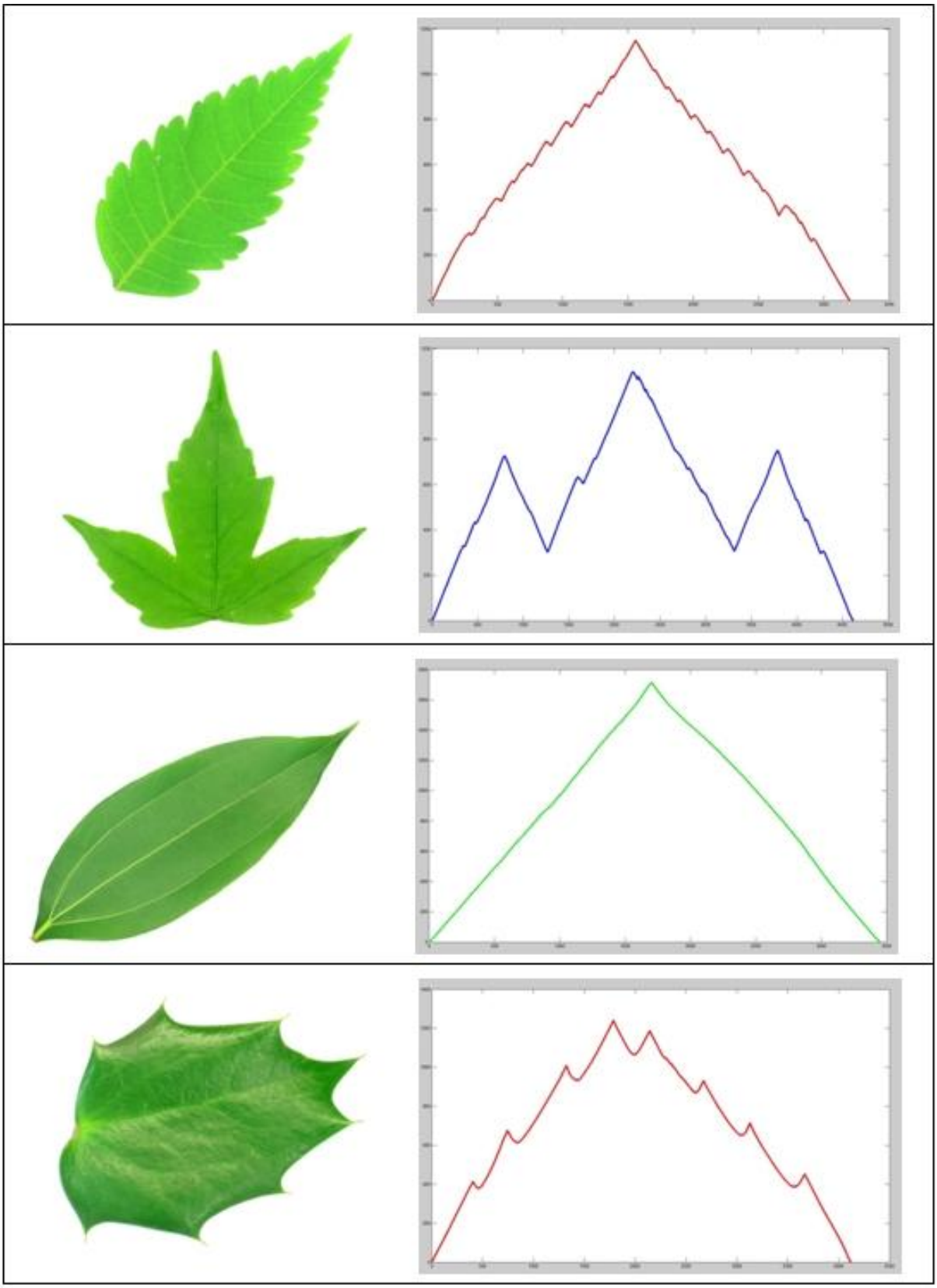


Figure 3.22 Different types of leaves and their BCD curves.

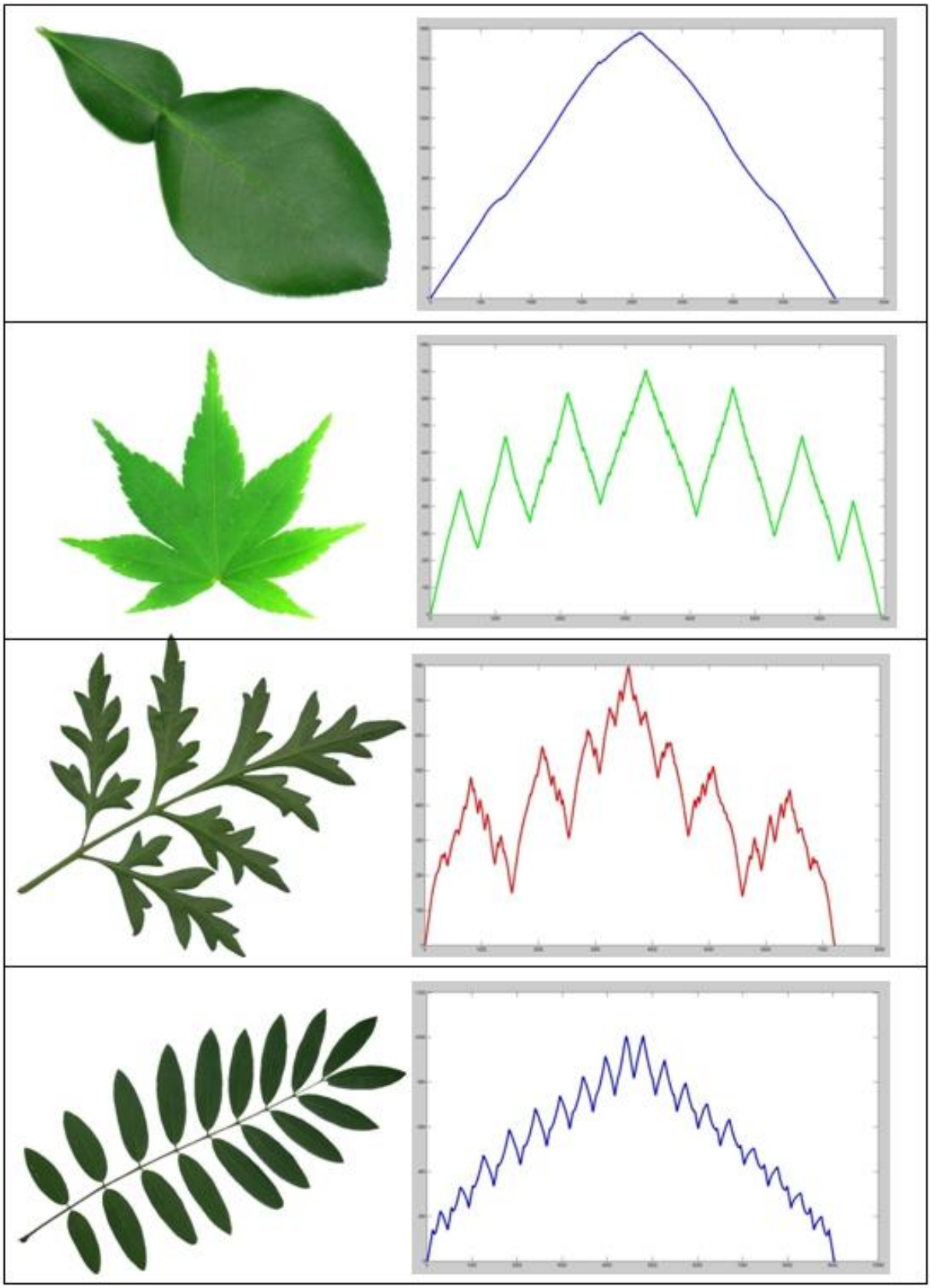


Figure 3.23 Different types of leaves and their BCD curves.

3.4 System Training

This step involves teaching the system which leaf belongs to which plant species. This phase is very important because the system will not perform as expected unless it is trained properly. No matter how appropriately the previous steps have been done, the system might still malfunction due to improper training. Two techniques have been applied to train the system. Both of them are discussed below.

3.4.1 Sum of Squared Distances (SSD)

The first technique requires storing the features of all the leaves in a two dimensional matrix, where a row of the matrix contains features of a single leaf and a particular column contains values of a particular feature of all the leaves. For a training dataset *D* containing of *n* leaves and *m* features the matrix structure is shown below.

$$D(x, y) = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1m} \\ f_{21} & f_{22} & \cdots & f_{2m} \\ \vdots & \vdots & & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nm} \end{bmatrix} \tag{3.9}$$

After the matrix is constructed the system is ready to be tested. In order to use this matrix for testing the same set of features needs to be extracted from the test leaf. Then the sum of squared distances (SSDs) of the feature set of the test leaf and every row or leaf from the matrix are calculated. The leaf or row of the matrix that produces the minimum distance is the best match. If the feature set of the test leaf is denoted by $T = \{ f_{T1}, f_{T2}, \ldots , f_{Tm} \}$ then the SSD between the test leaf and the i^{th} leaf of the matrix is given by

$$SSD_i = \sum_{k=1}^{m} (f_{Tk} - f_{ik})^2 \quad (3.10)$$

Although this technique is widely used in measuring differences between two points in a *m* dimensional space, but for a dataset matrix containing hundreds and thousands of leaves this technique would require a lot of calculations to be performed. Our method needs a faster way of finding the right match.

3.4.1 Probabilistic neural network (PNN)

Another potential option for training the system is to use artificial neural networks. Neural networks are excellent in greater generalization. The PNN, learns rapidly compared to the traditional back-propagation, and guarantees to converge to a Bayes classifier if enough training examples are provided, it also enables faster incremental training, and robust to noisy examples [12], [13], [14].

3.5 Testing

We have considered a couple of methods for testing our system. One of them is Repeated random sub-sampling validation and the other one is k-Fold cross validation. These two methods are discussed in detail in chapter 4.6.

Chapter 4

Implementation

Implementation of the proposed method has been a challenging task. This chapter describes the techniques that have been used in the implementation of our method discussed in the previous chapter.

4.1 Tools and System Specifications

The entire implementation was done and tested using 64-bit MATLAB 7.8.0 on 64-bit Windows Vista operating system with Intel Core i7 microprocessor and 6 GB DDR3 RAM. Although it has been developed using a powerful machine, it does not require such power to run. It has also been tested on low end machine having 32-bit Windows Vista environment with Intel Celeron processor and the program performs great.

4.2 Dataset

We collected a large number of leaf images to for our initial dataset. This dataset can be partitioned into two groups. First one includes 1907 leaf images of 32 plant species. All the leaves in this partition are simple leaves, acquired using digital camera by *Wu et al.* [1] and have 1600 x 1200 pixels resolution. The second part of the dataset contains 200 compound leaf images acquired by myself. These 200 compound leaves belong to 5 plant species. Images in this

partition were also captured using a digital camera and the resolution is 1024 x 768 pixels. All images are truecolor RGB images.

We reorganized our dataset by selecting taking 30 plant species from the first partition, each species having 40 specimens, making 1200 simple leaf images and all 5 plant species from the second partition, each species having 40 samples, making 200 compound leaf images. In total there are 1400 leaf images of 35 plant species, where 1200 are simple leaves and 200 are compound leaves. Figure-4.1 shows a glimpse of our dataset.

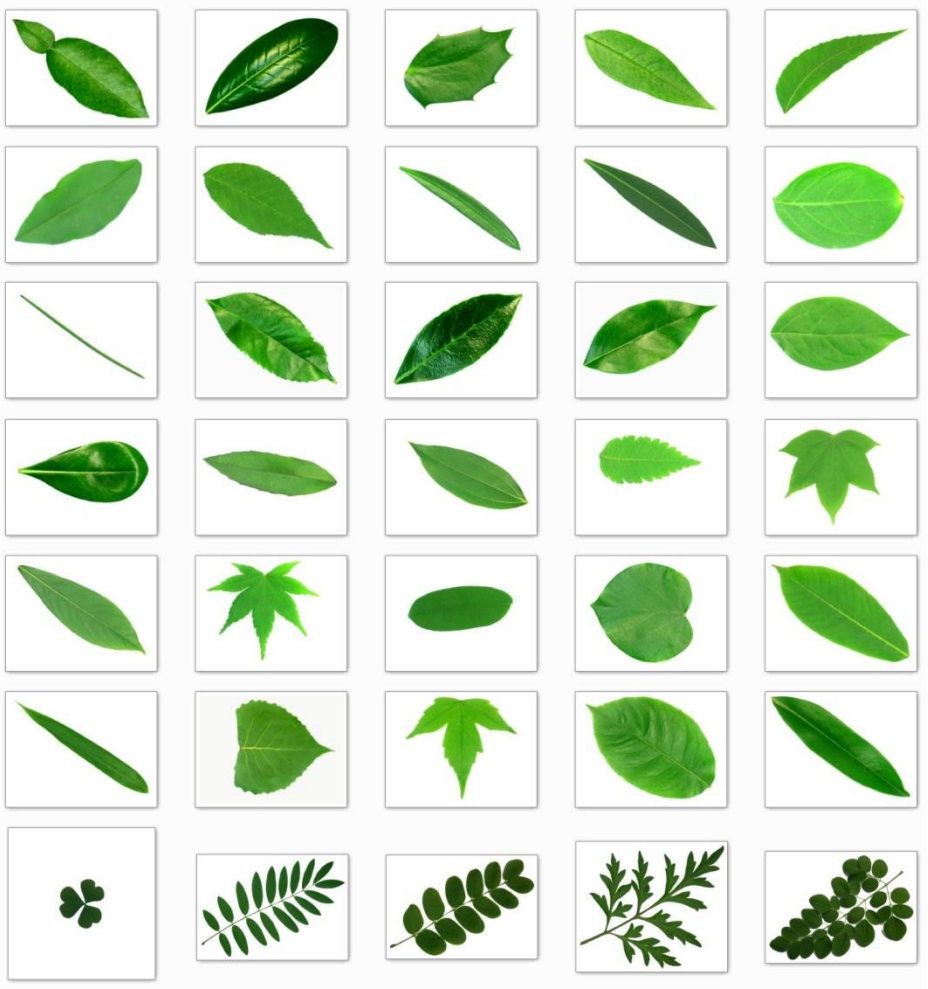


Figure 4.1 A glimpse of our dataset.

4.3 Preprocessing

The implementation of image preprocessing has been done in several steps. These steps are discussed in the following subsections.

4.3.1 Base point and reference point selection by the user

The user needs to selects the base point of the leaf from an RGB image, then few more points are required to be selected by the user. These points are called the reference points, which must be located within the blade/lamina of the leaf. Figure-4.2 shows the base point of a leaf and a few intuitively selected reference points that are located within the leaf area and have different intensities of green.

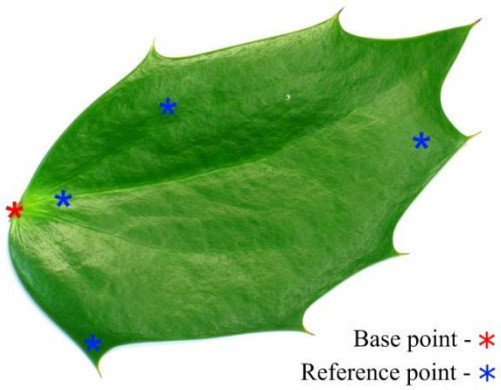


Figure 4.2 Base point and reference points of a leaf.

4.3.2 Conversion from RGB to grayscale

After the user selects the reference points the RGB image is converted to grayscale. Figure-4.3 shows the conversion of an RGB image to grayscale image.

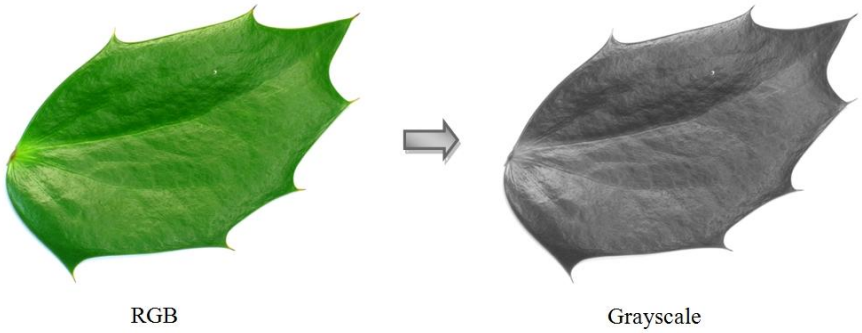


Figure 4.3 RGB to grayscale conversion.

4.3.3 Grayscale to binary conversion

Based on the reference points selected by the user, the surrounding pixels which have similar intensities as the reference points as well as 8-connected are set to 1 and the rest of the pixels are assigned 0. A custom function has been used for this purpose. This function takes the reference points and the RGB image as inputs, uses 8-connectivity to extract the leaf and returns a binary image where the pixels on the leaf are white and the rest of pixels are black. An example of the output of this step can be seen Figure-4.4.

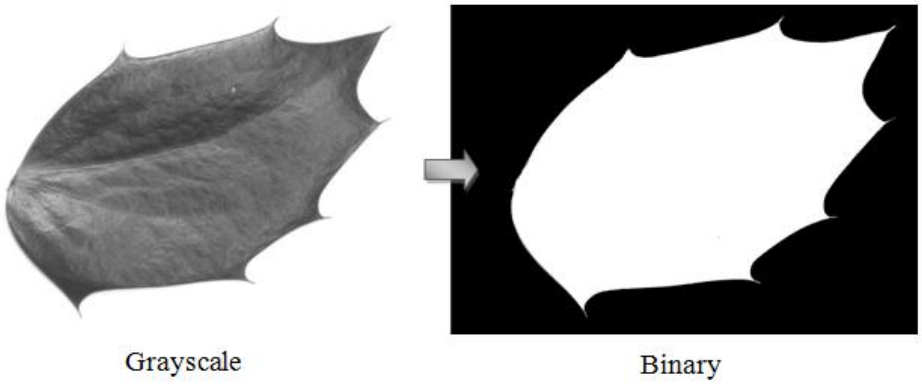


Figure 4.4 Grayscale to binary conversion.

4.3.4 Aligning the binary leaf

The orientation of the binary leaf is identified and the inclination of the major axis is calculated. Then finally the leaf is rotated and aligned horizontally with its base point on the left of the image and the tip on the right. Figure-4.5 shows the unaligned leaf, the inclination angle and the aligned leaf. The detailed method of finding the tip and aligning the leaf can be found in the preceding chapter, section 3.2.3.

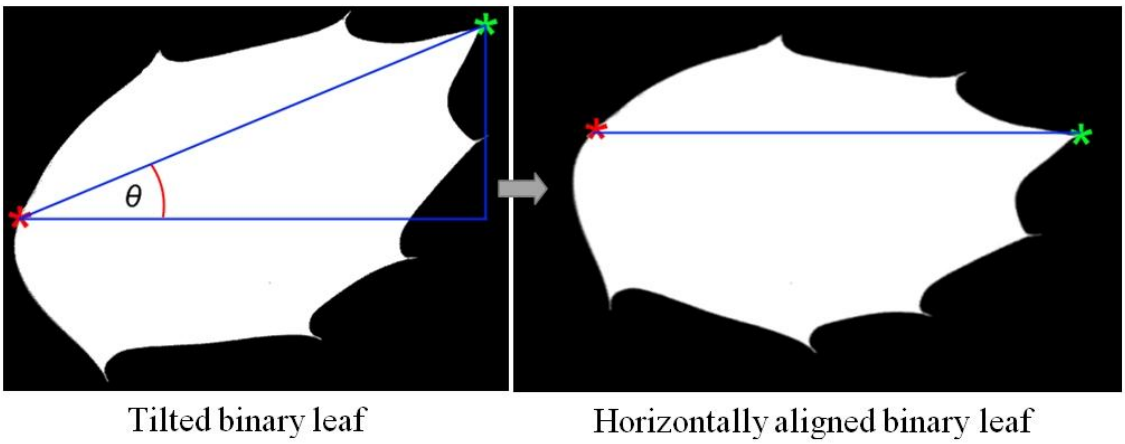


Figure 4.5 Aligning a binary leaf.

4.4 Feature Extraction

Feature extraction has been done differently for simple leaves and compound leaves. Few features were extracted for the purpose of differentiating simple leaves from compound leaves. Some features have been extracted from both simple leaves and compound leaves. A special set of features has been extracted from simple leaves.

4.4.1 Distinguishing simple and compound leaves

FFT filtering was applied on the BCD curve and the ratio of Filled area and normal area were calculated for differentiating simple leaves from the compound ones. The BCD curve of a leaf is first normalized by dividing all the distances by the maximum distance of curve. Then it is smoothed by applying FFT and using a threshold to reconstruct the BCD. FFT is applied 30 times on a BCD curve and the gradual smoothing is watched. Each time the number of peaks in the smoothed BCD is stored in an array. After completion of 30 iterations the number of peak that occurred maximum times is taken. If it is 1, then it is most certainly a simple leaf. If it is more than 1, then it would be either simple or compound. To determine if the leaf is simple or compound, the number of valleys in the BCD curve (smoothed with threshold 30) is counted. The number of valleys can be counted using an easy method. If the distance is gradually decreasing and at some point it starts rising, then that point is a valley in the BCD curve. Since it has been smoothed by factor 30, all the tooth become smooth and only the lobes/leaflets remain. Figure-4.6 shows the original BCD curve (blue) of a compound leaf and the smoothed BCD (red) having the valleys marked with *'s.

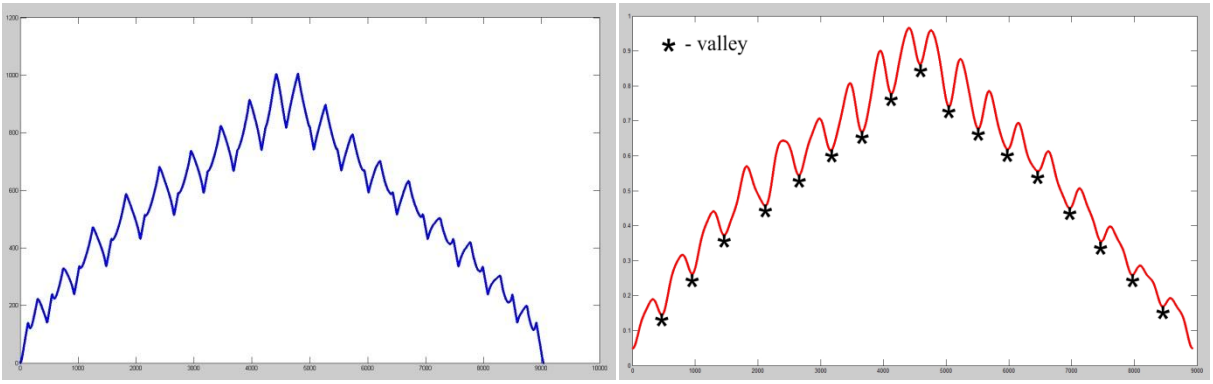


Figure 4.6 The original BCD curve (blue), the smoothed BCD and valleys identified.

Now that the valleys in the BCD curve are identified, the corresponding coordinates of the pixels from the binary image are extracted. Figure-4.7 shows the corresponding pixels of the leaf which formed valleys in the BCD curve shown in Figure-4.6.

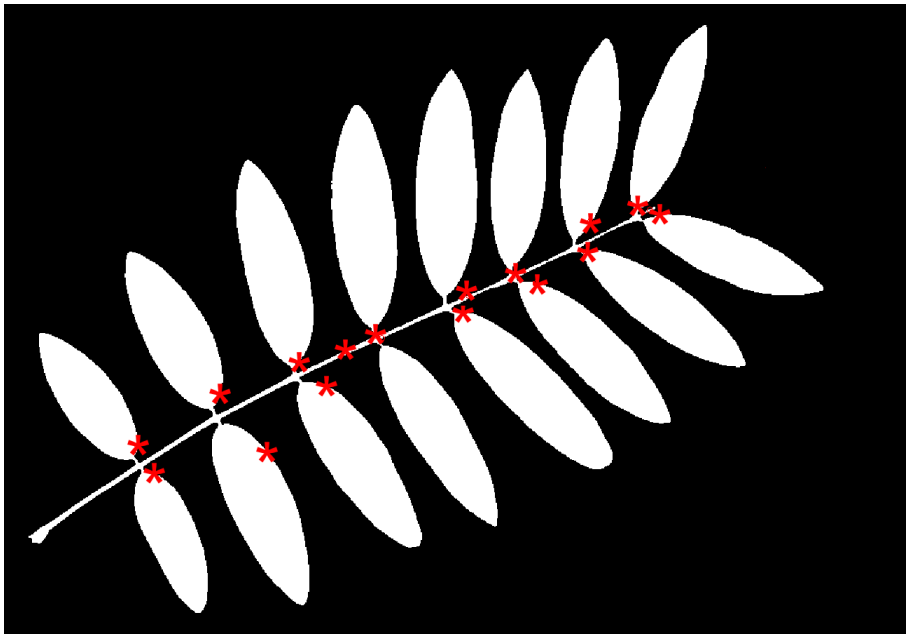


Figure 4.7 Corresponding pixels of the leaf to the valleys of BCD curve.

The distances between these points are calculated. If the distance between one or more pair of points is less than 15% of the length of the laminal major axis, then it is most likely a compound leaf. But if all the distances are greater than that, then the ratio of the Filled area and normal area of the leaf is calculated. This is done because some compound leaves do not have space among the leaflets and closed gaps/holes form due to touching leaflet and the rachis. The regions colored with red and blue in Figure-4.8 are the holes of the compound leaf. If a leaf has holes which occupies more than 10% of the leaf area than the leaf is identified as a compound leaf.

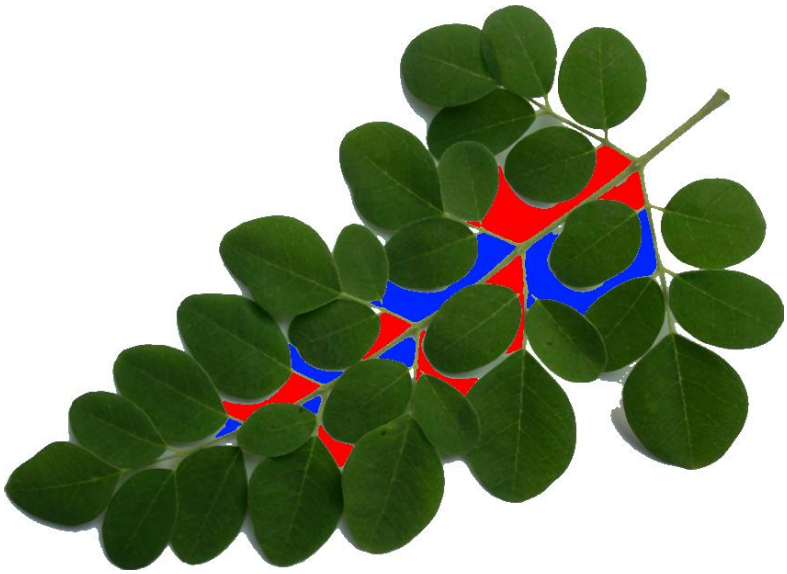


Figure 4.8 Holes/closed gaps between leaflets and the rachis of a compound leaf.

4.4.1 Simple leaf features

The following features have been extracted from the simple leaves.

1. Solidity
2. Area / Length of major axis
3. Perimeter / Length of major axis
4. Length of minor axis / Length of major axis
5. Equivalent diameter / Length of major axis
6. Extent
7. (Convex area - Area) / Area
8. Laminal width factor (LWF) of size 11

We tested the accuracy of the system with different sizes of LWF, starting from 1 up to 20. The highest accuracy occurs at the size of 11. Therefore we decided to set the size of the LWF to 11.

4.4.1 Compound leaf features

Features extracted from the compound leaves are same as the simple leaves except the LWF is not extracted from the compound leaves. Instead a different feature is extracted. The features extracted from the compound leaves are given in the following list.

1. Solidity
2. Area / Length of major axis
3. Perimeter / Length of major axis
4. Length of minor axis / Length of major axis
5. Equivalent diameter / Length of major axis
6. Extent
7. (Convex area - Area) / Area
8. (Filled area - Area) / Area

4.5 System Training

We decided to use Probabilistic Neural Networks for training our system. PNNs perform very fast both in training and testing. In Figure-4.9, R is the number of elements (features) in the input vector. Q is the total number of training (inputs, output) pairs, this is also the number of neurons in layer 1 (radial basis layer), and, IW and b_1 are the weight matrix and bias vector of the redial basis layer. K is the number of classes of input data, and the number of neurons in layer 2 (competitive layer) and LW is the weight matrix of the competitive layer.

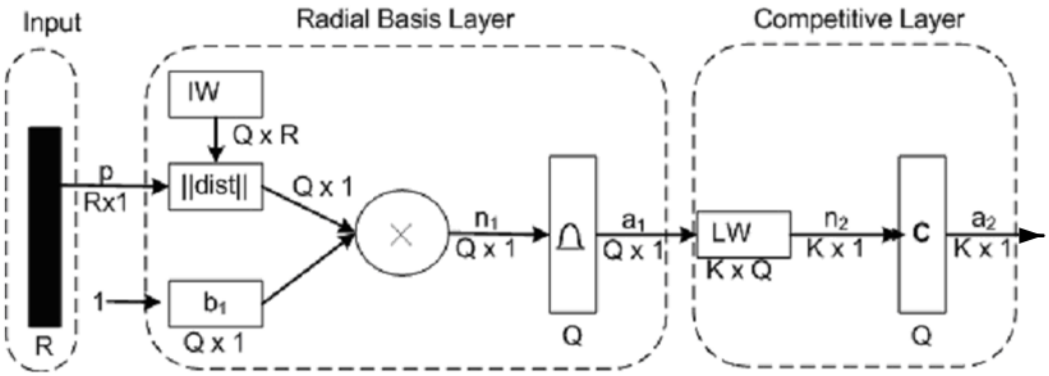


Figure 4.9 A typical Probabilistic Neural Network.

In the training phase, $IW = X^T$ where X is the training pattern matrix of size $R \times Q$ containing Q training patterns as column vectors. $LW = T$ where T is the training class matrix of size $K \times Q$ containing Q training class information as column vectors, here note that the entries to the matrix T are Boolean and if the i^{th} pattern (column) in X belongs to the class k then the k^{th} entry of the i^{th} column in matrix T is set to 1 and rest of the entries of that column is set to 0 to train the PNN and rest of the feature vectors are kept apart for testing.

Two neural networks have been implemented in our system. The first PNN was used for identifying simple leaves. This PNN has R={8,9,…,27}, Q = 30 × 36 and K = 30. This means varying number of features (8 to 27) were extracted from each leaf, 30 types of simple leaves, each type having 36 samples, were used to train the PNN. The second PNN was used for identifying compound leaves. This PNN has R = 8, Q = 5 × 36 and K = 5, that is, 8 features were extracted from each leaf, 5 types of compound leaves, each type having 40 samples, were used to train the network.

In the testing phase, when a test input vector is presented the || *dist* || box produces a vector whose elements indicate how close the input is to the vectors of the training set. These elements are multiplied element by element by the bias, and passed as the argument *n* to the radial basis transfer function ($a_1(i) = \exp(-n_1(i)^2)$). An input vector close to a training vector is represented by a number close to 1 in the output vector a_1. In the second layer the multiplication $T * a_1$ sums the elements of a_1 due to each of the *K* input classes. Finally, the second-layer uses a comparative transfer function and produces a 1 corresponding to the largest element of n_2, and 0 elsewhere. It illustrates that, the network has classified the test input vector into a specific one of the *K* classes because that class had the maximum probability of being correct as it is most similar with an example of that class.

4.6 Validation

Repeated random sub-sampling validation technique randomly splits the dataset into training and validation data. For each such split, the model is fit to the training data, and predictive accuracy is assessed using the validation data. The results are then averaged over the splits. The advantage of this method (over *k*-fold cross validation, which is discussed in the following paragraph) is that the proportion of the training/validation split is not dependent on the number of iterations (folds). The disadvantage of this method is that some observations may never be selected in the validation subsample, whereas others may be selected more than once. In other words, validation subsets may overlap.

In k-Fold cross validation the dataset needs to be partitioned into k subsamples. Of the *K* subsamples, a single subsample is retained as the validation data for testing our method, and the remaining ($K - 1$) subsamples were used as training data. The cross-validation process is then

repeated *K* times (the *folds*), with each of the *K* subsamples used exactly once as the validation data. The *K* results from the folds were then averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once.

In our case, we decided to use 10-fold cross validation considering its advantages over repeated random sub-sampling.

Chapter 5

Testing and Evaluation

Our system has been rigorously tested throughout different stages of development. In fact, the major decisions regarding preprocessing techniques, feature extraction and system training has been taken based on respective test results.

5.1 Performance of Simple Leaf Identifier

This section analyzes the performance of the first PNN, which was trained with and used for identifying simple leave, based on different features that have been extracted from the simple leaves. Table-5.1 shows accuracies of different features extracted from simple leaves.

Feature	Recognition Accuracy (%)
Solidity	28.08
Area / Length of major axis	28.0
Perimeter / Length of major axis	33.08
Length of minor axis / Length of major axis	33.16
Equivalent diameter / Length of major axis	31.16
Extent	20.58
(Convex area - Area) / Area	27.91
Combined	82.41

Table 5.1 Separate and combined accuracies of different features of simple leaves.

We recorded the accuracies of the system, excluding the features in Table-5.1, with different sizes of LWF, starting from 1 up to 20. The highest accuracy, 84.50%, occurs at the size 11. Therefore we decided to set the size of the LWF to 11. Figure-5.1 shows the size of LWF versus the recognition accuracy curve.

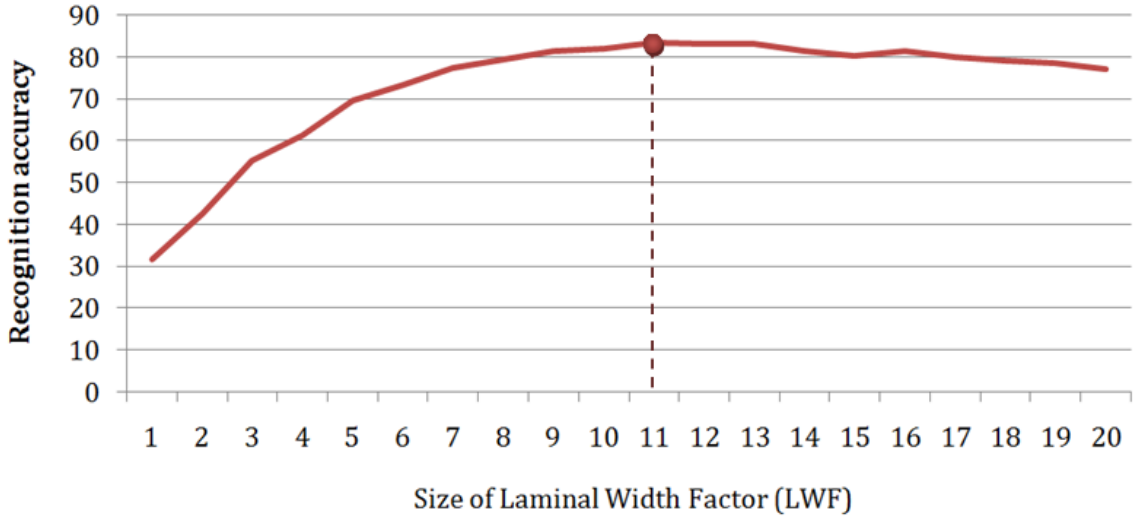


Figure 5.1 Size of LWF vs. recognition accuracy curve.

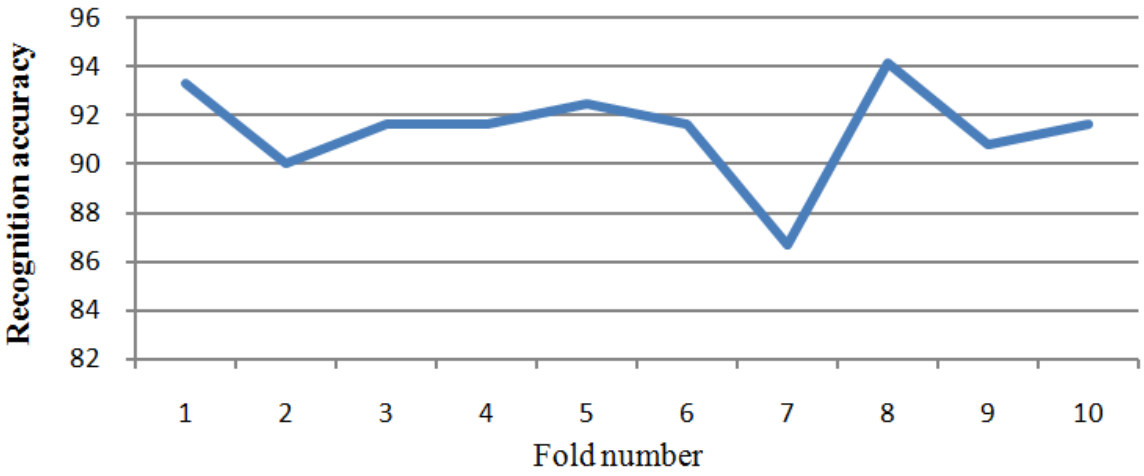


Figure 5.2 Fold number vs. recognition accuracy of the simple leaf unit.

Figure-5.2 shows the recognition accuracies of this unit at different fold numbers, when all the features are employed, that is, all features from Table-5.1 and the LWF of size 11 combined. The average accuracy is 91.41%.

5.2 Performance of Compound Leaf Identifier

This section analyzes the performance of the second PNN, which was trained with and used for identifying compound leave, based on different features that have been extracted from the compound leaves. Table-5.1 shows accuracies of different features extracted from simple leaves.

Feature	Recognition Accuracy (%)
Solidity	70.0
Area / Length of major axis	48.5
Perimeter / Length of major axis	74.0
Length of minor axis / Length of major axis	74.0
Equivalent diameter / Length of major axis	67.0
Extent	85.0
(Convex area - Area) / Area	75.5
(Filled area - Area) / Area	66.5
Combined	98.5

Table 5.2 Separate and combined accuracies of different features of compound leaves.

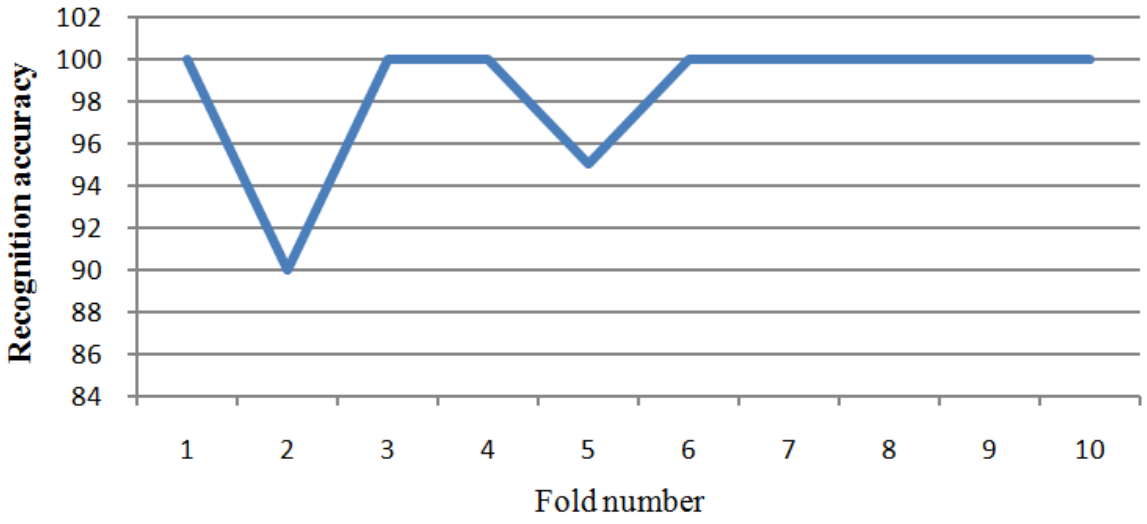


Figure 5.3 Fold number vs. recognition accuracy of the compound leaf unit.

Figure-5.3 shows the recognition accuracies of the compound leaf unit at different fold numbers, when all the features from Table-5.2 are employed. The average accuracy is 98.5%. The reason behind such high recognition accuracy is the fact that only 5 types of compound leaves have been used to train the network, on the contrary the simple leaf unit was trained with 30 types of leaves and has relatively lower recognition accuracy.

5.3 Species Based Performance

Our system has been trained with 35 plant species. Some species got as high as 100% accuracy and the lowest accuracy is 70%. These variations occur due to similarities of leaf shape among different species. Some plants can be recognized with 100% accuracy. This means leaves of those plants have unique leaf shape. These accuracies are given in the following table. The average species based recognition accuracy is 91.71%. All the species based accuracies are given in Table-5.3.

Table 5.3 Species based recognition accuracy.

#	Plant Species	Recognition Accuracy (%)
1	Pubescent Bamboo	100
2	Chinese Horse Chestnut	80
3	Chinese Redbud	90
4	True Indigo	100
5	Japanese Maple	100
6	Nanmu	90
7	Castor Aralia	100
8	Goldenrain Tree	100
9	Chinese Cinnamon	80
10	Anhui Barberry	70
11	Big-Fruited Holly	90
12	Japanese Cheesewood	100
13	Wintersweet	100
14	Camphortree	100
15	Japan Arrowwood	70
16	Sweet Osmanthus	70
17	Deodar	100
18	Crape Myrtle	90
19	Oleander	100
20	Yew Plum Pine	100
21	Japanese Flowering Cherry	80
22	Glossy Privet	80
23	Chinese Toon	90
24	Peach	70
25	Ford Woodlotus	100
26	Trident Maple	100
27	Beale's Barberry	90
28	Southern Magnolia	90
29	Canadian Poplar	80
30	Tangerine	100
31	Wood Sorrel	100
32	Rattlebush	100
33	Unknown Plant 3	100
34	Orange Cosmos	100
35	Moringa	100
	Average accuracy	**91.71%**

5.4 Performance with Partial Leaves

Our system has been able to successfully identify plant when tested with partial leaves.

Figure 5.4 Recognition with partial compound leaves.

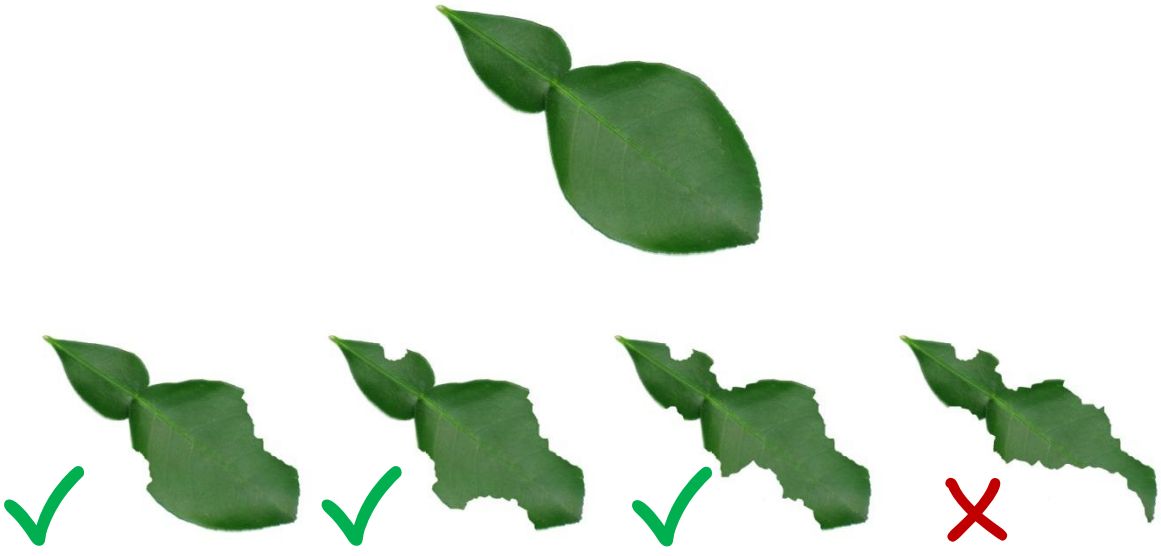


Figure 5.5 Recognition with partial simple leaves.

Figure 5.4 and 5.5 shows the partial leaves that out system is able to recognize correctly. The partial leaves with green tick marks was correctly recognized and those with a red crosses could not be recognized. The images on the top, in both figures, show the original leaves.

5.5 Evaluation

The overall recognition accuracy of our system is 92.42%. This overall accuracy *A* was calculated by

$$A = \frac{(A_s \times D_s) + (A_c \times D_c)}{D_s + D_c} \quad (5.1)$$

Here,

A_s = Accuracy of the simple leaf unit

= 91.41%

D_s = Dataset size of simple leaf unit

= 1200

A_c = Accuracy of the compound leaf unit

= 98.5%

D_c = Dataset size of compound leaf unit

= 200

Our system has the capability to differentiate simple leaves from compound leaves as well as identify species of the compound leaves. None of the previous works, that we have come across, has this unique ability. In fact, all these previous works dealt with simple leaves only. This is the biggest achievement of our work.

Performance analysis with partial leaves shows that our system can identify the plant from partial leaves up to a certain extent. If the shape of a leaf is not badly destroyed or distorted, our system can identify it. However, there is no concrete measurement of this capability. But it is enough to claim that our system is intelligent enough to identify the species of partially damaged leaves.

Chapter 6

Conclusions

Our system is no match to the human ability of plant identification. Human brain is far too powerful to be compared with our system. But when it comes to identifying a plant from hundreds and thousands leaf samples, a system like ours can definitely be of use.

In this dissertation we presented a robust and computationally efficient method for plant species recognition from leaf image. Our system has three major functional units. The task of the first unit is to identify whether the input leaf is simple or compound. The second unit deals with the identification of which plant a particular simple leaf belongs to. And the third unit identifies the species of compound leaves. The unique feature of our system is the ability to differentiate simple leaves from compound leaves. It can also distinguish one type of compound leaf from another. All the previous works on this topic, that we have studied, dealt with simple leaves only. This is the biggest achievement of our work. Our system can also identify which plant a leaf belongs to even when the leaf is partially damaged or broken. 1200 simple leaves from 30 types of plants and 200 compound leaves from 5 types of plants have been used to train our system. Using 10-fold cross validation our system shows 91.41% recognition accuracy for simple leaves and 98.5% accuracy for compound leaves. And the overall accuracy of our system is 92.42%.

The biggest limitation of our system is, it requires user help in the preprocessing stage. Another limitation is its inability to work with images with complicated background. We would like to overcome these limitations in our future work and make the system even more robust.

Bibliography

[1] S. Wu, F. Bao, E. Xu, Y. Wang, Y. Chang, and Q. Xiang, "A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network," IEEE 7th International Symposium on Signal Processing and Information Technology, December 2007.

[2] Z. Wang, Z. Chi, and D. Feng, "Shape based leaf image retrieval," IEEE Proceedings-Vision, Image and Signal Processing, vol. 150, no. 1, February 2003.

[3] H. Fu, Z. Chi, D. Feng, and J. Song, "Machine learning techniques for ontology-based leaf classification," IEEE 2004 8th International Conference on Control, Automation, Robotics and Vision, Kunming, China, 2004.

[4] X. Wang, D. Huang, J. Dua, H. Xu, L. Heutte, "Classification of plant leaf images with complicated background", Special Issue on Advanced Intelligent Computing Theory and Methodology in Applied Mathematics and Computation, Volume 205, Issue 2, Pages 916-926, 15 November 2008.

[5] J. Du, D. Huang, X. Wang, and X. Gu, "Computer-aided plant species identification (CAPSI) based on leaf shape matching technique," Transactions of the Institute of Measurement and Control. 28, 3 (2006) pp. 275-284.

[6] Y. Ye, C. Chen, C.-T. Li, H. Fu, and Z. Chi, "A computerized plant species recognition system," Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing, Hong Kong, October 2004.

[7] Y. Li, Q. Zhu, Y. Cao, and C. Wang, "A leaf vein extraction method based on snakes technique," in Proceedings of IEEE International Conference on Neural Networks and Brain, 2005.

[8] X. Gu, J. Du, and X. Wang, “Leaf recognition based on the combination of wavelet transform and gaussian interpolation,” in Proceedings of International Conference on Intelligent Computing 2005, ser. LNCS 3644. Springer, 2005.

[9] X. Wang, J. Du, and G. Zhang, “Recognition of leaf images based on shape features using a hypersphere classifier,” in Proceedings of International Conference on Intelligent Computing 2005, ser. LNCS 3644. Springer, 2005.

[10] J. Du, D. Huang, X. Wang, and X. Gu, “Shape recognition based on radial basis probabilistic neural network and application to plant species identification,” in Proceedings of 2005 International Symposium of Neural Networks, ser. LNCS 3497. Springer, 2005.

[11] F. Mokhtarian, S. Abbasi, “Matching shapes with self-intersection: application to leaf classification”, IEEE Trans. Image Process. 13 (5) (2004) 653–661.

[12] P. Wasserman, (1993): Advanced Methods in Neural Computing, Van Nostrand Reinhold, New York.

[13] D. Specht, (1990): “Probabilistic Neural Networks,” Neural Networks, No. 3, pp. 109-118.

[14] S. Kung, & J. Taur, (1995): “Decision-based neural networks with signal/image classification applications,” IEEE Trans. on Neural Networks, No. 6, pp. 170–181.

[15] F. Mokhtarian, S. Abbasi, and J. Kittler. “Efficient and robust retrieval by shape content through curvature scale space,” in Proceedings of the International Workshop Image DataBases and MultiMedia Search 1996, 35-42.

[16] C. Im, H. Nishida, T. Kunii, “Recognizing plant species by leaf shapes – a case study of the Acer family,” Proc. Pattern Recog. 2 (1998) 1171–1173.

[17] Simple and Compound Leaves. Retrieved March 14, 2010, from http://www.tutorvista.com/content/biology/biology-iii/angiosperm-morphology/simple and-compound-leaves.php.

[18] Leaf Morphology. Retrieved January 14, 2010, from http://generalhorticulture.tamu.edu/h202/labs/lab2/leafm.html.

[19] Parts of a leaf. Retrieved January 16, 2010, from http://www.botanical-online.com/lahojaangles.htm.

[20] The Parts of a Leaf. Retrieved January 16, 2010, from http://www.robinsonlibrary.com/science/botany/anatomy/leafparts.htm.

[21] Simple leaf dataset, Retrieved October 18, 2010, from http://sourceforge.net/projects/flavia/files/Leaf%20Image%20Dataset/1.0/Leaves.tar.bz2/download.

[22] R. Gonzalez, R. Woods, (2002): Digital Image Processing, Pearson Education, Inc., Delhi, India.

[23] Simulation of Photoshop's magic wand tool, Retrieved April 2, 2010, from http://www.mathworks.com/matlabcentral/fileexchange/6034.

Source Code

```
function binaryLeaf = extractLeaf(image)

%*************************************************************************
% EXTRACTLEAF extracts the leaf from an RGB or gray image, sets the pixels
% in the leaf to 1 (white) and rest of image to 0(black) then returns the
% binary leaf.
%
% PARAMETERS:
%       image = an RGB or gray leaf image
%
% OUTPUT:
%       binaryLeaf = a binary image where the parts of leaf is white and
%       rest of the image is black
%
% MODIFIED BY:
%       Javed Hossain
%       Date: April 20, 2010.
% ORIGINAL VERSION BY
%       Daniel Leo Lau, April 7, 1997
%       email: lau@ece.udel.edu  (mex file)
% UPDATED FOR MATLAB BY
%       Yoram Tal, June 30, 2003
%       email: yoram_tal@yahoo.com (mex file)
%*************************************************************************

% Set tolerance to 80
tolerance = 80;

% Get points interactively, until right mouse button is clicked
but   = 0;
ii    = 0;
xlist = [];
ylist = [];
hplot = [];
hold on
while but ~= 3,
    ii = ii + 1;
    [x, y, but] = ginput(1);
    xlist(ii)   = round(x);
    ylist(ii)   = round(y);
    hplot(ii)   = plot(x,y,'*'); % plot the points immediately
end
xlist(end) = [];
ylist(end) = [];
delete(hplot);
hold off

% Check points validity
if isempty(xlist) || isempty(ylist),
    error('Point list is empty');
end
```

```
H = size(image, 1); % image height
W = size(image, 2); % image width

% Check if any point is out of range
k = ylist > 0 & ylist <= H;
k = k & xlist > 0 & xlist <= W;
if ~any(k),
    error('Coordinates out of range');
elseif ~all(k),
    disp('Warning: some coordinates out of range');
end

ylist = ylist(k);
xlist = xlist(k);
N = length(ylist); % Number of reference pixels

%Create the binary mask
color_mask = false(H, W);

if ndims(image) < 3,
    g = double(image);
    for i = 1:N,
        ref = double(image(ylist(i),xlist(i)));
        color_mask = color_mask | (g - ref).^2 <= tolerance^2;
    end
elseif ndims(image) == 3,
    c_r = double(image(:, :, 1)); % Red channel
    c_g = double(image(:, :, 2)); % Green channel
    c_b = double(image(:, :, 3)); % Blue channel
    for i = 1:N,
        ref_r = double(image(ylist(i), xlist(i), 1));
        ref_g = double(image(ylist(i), xlist(i), 2));
        ref_b = double(image(ylist(i), xlist(i), 3));
        color_mask = color_mask | ...
            ((c_r - ref_r).^2 + (c_g - ref_g).^2 + (c_b - ref_b).^2)...
             <= tolerance^2;
    end
end

% Connected component labelling
[objects, count] = bwlabel(color_mask, 8);
[y x v] = find(objects);
segList = [];

for i = 1:N,
    k = find(x == xlist(i) & y == ylist(i));
    segList = [segList; v(k)];
end

segList = unique(segList);
LUT = zeros(1,count+1);
LUT(segList+1) = 1;
binaryLeaf = LUT(objects+1);

end
```

```
function [x y] = getLeftMostPoint(image)

%*************************************************************************
% GETLEFTMOSTPOINT returns the left most point of the binary input image.
%
% PARAMETERS:
%       image = a binary leaf image
%
% OUTPUT:
%       x = x-coordinate of the left most point of binary leaf I
%       y = y-coordinate of the left most point of binary leaf I
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Initialize the coordinates of the left most point
x = -1;
y = -1;

[height width] = size(image);

% Start checking from the left most column of the image and return the
% coordinates of the white(1) pixel that is found first.
for c = 1:width
    for r = 1:height
     if( image(r,c)==1 )
         x = r;
         y = c;
         return;
     end;
   end;
end;

end
```

```
function boundary = getBoundary(I, x, y)

%*************************************************************************
% GETBOUNDARY gets the coordinates of the boundary pixels of the binary
% leaf I. The boudary tracing starts at pixel (x,y) and terminates
% when it arrives this starting point.
%
% PARAMETERS:
%       I = a binary leaf image
%       X = x-coordinate of the starting point of binary leaf I
%       Y = y-coordinate of the starting point of binary leaf I
%
% OUTPUT:
%       boundary = the coordinates of the boundary pixels of the binary
% leaf I
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Initialize boundary
boundary = -1;

% Extract the boundary with initial search direction set to North (N)
try
    boundary = bwtraceboundary(I,[x, y],'N');
catch
    try % Try East (E) is North fails
        boundary = bwtraceboundary(I,[x, y],'E');
    catch
        try % Try South (S) is East fails
            boundary = bwtraceboundary(I,[x, y],'S');
        catch
            try % Try West (W) is East fails
                boundary = bwtraceboundary(I,[x, y],'W');
            catch
                error('Boundary extraction failed.');
            end
        end
    end
end

end
```

```
function [x y] = getClosestBoundaryPoint(I,X,Y)

%*************************************************************************
% GETCLOSESTBOUNDARYPOINT Finds the closest point on the contour/boundary
% of binary leaf I from the base-point (X,Y).
%
% PARAMETERS:
%       I = a binary leaf image
%       X = x-coordinate of the base-point of binary leaf I
%       Y = y-coordinate of the base-point of binary leaf I
%
% OUTPUT:
%       x = x-coordinate of the closest contour/boundary point from (X,Y)
%       y = y-coordinate of the closest contour/boundary point from (X,Y)
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Get the left most point of binary leaf image I
[x y] =  getLeftMostPoint(I);

% Get the boundary clockwise starting from (x,y)
boundary = getBoundary(I,x,y);

[r c] = size(boundary);

% Calculate the distances of all the points on the border from the
% point (x,y) and remember the point with minimum distance

min_d = sqrt(( X - boundary(1,2) )^2 + ( Y - boundary(1,1) )^2);
min_index = 1;
for m = 2:r
    d = sqrt((X - boundary(m,2) )^2 + ( Y - boundary(m,1))^2);
    if d < min_d
        min_d = d;
        min_index = m;
    end
end

% Store the x and y-coordinates of the closest point
x = boundary(min_index,1);
y = boundary(min_index,2);

end
```

```
function AL = alignLeaf(I, X, Y)

%*************************************************************************
% ALIGNLEAF aligns a binary leaf, I, provided the base-point coordinate X
%   and Y.
%
% PARAMETERS:
%       I = a binary leaf image
%       X = x-coordinate of the base-point of binary leaf I
%       Y = y-coordinate of the base-point of binary leaf I
%
% OUTPUT:
%       AL = The horizontally aligned binary leaf with its base-point on
%       the left.
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Get the boundary of the leaf using X and Y
boundary = getBoundary(I,X,Y);

% Get the Base-point Contour Distance vector
dist = getDistanceVector(boundary, X, Y);

% Find the furthest point (tip of the leaf)
[r c] = size(dist);
maxD = dist(1,1);
max_index = 1;
for i = 1:r
    if maxD < dist(i,1)
        maxD = dist(i,1);
        max_index = i;
    end
end
mx = boundary(max_index,1);
my = boundary(max_index,2);

% Find out the inclination of the line connecting base-point and tip
u = x - mx;
v = y - my;

angle = (180/pi)*atan(abs(u/v));

% Based on the orientation of the leaf correct the angle
% so the base-point is on the left after rotation.


if u < 0 && v < 0 %04:30
    angle = angle + 0;
end

if u > 0 && v > 0 %10:30
```

```
    angle = angle + 180;
end

if u > 0 && v < 0 %01:30
    angle = 0 - angle;
end

if u < 0 && v > 0 %07:30
    angle = 180 - angle;
end

if u == 0 && v < 0 %03:00
    angle = 0;
end

if u == 0 && v > 0 %09:00
    angle = 180;
end

if u > 0 && v == 0 %12:00
    angle = -90;
end

if u < 0 && v == 0 %06:00
    angle = 90;
end

% Rotate the leaf
R = imrotate(I, angle);

% Crop the leaf
P = regionprops(R,'Image');

AL = P.Image;

end
```

```
function DV = getDistanceVector(contour, x, y)

%*************************************************************************
% GETDISTANCEVECTOR calculates the distances of all the points in the
% contour vector from the point (x,y), then returns the distance vector.
%
% PARAMETERS:
%       contour = the coordinates of the boundary pixels of a leaf
%       x = x-coordinate of the reference point to compute distance from
%       y = y-coordinate of the reference point to compute distance from
%
% OUTPUT:
%       DV = the coordinates of the boundary pixels of the binary
% leaf I
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Get the size of contour matrix
[r c] = size(contour);

% Initialize distance vector
d = zeros(r,1);

% Calculate the distances of all the countour points from the point (x,y)
for i = 1:r
    x1 = contour(i,1);
    y1 = contour(i,2);
    dist = sqrt((x - x1)^2 + (y - y1)^2);
    d(i,1) = dist;
end

% Find the maximum distance
maxDist = max(d);

% Normalize the distances by dividing all of them by the max distance
for i = 1:r
    d(i,1) = d(i,1)/maxDist;
end

DV = d;
end
```

```
function [vc pc] = getFilteredDFT(distCurve, boundary)

%*************************************************************************
% GETFILTEREDDFT smooths the BCD vector of a leaf by applying FFT and IFFT,
% counts the number of valleys in the smoothed BCD and the number of peaks
% that occurred the maximum times.
%
% PARAMETERS:
%       distCurve = BCD curve of a leaf
%       boundary = boundary of a leaf
%
% OUTPUT:
%       vc = valley count of the smoothed BCD
%       pc = peak count of the smoothed BCD
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

d1 = distCurve(:,1);

% Smooth and count peaks over 30 iterations using FFT and IFFT
ind = 1;
for j = 1:1:30
    F = fft(d1);
    AF = abs(F);
    [r c] = size(F);
    F2 = zeros(r,c);
    thresh = j;
    for i = 1:r
        if AF(i,1) >= thresh % thresh to filter out sine waves with low amp
            F2(i,1) = F(i,1);
        end
    end
    F3 = ifft(F2);
    N = nCount(F3);
    Na(ind,1) = N(1,3);
    ind = ind + 1;
end

% Find out the boundary pixels corresponding to the valleys
N = nCount(F3);
dec_arr_size = N(1,3) - 1;
dec_arr = zeros(dec_arr_size,2);
for i = 1:N(1,1)
    v = N(2,i);
    if v ~= 0
        dec_arr(i,1) = boundary(v,1);
        dec_arr(i,2) = boundary(v,2);
    end
end

count = 0;
% Compute distances among the valleys
```

```
for i = 1:dec_arr_size
    x1 = dec_arr(i,1);
    y1 = dec_arr(i,2);
    for j = i+1:dec_arr_size
        x2 = dec_arr(j,1);
        y2 = dec_arr(j,2);
        distance = sqrt((x1 - x2)^2 + (y1 - y2)^2);
        if distance <= 0.15
            count = count + 1;
        end
    end
end
pc = maxOccurance(Na);
vc = count;

end

function out = nCount( dist )

    [r c] = size(dist);
    inc = zeros(r,1);
    dec = zeros(r,1);
    all = zeros(r,1);
    N = 0;
    out = zeros(4,r);

    increasing = 0;
    in_count = 0;

    decreasing = 0;
    de_count = 0;

    thresh = 1;
    for i = 1:r-1
        if dist(i+1,1) > dist(i,1)
            %Change from decreasing to increasing
            if decreasing >= thresh
                N = N + 1;
                decreasing = 0;
                in_count = in_count + 1;
                inc(in_count,1) = i;
                all(N,1) = dist(i,1);
            end
            increasing = increasing + 1;
        end
        if dist(i+1,1) < dist(i,1)
            %Change from increasing to decreasing
            if increasing >= thresh
                N = N + 1;
                increasing = 0;
                de_count = de_count + 1;
                dec(de_count,1) = i;
                all(N,1) = dist(i,1);
            end
            decreasing = decreasing + 1;
        end
```

```
    end

    out(1,1) = N;
    out(1,2) = in_count;
    out(1,3) = de_count;
    out(2,:) = inc(:,:);
    out(3,:) = dec(:,:);
    out(4,:) = all(:,:);
end
```

```
function M = maxOccurance(A)

%*************************************************************************
% MAXOCCURANCE returns the value that occurred the highest number of times in
% the input array.
%
% PARAMETERS:
%       A = an array of value
%
% OUTPUT:
%       M = the value that occurred the highest number of times in
%       the input array A.
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Get the maximum value in array A
mx = max(A);

% Create an array of size A
arr = zeros(mx,1);
[r c]= size(A);

% Count the number of times each value has occurred in array A
for i = 1:r
    arr(A(i,1),1) = arr(A(i,1),1) + 1;
end
max_occur = 1;

% Find the value with highest occurance
for i = 1:mx
    if arr(i,1) > arr(max_occur,1)
        max_occur = i;
    end
end
% return the value with highest occurrance
M = max_occur;
end
```

```
function leafType = identifyLeaf(I,x,y)

%*************************************************************************
% IDENTIFYLEAF identifies whether the input binary leaf is simple or
% compound, provided the coordinates of the base-point.
%
% PARAMETERS:
%       I = a preprocessed binary leaf image
%       X = x-coordinate of the base-point of binary leaf I
%       Y = y-coordinate of the base-point of binary leaf I
%
% OUTPUT:
%       leafType = type of the leaf, possible return values are 'simple',
%       'compound' and 'simple/compound'(returned if identification fails).
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Get the boundary of the input binary image
B = getBoundary(I,x,y);

% Get the Base-point Contour Distance (BCD) vector of the input leaf
D = getDistanceVector(B,x,y);

% Smooth the BCD vector
F = getFilteredDFT(D,B,I);

% Get the largest connected component in input image
BW1 = I;
CC = bwconncomp(BW1);
[h w] = size(BW1);
BW2 = zeros(h,w);
numPixels = cellfun(@numel,CC.PixelIdxList);
[biggest,idx] = max(numPixels);
BW2(CC.PixelIdxList{idx}) = 1;

% Get the Filled Area and Area(normal) of the input leaf
P = regionprops(BW2,'FilledArea','Area');
Ar = P.Area;
Far = P.FilledArea;

% Find the amount of holes in the leaf
holeArea = ((Far - Ar)/Ar)*100;

% Get the number of valleys in the BCD vector
totalValleyCount = F(1,1);

% Get the number of valleys in the BCD vector that are very close
closeValleyCount = F(1,2);

% Classify the input leaf based on the features extracted above
if totalValleyCount == 1
```

```
    leafType = 'Simple';
elseif totalValleyCount > 0 && holeArea > 1
    leafType = 'Compound';
elseif holeArea > 2
    leafType = 'Compound';
elseif totalValleyCount > 0 && closeValleyCount > 1 && holeArea < 0.5
    leafType = 'Compound';
elseif totalValleyCount > 0 && holeArea < 0.5
    leafType = 'Simple/Compound';
end

end
```

```
function PSF = getPSF()

%*************************************************************************
% GETPSF extracts the Primary Shape Features of all 1200 simple leaves and
% returns the feature matrix.
%
% PARAMETERS:
%       None
%
% OUTPUT:
%       PSF = the feature matrix of size 8 x 1200
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

    % Initialize feature matrix PSF
    PSF = zeros(8,1200);

    for i = 1:1200

        % Read the image and convert to black and white (BW) image
        filename = strcat(int2str(1000 + f),'.bmp');
        I = im2bw(imread(filename));

        % Crop the image, keep only leaf
        P = regionprops(I, 'Image');
        C = P.Image;

        % Get region properties of binary leaf I
        PROPS = regionprops(I,'Eccentricity','Area','ConvexArea','Perimeter',
                        'EquivDiameter','MajorAxisLength','MinorAxisLength',
                        'Extent','Solidity');

        % Extract the Primary Shape Features (PSF)
        ec = PROPS.Eccentricity;
        ar = PROPS.Area;
        ca = PROPS.ConvexArea;
        pr = PROPS.Perimeter;
        ed = PROPS.EquivDiameter;
        mj = PROPS.MajorAxisLength;
        mn = PROPS.MinorAxisLength;
        ex = PROPS.Extent;
        sd = PROPS.Solidity;

        % Normalize features
        ar = ar/mj;
        pr = pr/mj;
        mr = mn/mj;
        ed = ed/mj;
        ca = (ca-PROPS.Area)/PROPS.Area;

        % Store features
        features(1,i) = ec;
```

```
        features(2,i) = sd;
        features(3,i) = ar;
        features(4,i) = pr;
        features(5,i) = mr;
        features(6,i) = ed;
        features(7,i) = ex;
        features(8,i) = ca;
        PSF(:,i) = features(:,i);

        % Print progress
        fprintf('Progress: %g',100*i/1200);
        disp(' ');
    end
end
```

```
function LWF = getWidthVector(image, length)

%*************************************************************************
% GETWIDTHVECTOR extracts and return the Laminal Width Factor (LWF) of a
% binary leaf.
%
% PARAMETERS:
%       image = an aligned binary (fully preprocessed) simple leaf
%       length = specifies the length of the output vector LWF
%
% OUTPUT:
%       LWF = The Laminal Width Factor of the input simple leaf
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

    % Get the width of the image, which is actually the length of the leaf
    C = image;
    [r c] = size(C);
    leafLength = c;

    % Calculate the interval based on the specified length (of LWF)
    interval = floor(leafLength /(length+1));
    x = interval;

    % Calculate the LWF
    for i = 1:length
        col = C(:,x);
        w = sum(col); % width of column x of image C
        LWF(i,1) = w/leafLength; % normalize with the leaf length
        x = x + interval;
    end
end
```

```
function CPSF = getCompoundPSF()

%*************************************************************************
% GETCOMPOUNDPSF extracts the Primary Shape Features of all 200 compound
% leaves and returns the feature matrix.
%
% PARAMETERS:
%       None
%
% OUTPUT:
%       CPSF = the feature matrix of size 9 x 200
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Initialize feature matrix with 0's
CPSF = zeros(9,200);

for i = 1:200

    % Read image
    filename = strcat(int2str(400 + f),'.bmp');
    I = imread(filename);

    % Get region properties of binary leaf I
    PROPS = regionprops(I,'Eccentricity','Area','ConvexArea','Perimeter',
                        'EquivDiameter','MajorAxisLength','MinorAxisLength',
                        'Extent','Solidity','FilledArea','FilledImage');

    % Extract the features
    ec = PROPS.Eccentricity;
    ar = PROPS.Area;
    ca = PROPS.ConvexArea;
    pr = PROPS.Perimeter;
    ed = PROPS.EquivDiameter;
    mj = PROPS.MajorAxisLength;
    mn = PROPS.MinorAxisLength;
    ex = PROPS.Extent;
    sd = PROPS.Solidity;
    fa = PROPS.FilledArea;

    % Normalize features
    ar = ar/mj;
    pr = pr/mj;
    mr = mn/mj;
    ed = ed/mj;
    ci = (ca-ar)/ar;
    fa = (fa-ar)/ar;

    % Store features
    features(1,i) = ec;
    features(2,i) = sd;
    features(3,i) = ar;
```

```
    features(4,i) = pr;
    features(5,i) = mr;
    features(6,i) = ed;
    features(7,i) = ex;
    features(8,i) = ci;
    features(9,i) = fa;

    CPSF(:,i) = features(:,i);

    % Print progress
    fprintf('Progress: %g',100*i/200);
    disp(' ');
end

end
```

script PNN

```
%*************************************************************************
% PNN Creates a Probabilistic Neural Network for simple leaves. It
% trains the network and tests the network using 10-fold cross-validation
% technique. Accuracies of each fold is returned in an array as output.
%
% MODIFIED BY:
%       Javed Hossain
%       Date: April 20, 2010.
%
% ORIGINAL VERSION BY:
%       Dr. Ashraful Amin
%*************************************************************************

% Load the target class vector for PNN training.
load('T30.mat');

for n = 1:1

    % Obtain primary shape features for all 1200 simple leaves
    PSF = getPSF(1,1200);

    % Obtain Laminal Width Vector for all 1200 simple leaves
    WVR = getWidthVector(1,1200,11);

    % Merge PSF and WVR to create input vector for PNN
    P = [WVR PSF];

    P_norm = P;
    Pt = P';

    % Get arithmatic mean and standard deviations of the features
    RowMeans = mean(Pt);
    RowStds = std(Pt);

    [r c]=size(P);

    % Normalize features
    for j = 1:c
      P_norm(:,j) = P(:,j) - RowMeans';
      P_norm(:,j) = P_norm(:,j) ./ RowStds';
    end;

    clear RowMeans RowStds Pt c i r;

    % Partition P and T ( for 10-fold cross-validation )
    for j = 0:9
        testsize = 10; % 10 for 10-fold corss-validation
        ind1 = 1;
        ind2 = 1;
        for i = 1:1200
            if mod(i,testsize)== j % Testing partition
                temp1(:,ind1)= P_norm(:,i);
```

```
            temp2(:,ind1)=T(:,i);
            ind1=ind1+1;
        else % Training partition
            temp3(:,ind2)=P_norm(:,i);
            temp4(:,ind2)=T(:,i);
            ind2=ind2+1;
        end
    end

    % Make the PNN with training partition
    net = newpnn(temp3,temp4);

    % Test the PNN with testing partition
    Y = sim(net,temp1);
    Yc = vec2ind(Y)';

    [r c] = size(Yc);
    temp = 0;
    Yc_actual = vec2ind(temp2)';

    % Check the output of the PNN, count number of mistakes
    for i=1:r
          if (Yc_actual(i,1)~= Yc(i,1))
             temp = temp + 1;
          end
    end

    % Store the accuracy
    a(1,j+1) = 100-100*(temp/r);

    clear temp temp1 temp2 temp3 temp4 ind1 ind2 i j  r c Y testsize;
  end
end
```

script CPNN

```
%*************************************************************************
% CPNN Creates a Probabilistic Neural Network for compound leaves. It
% trains the network and tests the network using 10-fold cross-validation
% technique. Accuracies of each fold is returned in an array as output.
%
% MODIFIED BY:
%       Javed Hossain
%       Date: April 20, 2010.
%
% ORIGINAL VERSION BY:
%       Dr. Ashraful Amin
%*************************************************************************

% Load the target class vector for PNN training.
load('CT.mat');

for n = 1:1

    % Get input vector (feature matrix of preprocessed compound leaves)
    P = getCompoundPSF(1,200);

    T = CT;
    P_norm = P;
    Pt = P';

    % Get arithmatic mean and standard deviations of the features
    RowMeans = mean(Pt);
    RowStds = std(Pt);

    [r c]=size(P);

    % Normalize features
    for j = 1:c
      P_norm(:,j) = P(:,j) - RowMeans';
      P_norm(:,j) = P_norm(:,j) ./ RowStds';
    end;

    clear RowMeans RowStds Pt c i r;

    % Partition P and T ( for 10-fold cross-validation )
    for j = 0:9
        testsize = 10; % 10 for 10-fold corss-validation
        ind1 = 1;
        ind2 = 1;
        for i = 1:200
            if mod(i,testsize)== j % Testing partition
                temp1(:,ind1)= P_norm(:,i);
                temp2(:,ind1)=T(:,i);
                ind1=ind1+1;
            else % Training partition
                temp3(:,ind2)=P_norm(:,i);
                temp4(:,ind2)=T(:,i);
```

```
            ind2=ind2+1;
        end
    end

    % Make the PNN with training partition
    net = newpnn(temp3,temp4);

    % Test the PNN with testing partition
    Y = sim(net,temp1);
    Yc = vec2ind(Y)';

    [r c] = size(Yc);
    temp = 0;
    Yc_actual = vec2ind(temp2)';

    % Check the output of the PNN, count number of mistakes
    for i=1:r
        if (Yc_actual(i,1)~= Yc(i,1))
            temp = temp + 1;
        end
    end

    % Store the accuracy
    a(1,j+1) = 100-100*(temp/r);

    clear temp temp1 temp2 temp3 temp4 ind1 ind2 i j  r c Y testsize;
  end
end
```

```
function Yc = searchSimpleLeaves(I)

%*************************************************************************
% SEARCHSIMPLELEAVES searches using a pre-trained PNN for compound leaves
% and returns which class the input leaf belongs to.
%
% PARAMETERS:
%       I = a fully preprocessed binary leaf image
%
% OUTPUT:
%       Yc = value indicating which of the 5 classes the compound
%       leaf belongs to
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Load the PNN trained with simple leaves
load('SimpleLeafPNN.mat');

% Fill any holes that might exist in the input image
BW1 = imfill(I,'holes');

% Find the largest connected component
CC = bwconncomp(BW1);
[h w] = size(BW1);
BW2 = zeros(h,w);
numPixels = cellfun(@numel,CC.PixelIdxList);
[biggest,idx] = max(numPixels);
BW2(CC.PixelIdxList{idx}) = 1;
I = BW2;

% Get region properties of binary leaf I
PROPS =
regionprops(I,'Eccentricity','Area','ConvexArea','Perimeter','EquivDiameter',
'MajorAxisLength','MinorAxisLength','Extent','Solidity');


% Extract the shape features
ec = PROPS.Eccentricity;
ar = PROPS.Area;
ca = PROPS.ConvexArea;
pr = PROPS.Perimeter;
ed = PROPS.EquivDiameter;
mj = PROPS.MajorAxisLength;
mn = PROPS.MinorAxisLength;
ex = PROPS.Extent;
sd = PROPS.Solidity;

% Normalize the features
ar = ar/mj;
pr = pr/mj;
mr = mn/mj;
ed = ed/mj;
```

```
ca = (ca-PROPS.Area)/PROPS.Area;

% Store the features in an array
features(1,1) = ec;
features(2,1) = sd;
features(3,1) = ar;
features(4,1) = pr;
features(5,1) = mr;
features(6,1) = ed;
features(7,1) = ex;
features(8,1) = ca;

% Get LWF of the input image
wv = getWidthVector(I,11);

% Combine all the features
F(1:11,1) = wv;
F(12:19,1) = features;

% Normalize all features
F(:,1) = F(:,1) - RowMeans';
F(:,1) = F(:,1) ./ RowStds';

% Use the final feature array F with the PNN to find out the plant class
Y = sim(net,F);
Yc = vec2ind(Y)';

end
```

```
function Yc = searchCompoundLeaves(I)

%*************************************************************************
% SEARCHCOMPOUNDLEAVES searches using a pre-trained PNN for compound leaves
% and returns which class the input leaf belongs to.
%
% PARAMETERS:
%       I = a fully preprocessed binary leaf image
%
% OUTPUT:
%       Yc = value indicating which of the 5 classes the compound
%       leaf belongs to
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Load the PNN trained with compound leaves
load('CompoundLeafPNN.mat');

% Get region properties of binary leaf I
PROPS =
regionprops(I,'Eccentricity','Area','ConvexArea','Perimeter','EquivDiameter',
'MajorAxisLength','MinorAxisLength','Extent','Solidity','FilledArea','FilledI
mage');

% Extract the shape features
ec = PROPS.Eccentricity;
ar = PROPS.Area;
ca = PROPS.ConvexArea;
pr = PROPS.Perimeter;
ed = PROPS.EquivDiameter;
mj = PROPS.MajorAxisLength;
mn = PROPS.MinorAxisLength;
ex = PROPS.Extent;
sd = PROPS.Solidity;
fa = PROPS.FilledArea;

% Normalize the features
ar = ar/mj;
pr = pr/mj;
mr = mn/mj;
ed = ed/mj;
ci = (ca-ar)/ar;
fa = (fa-ar)/ar;

% Store the features in an array
features(1,1) = ec;
features(2,1) = sd;
features(3,1) = ar;
features(4,1) = pr;
features(5,1) = mr;
features(6,1) = ed;
features(7,1) = ex;
```

```
features(8,1) = ci;
features(9,1) = fa;

% Normalize features
features(:,1) = features(:,1) - RowMeans';
features(:,1) = features(:,1) ./ RowStds';

% Use the feature array with the PNN to find out the plant class
Y = sim(net,features);
Yc = vec2ind(Y)';

end
```

```
function [s_name l_name] = getPlantName( plantIndex )

%*************************************************************************
% GETPLANTNAME gets the local and scientific name of a plant based on the
% plant index.
%
% PARAMETERS:
%       plantIndex = predetermined index associated with the plant species
%
% OUTPUT:
%       s_name = scientific name of the plant
%       l_name = local name of the plant
%
% AUTHOR:
%       Javed Hossain
%       Date: April 20, 2010.
%*************************************************************************

% Initialize plant names
s_name = 'Unknown plant';
l_name = 'Unknown plant';

% Get the plant names based on the plant index.
switch plantIndex
    case 101
        s_name = ' Oxalis Arborea';
        l_name = ' Wood Sorrel';
    case 102
        s_name = ' Sesbania Drummondii';
        l_name = ' Rattlebush';
    case 103
        s_name = ' Senna Pendula';
        l_name = ' Cassia';
    case 104
        s_name = ' Cosmos Sulphureus';
        l_name = ' Orange Cosmos';
    case 105
        s_name = ' Moringa Oleifera';
        l_name = ' Sajna';
    case 1
        s_name = ' Phyllostachys Edulis';
        l_name = ' Pubescent Bamboo';
    case 2
        s_name = ' Aesculus Chinensis';
        l_name = ' Chinese Horse Chestnut';
    case 3
        s_name = ' Cercis Chinensis';
        l_name = ' Chinese Redbud';
    case 4
        s_name = ' Indigofera Tinctoria';
        l_name = ' True Indigo';
    case 5
        s_name = ' Acer Palmatum';
        l_name = ' Japanese Maple';
    case 6
        s_name = ' Phoebe Nanmu';
```

```
    l_name = ' Nanmu';
case 7
    s_name = ' Koelreuteria Paniculata';
    l_name = ' Castor Aralia';
case 8
    s_name = ' Kalopanax Septemlobus';
    l_name = ' Goldenrain Tree';
case 9
    s_name = ' Cinnamomum Japonicum';
    l_name = ' Chinese Cinnamon';
case 10
    s_name = ' Berberis Anhweiensis';
    l_name = ' Anhui Barberry';
case 11
    s_name = ' Ilex Macrocarpa';
    l_name = ' Big-Fruited Holly';
case 12
    s_name = ' Pittosporum Tobira';
    l_name = ' Japanese Cheesewood';
case 13
    s_name = ' Chimonanthus Praecox.';
    l_name = ' Wintersweet';
case 14
    s_name = ' Cinnamomum Camphora';
    l_name = ' Camphortree';
case 15
    s_name = ' Viburnum Awabuki';
    l_name = ' Japan Arrowwood';
case 16
    s_name = ' Osmanthus Fragrans';
    l_name = ' Sweet Osmanthus';
case 17
    s_name = ' Cedrus Deodara';
    l_name = ' Deodar';
case 18
    s_name = ' Lagerstroemia Indica';
    l_name = ' Crape Myrtle';
case 19
    s_name = ' Nerium Oleander';
    l_name = ' Oleander';
case 20
    s_name = ' Podocarpus Macrophyllus';
    l_name = ' Yew Plum Pine';
case 21
    s_name = ' Prunus Serrulata';
    l_name = ' Japanese Cherry ';
case 22
    s_name = ' Ligustrum Lucidum';
    l_name = ' Glossy Privet';
case 23
    s_name = ' Tonna Sinensis';
    l_name = ' Chinese Toon';
case 24
    s_name = ' Prunus Persica';
    l_name = ' Peach';
case 25
    s_name = ' Manglietia Fordiana';
```

```
        l_name = ' Ford Woodlotus';
    case 26
        s_name = ' Acer Buergerianum';
        l_name = ' Trident Maple';
    case 27
        s_name = ' Mahonia Bealei';
        l_name = ' Beale`s Barberry';
    case 28
        s_name = ' Magnolia Grandiflora';
        l_name = ' Southern Magnolia';
    case 29
        s_name = ' Populus Canadensis';
        l_name = ' Canadian Poplar';
    case 30
        s_name = ' Citrus Reticulata';
        l_name = ' Tangerine';
end
```