University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Plant Leaf Recognition by Leaf Venation and Shape using Artificial Neural Networks (LeaVeS)

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Azeil Louisse Codizar

May 2014

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Plant Leaf Recognition by Leaf Venation and Shape using Artificial Neural Networks (LeaVeS)" prepared and submitted by Azeil Louisse Codizar in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Geoffrey A. Solano, M.Sc.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*candidate*)	_____	_____
2. Avegail D. Carpio, M.Sc.	_____	_____
3. Aldrich Colin K. Co, M.Sc. (*candidate*)	_____	_____
4. Ma. Sheila A. Magboo, M.Sc.	_____	_____
5. Vincent Peter C. Magboo, M.D., M.Sc.	_____	_____
6. Richard Bryann L. Chua, M.Sc.	_____	_____
7. Bernie B. Terrado, M.Sc. (*candidate*)	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Alex C. Gonzaga, Ph.D., Dr.Eng.
Dean
College of Arts and Sciences

Abstract

A leaf recognition system for plant classification by a leaf's shape and venation using the centroid-radii model, moment-invariant model, canny edge detection, morphological operations, image difference and artificial neural networks.

Keywords: plant leaf recognition, leaf shape, venation, artificial neural networks, centroid-radii, moment-invariant, canny edge detection, geographic information systems

Contents

List of Figures

I. Introduction

A. Background of the Study

It is but common knowledge that plants are vital to human existence. Be it because of medicinal purposes, or energy purposes, [1] or other purposes that may be more obvious than those stated or less, it is undeniable that to be able to survive in this world, humans need plants. However, as time goes by, the number of known and unknown plant species increases [2] . And this is the primary reason why the classification of plants is of prime importance - to understand and identify numerous species.

But the classification of plants is not an easy task, as though it may seem. In fact, there are various ways to do so (manual and automatic). Manual methods include expert determination and involves a lot of time. Also, only the experts can accurately identify a plant that a normal person without any training could only put his chances on. Thanks to the advent of technology, there are automatic and faster ways to classify a plant even without sufficient training in the field of biology, or phytochemistry, or the like. Even so, plant classification is still too broad a topic. A plant can be classified through its flower, fruit, stem, branch, or leaf. [3]

Leaf Recognition stands out among the said automatic classification methods because: first, it is convenient; second, it is low-cost; [4] and third, it is relatively-simpler. [3] As we are taught, a leaf has distinct characteristics. And one of the most important is the leaf shape [5] which can easily be represented in 2D. After shape, comes venation (vein structure) as the most studied aspect of the leaf. [6]

Fortunately, there exist reliable and effective systems to recognize a plant's leaf. One such system used Canny Edge Detection, the Centroid-Radii Model and Hu's moment features to be able to come up with a vector descriptor of the shape of the leaf. The application proves to be effective because it has an average accuracy rate of 98%. However, the shape of the leaf is measured at regular intervals where important features and curves may occur between. [7]

Also, there are many proposed systems that extract the leaf vein image to aid

in the classification process, systems like [4] and [8]. Adding the leaf vein data of the image has been recommended in [7] because it is hypothesized to increase the accuracy in classifying a leaf image.

B. Statement of the Problem

Existing plant recognition systems only make use of either the leaf shape or the venation to classify the leaf images into a species of plant. The most promising shape descriptor that has been previously implemented, however, only measures distances from the centroid to the margin of the leaf at regular intervals where curves or other leaf margin features could still occur.

Systems that utilize both the leaf shape and the venation, do not, however, use a neural network classifier to aid in the classification process. Therefore, there is no way of comparing its performance with the existing systems that implement machine learning in the classification process of the leaf.

Also, there have been many proposed models to extract the leaf shape from the leaf image and most of them show promising results. Still, the one implemented by [7], the Centroid-Radii Model, measures the radii at 10°intervals. But given two almost identical leaves, differing only in small details of the shape, the system may detect them as one and the same and would therefore lead to a wrong classification of one or both of these said leaves.

Although the recommended approach to extract the leaf vein image is to utilize the edge detection algorithm and change its parameters, it proved to be inefficient as the noise cannot be avoided. This results to a leaf vein image where the leaf vein is hardly recognizable amidst the noise that is also detected inside the leaf.

C. Objectives of the Study

- **General:**

  To be able to come up with an intelligent system that can classify plants according to its leaf images by extracting its leaf shape and venation using neural networks.

- **Specific:**

    1. User

        - View instructions and guidelines of accepted leaf images.
        - Download reference database of leaves
        - Upload his own leaf images
        - Classify into plants his own leaf images
        - See list of probable plant classifications
        - Upload new image of leaf for classification
        - Enter optional additional information about identified leaf
        - Delete images that he uploaded

    2. Morphology Expert

        - Update reference database of classified leaf images
        - Delete leaf image from reference database
        - Train the neural network for classification
        - Upload trained neural network

D. Significance of the Project

There is a need for a quick and accurate system that can identify plants [2] and hasten the classification process by making a shortcut because further researches consider only plant classification as a first step towards carrying on with the study. If a scientific study was carried out using a wrong plant leaf, it has been proven to be detrimental to the whole research. Also, a quick and accurate system can provide the general users with more information on leaves that only morphology experts can provide themselves. [9]

However, a fast system that is not readily accessible may not really serve its purpose. And a downloadable software answers this problem as it makes the system available for the public and they have access to the software in their own homes. More so, they need not taxonomic training just to be able to identify a plant leaf. [9]

Even though there are already many existing systems that are reliable in plant classification, a hybrid method using the leaf's shape and venation may minimize the margin of error given it is the two most commonly used in plant leaf identification.

E. Scope and Limitations

1. User can view instructions and guidelines.
2. User can upload a leaf image.
3. User can classify a plant based on the leaf image.
4. User can download the collection of identified leaves
5. User has the option to provide additional information about the leaf.
6. User can't make changes on the downloaded reference set of leaf images.
7. There is only one reference set of identified leaf images and each public user has his own set of leaf images.
8. Classification of leaf images is dependent on the reference set of identified leaves.
9. Proposed system only uses the leaf shape and venation to classify a plant leaf image.

F. Assumptions

1. The leaf is not distorted, nor crumpled, nor broken in any way.
2. Leaf is in mature stage (fully grown).
3. The background of the leaf image is white.
4. The leafstalk has been removed from the leaf before taking its picture.
5. There is a group of morphology experts that is both knowledgeable on plant morphology and neural networks to update the database.
6. Images uploaded should only be in JPEG format.

7. Images should be of size 800 x 600.

II. Review of Related Literature

To be able to come up with the backbone of an effective and efficient plant leaf classification system, there are important things to be considered. Some of the concepts to be studied are image processing, pattern recognition, neural networks, and other existing systems and algorithms that have already been done and their performance so as to identify their possible drawbacks that the proposed software needs to address.

Image processing has been long used in different disciplines' advancement. Almost all walks of life are now treading the path towards automation and one key part of this is image processing. But it is emphasized in [10] that "Computer researchers should identify features required for interactive image understanding, rather than their disciplines current emphasis on automatic techniques."

One of the most important steps in image processing is pattern recognition. Pattern recognition consists of extracting, processing and matching features. [11] The extracting part of pattern recognition makes use of the features of the image that could be needed to classify a pattern within the image. The processing part uses the extracted features and the matching part matches the processed features with features that already known to be able to identify the pattern of the image.

However, we need to make sense of these recognized patterns to be able to fully process an image. And for us to do so, we need to classify these patterns and extracted features. One efficient way to do this is by neural networks - which is an attempt to model the functioning of the human brain and neurones and imitates the connections that exist between these neurones. [12]

Many existing systems already serve the purpose of classifying plant leaf images and are done using various methods. Some of these methods include the extraction of the shape using the Centroid-Radii and Moment-Invariant models, use of an MMC classifier, rotary matching of edges, the use of probabilistic neural networks, the use of manifold learning, the use of Fast Fourier Transform, and the extraction of the leaf shape, dent and vein.

Chaki and Parekh proposed an algorithm for classifying a plant by extracting its shape using the Centroid-Radii and the Moments-Invariant model. The image is pre-processed and normalized using Hu's moments. These moment features are invariant to translation, rotation, and scaling. Once the Hu's moments are computed, the centroid is now determined by the average of the x and y pixels. Once the centroid has been determined, the length of the radii are measured using Euclidean distances from the centroid to the boundary points of the leaf at regular intervals. Their system had 97.19% accuracy. [1]

Du et al. used an MMC hypersphere classifier in their system. As with other systems of classifying plant leaves, they preprocessed the image, extracted the features, and used a classifier. Their classifier was the unique aspect of their system because it implemented moving median centers. The MMC's main algorithm is to find an initial center and have that center "hop" from point to point and simulating hyperspheres. Their proposed system turned out to have 91% accuracy. [13]

Gwo, Wei and Li's method was the rotary matching of edges to accomodate leaf edge variation. This is carried out by preprocessing the image, detecting feature points, rotary matching of these points to find the optimal matching composition, and modeling of the leaf using the Viterbi training algorithm to identify the model parameters. It was concluded that their system is highly dependent on the accuracy of the feature points and the feature points of the correct model in the training set. [14]

Wu et al. used PNN (Probabilistic Neural Network) as a classifier for a leaf recognition system[4]. They preprocessed the leaf image, extracted basic geometrical features to be able to extract 12 digital morphological features. PCA is then performed on these morphological features to orthogonalize them and the resulting components are then fed into the PNN to train it. Once the training is finished on the training set, the system was tested and its results were 90% accurate for the respective testing set.

Zhang and Lei's method provides an alternative to PCA (Principal Component Analysis) and LDA (Linear Discriminant Analysis) that only accomodates flat features. The Modified Locally Linear Discriminant Embedding (MLLDE), which is the alternative

solution described, uses manifold learning to capture the nonlinear relationship between data. They also made use of neighborhood graphs and optimal discriminant features. Their results showed that MLLDE had an average recognition rate of .9363. [5]

Lee and Hong's method used a centroid and then performed FFT on the distances from the centroid to be able to extract the features of the leaf shape. They also extracted digital morphological features like [4] using basic geometric features. But their additional step towards extracting features was using the convex hull. They extracted the leaf vein by steps such as morphological opening and projecting their vertical and horizontal histograms. Then they used the peaks of the FFT magnitude in the recognition process. Their system turned out to have an average recognition accuracy of 97.19%. [8]

Wu, Zhou and Wang extracted the features of the leaf shape, the leaf dent (coarseness, size, angle, sharpness of dent), and the leaf vein. Their approach stood out from all the abovementioned because the others just extracted the leaf shape and the contours of the edges or the venation. Leaf dent, apparently, contained rich information to be able to identify a leaf. After testing the neural network that was trained, they had an average recognition rate of 95.56% for 6 kinds of leaves. [15]

III. Theoretical Framework

A. Plant Recognition

Plant Recognition is the process of classifying a plant through its characteristic parts: the leaf, the flower, or the stem, or the fruit, or a combination of some or all. [3] This may be done manually or automatically. Manual methods of plant classification require expert training or sufficient knowledge of plant ecology while automatic methods include image processing.

B. Image Processing

Image Processing is the use of various automatic methods available to be able to extract features from images. The said features are to be then used as data for computer programs. But before the features are used by applications, the pattern of the features has to be first recognized. [10]

C. Pattern Recognition

Pattern Recognition is the first step in image processing and includes identification of known features that can be selected to classify an image's pattern. These features are then represented in a way that can be manipulated automatically by a computer. After the features has been transformed into computer data, noise is removed from the features. The next step in pattern recognition is making sense of the extracted features data and matching them to known patterns that can lead to labeling these features as a member of a certain class. [11]

D. Leaf

The leaf is the part of the plant that is borne out of the plant via stem or stalk. A leaf has many characteristic parts namely: shape, margin, venation, pattern, and arrangement on the leaf.

- Leaf Shape

  A leaf's shape is any form of a shape that the extended part of the leaves can assume; can be ovate (widest near base), obovate (widest near apex), elliptic (widest near middle) or oblong (parallel-sided). [16]

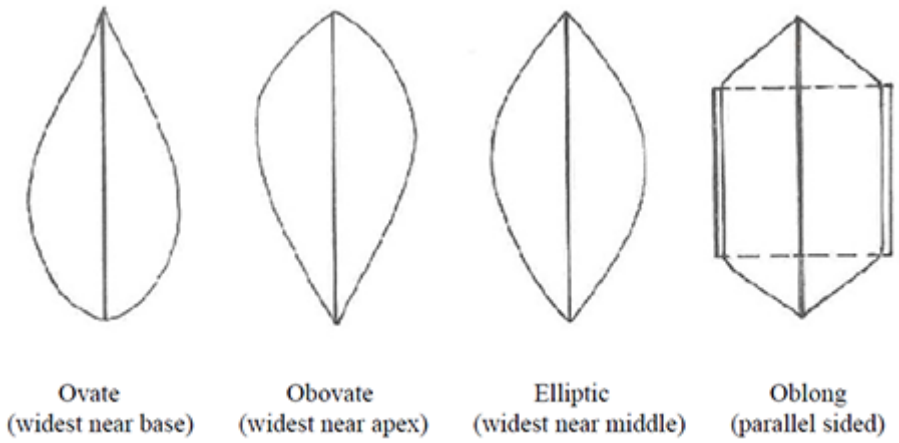


Figure 1: Leaf Shape

- Leaf Margin

  The margin of the leaf is the edge of a leaf; maybe smooth (entire) or toothed (with small notches). [17]

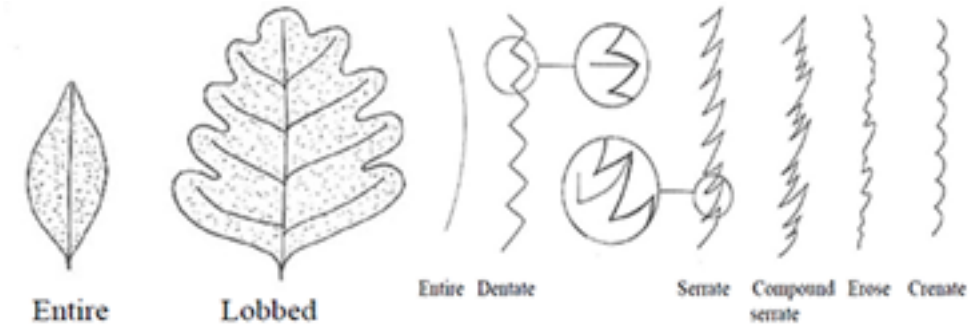


Figure 2: Leaf Margin

- Leaf Venation

  The venation of a leaf is the arrangement of veins on a leaf; can be arcuate (bending towards the apex), cross-venulate (small veins that connect the secondary veins), dichotomous (veins branching symmetrically in pairs), longitudinal (veigns aligned mostly along long axis of leaf), palmate (several primary veins diverging from a

point), parallel (veins arranged axially, not intersecting), pinnate (secondary veins paired oppositely), reticulate (smaller veins forming a network), rotate (in peltate leaves, veins radiating). [18]

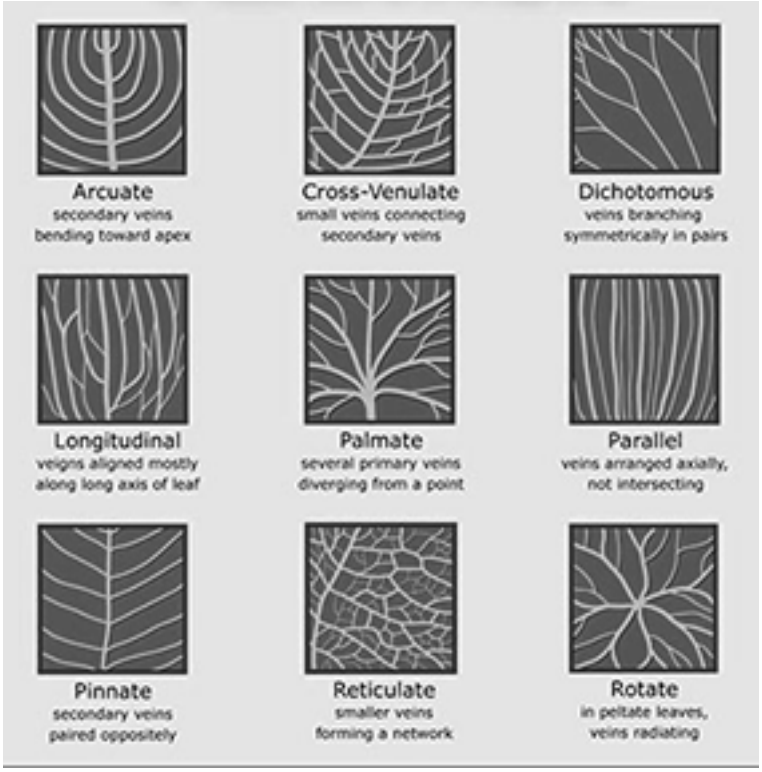


Figure 3: Leaf Venation

- Leaf Pattern

  The pattern of a leaf is the way the leaf buds from the stem or branch; can be simple (undivided growing from a bud) or compound (divided into leaflets and growing from an auxilliary bud). [16]

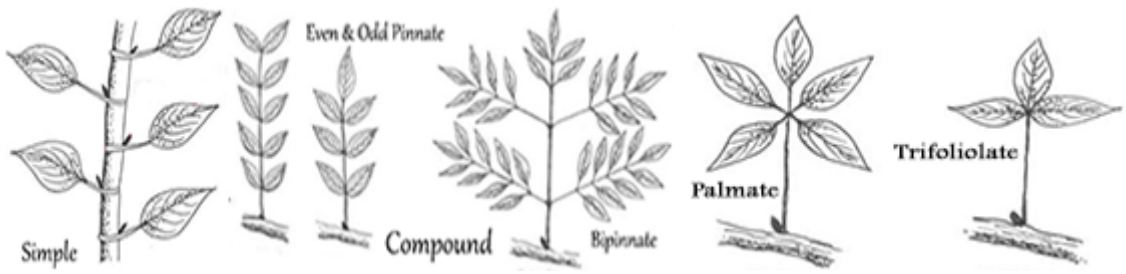


Figure 4: Leaf Pattern

- Leaf Arrangement

  The arrangement of the leaf on the stem or the branch or the stalk can be alternate, opposite or whorled. [16]

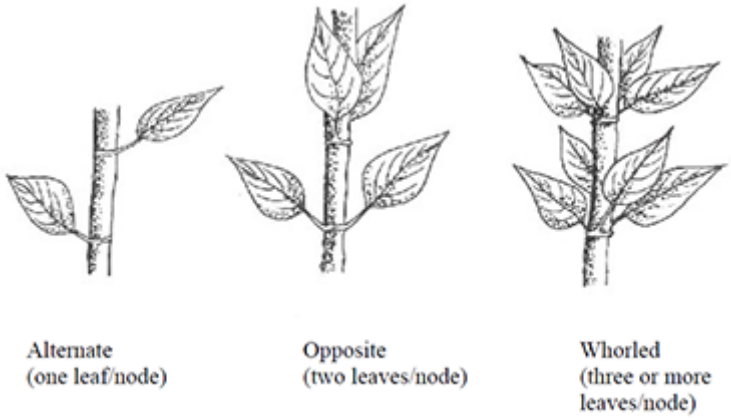


Figure 5: Leaf Arrangement

E. Machine Learning

Machine Learning refers to the changes in systems that perform Artificial Intelligence (AI)-related tasks. [19] It is also the process of providing a computer with prior knowledge and training it to make decisions based on what it already knows.

F. Neural Networks

A neural network is a programming paradigm that aims to imitate the human brain, how it thinks, how it functions, and how it responds. It models the microstructure of the human brain which consists of neurones and connections between them.

The connections that exist between neurones determine whether the next neurone fires (activates). The next neurone is activated through chemicals called *neurotransmitters*. When being activated, the neurotransmitter passes on the input to the neurone and either a message of excitation, inhibition or potentiation.

Since neurones are either activated or deactivated, they can be modeled as switches and the connections between them can be modeled as a matrix of numbers (*weights*), the *weights* can be positive to symbolize the excitation message and negative for the inihibition message. [12]

G. Multilayer Perceptron

The Multilayer Perceptron is a type of feedforward neural network most commonly used for pattern recognition that addresses the limitations of the Perceptron neural network model. The Multilayer Perceptron can be implemented using a series of steps indicated in [12]:

1. Initialization of the network with the weights arbitrarily set to either -1 or 1.
2. Presentation of the training pattern and obtaining the output.
3. Comparison of the output from the network to the target output.
4. Propagation of the error backwards.

   The output layer of weights can be corrected using:

$$w_{ho} = w_{ho} + (\eta \delta_o o_h) \tag{1}$$

   where w_{ho} is the weight from the hidden unit h to the output unit o, η is the learning rate, o_h is the output of the hidden unit h. δ_o is obtained using:

$$\delta_o = o_o(1 - o_o)(t_o - o_o) \tag{2}$$

The input layer can also be corrected using:

$$w_{ih} = w_{ih} + (\eta \delta_h o_i) \tag{3}$$

where w_{ih} is the weight from node i in the input layer to node h of the hidden layer, o_i is the input at node i, η is the learning rate. δ_h can be computed by:

$$\delta_h = o_h(1 - o_h) \sum_o (\delta_o w_{ho}) \tag{4}$$

5. Compute for the average difference between the target vector and the output vector by using the following:

$$E = \frac{\sqrt{\sum_{n=1}^{p} (t_o - o_o)^2}}{p} \tag{5}$$

where p is the number of nodes in the output layer.

6. Repeat above steps from step 2 to every pattern in the training set for the completion of one iteration or *epoch*.

7. Shuffle the training data to avoid the network from developing a bias toward the sequence of data.

8. Repeat from step 2 until error rate become constant or for a finite number of epochs.

H. Centroid-Radii

A Centroid-Radii model captures the shape of an object in an image by finding boundary points and computing their distances from the centroid of the object. The boundary points are defined to be black pixels next to white pixels when black pixels define the area of the object while white pixels define the area of the background. The centroid

(C_x, C_y) is calculated by the average of the x-coordinates and y-coordinates of the black pixels. The lengths of radii is then computed using the Euclidean distance at regular intervals; with a radius being defined as the straight line joining the boundary point and the centroid. If θ is the regular interval (measured in degrees) between the radii, the number of intervals can be stated as $k = 360/\theta$. When the intervals are taken clockwise from the x-axis direction, the descriptor of the shape could be the vector S defined by: [1]

$$S = \{r_0, r_\theta, r_{2\theta}, ..., r_{(k-1)\theta}\} \tag{6}$$

where $r_{i\theta}, 0 \leq i \leq (k-1)$ is the $(i+1)$-th radius from the centroid to the respective boundary point.

I. Moment-Invariants

M-K Hu's moment features are used to describes shapes that are invariant to rotation, translation and scaling. A moment of a pixel $P(x, y)$ at location (x, y) can be computed by taking the product of the pixel value and its coordinate distances i.e. $m = x \cdot y \cdot P(x, y)$. The moment of the entire image is defined as the summation of all the moments of its pixels. In general, the moment of order (p, q) of an image $I(x, y)$ is: [1]

$$m_{pq} = \sum_x \sum_y [x^p y^q I(x, y)] \tag{7}$$

Based on the p and q values, the following can be defined:

$$m_{00} = \sum_x \sum_y [x^0 y^0 \cdot I(x, y)] = \sum_x \sum_y [I(x, y)] \tag{8}$$

$$m_{10} = \sum_x \sum_y [x^1 y^0 \cdot I(x, y)] = \sum_x \sum_y [x \cdot I(x, y)] \tag{9}$$

$$m_{01} = \sum_x \sum_y [x^0 y^1 \cdot I(x,y)] = \sum_x \sum_y [y \cdot I(x,y)] \tag{10}$$

$$m_{11} = \sum_x \sum_y [x^1 y^1 \cdot I(x,y)] = \sum_x \sum_y [xy \cdot I(x,y)] \tag{11}$$

$$m_{20} = \sum_x \sum_y [x^2 y^0 \cdot I(x,y)] = \sum_x \sum_y [x^2 \cdot I(x,y)] \tag{12}$$

$$m_{02} = \sum_x \sum_y [x^0 y^2 \cdot I(x,y)] = \sum_x \sum_y [y^2 \cdot I(x,y)] \tag{13}$$

$$m_{21} = \sum_x \sum_y [x^2 y^1 \cdot I(x,y)] = \sum_x \sum_y [x^2 y \cdot I(x,y)] \tag{14}$$

$$m_{12} = \sum_x \sum_y [x^1 y^2 \cdot I(x,y)] = \sum_x \sum_y [xy^2 \cdot I(x,y)] \tag{15}$$

$$m_{30} = \sum_x \sum_y [x^3 y^0 \cdot I(x,y)] = \sum_x \sum_y [x^3 \cdot I(x,y)] \tag{16}$$

$$m_{03} = \sum_x \sum_y [x^0 y^3 \cdot I(x,y)] = \sum_x \sum_y [y^3 \cdot I(x,y)] \tag{17}$$

The first four Hu invariant moments are invariant to rotation and are given by:

$$\varphi_1 = m_{20} + m_{02} \tag{18}$$

$$\varphi_2 = (m_{20} - m_{02})^2 + (2m_{11})^2 \tag{19}$$

$$\varphi_3 = (m_{30} - 3m_{12})^2 + (3m_{21} - m_{03})^2 \tag{20}$$

$$\varphi_4 = (m_{30} + m_{12})^2 + (m_{21} + m_{03})^2 \tag{21}$$

The image is then shifted so that its centroid coincides with the origin of the coordinate system. This is done to make the moments invariant to translation. The centroid of the image is obtained by (in terms of moments):

$$x_c = \frac{m_{10}}{m_{00}} \tag{22}$$

$$y_c = \frac{m_{01}}{m_{00}} \tag{23}$$

The central moments can now be defined as:

$$\mu_{pq} = \sum_x \sum_y [(x - x_c)^p (y - y_c)^q I(x, y)] \tag{24}$$

To compute for the Hu moments using central moments, the φ terms above need to be replaced by μ terms. This results to $\mu_{00} = m_{00}, \mu_{10} = 0 = \mu_{01}$.

Next, we divide the moments by a power of μ_{00} to normalize them and make them invariant to scaling. The *normalized central moments* are:

$$M_{pq} = \frac{\mu_{pq}}{(\mu_{00})^\omega} \tag{25}$$

where

$$\omega = 1 + \tfrac{p+q}{2}$$

J. Canny Edge Detection

Canny Edge Detection is an algorithm for edge detection developed by John F. Canny in 1986 that is optimal in detection, localization, and number of responses. The algorithm has 5 steps and they are: [20]

1. Smoothing

   The blurring of the image to remove noise that may be mistaken for edges is done by applying a Gaussian filter. The kernel of a Gaussian filter with a standard deviation of $\sigma = 1.4$ is:

$$B = \frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

(a) Original (b) Smoothed

Figure 6: Example of Smoothing

2. Finding gradients

   The algorithm determines the gradients of the image to identify the edges where the grayscale intensity changes the most. By applying the *Sobel-operator*, the gradients at each pixel can be computed. The first step is to approximate the gradients in the x- and y- directions by the kernels:

$$K_{GX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$K_{GY} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The *edge strengths* (gradient magnitudes) are determined as an Euclidean distance measure resulting to:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{26}$$

$$|G| = |G_x| + |G_y| \tag{27}$$

where G_x and G_y are the gradients in the x- and y-directions.

Some edges are typically broad and fail to indicate the exact edge. To address this, the direction of the edges are first determined then stored as such:

$$\theta = arctan\left(\frac{|G_y|}{|G_y|}\right) \tag{28}$$

(a) Smoothed (b) Gradient magnitudes

Figure 7: Example of finding gradients

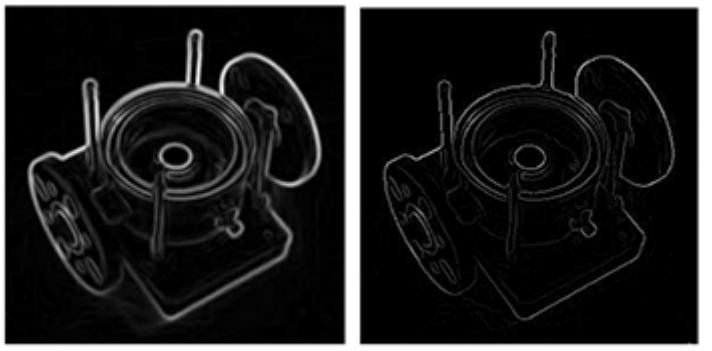
3. Non-maximum suppression

   This step converts the "blurred" edges to "sharp" edges by preserving all local maxima and deleting everything else. The algorithm for each pixel in the gradient image is:

   (a) Round the gradient direction θ to nearest 45°, which leads to the use of an 8-connected neighbourhood.

   (b) Compare the edge strength of the current pixel to the edge strength of the pixel in the positive and negative gradient direction.

   (c) If the edge strength of the current pixel is the largest, preserve its value; if not, remove it.



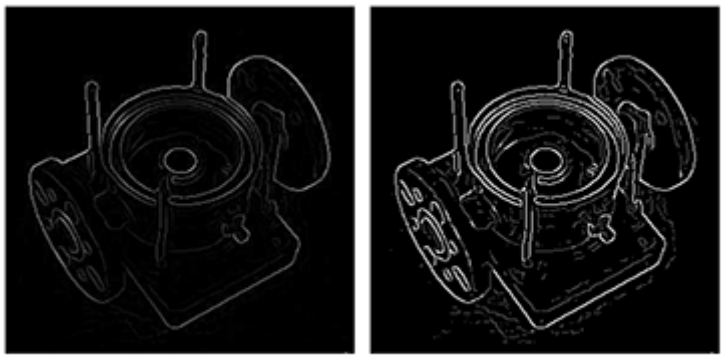
(a) Gradient values (b) Edges after non-maximum suppression

Figure 8: Example of non-maximum suppression

4. Double Thresholding

   Even after non-maximum suppression, the remaining edges may be the true edges but some of them may be caused by noise or color variations. Using a threshold differentiates one from the other by preserving only those edges stronger than a certain value. Canny Edge uses double thresholding; edge-pixels stronger than the high threshold are *strong* while those below the low threshold are removed, and those between the high threshold and the low threshold are *weak*. An example with thresholds of 20 and 80 is shown:

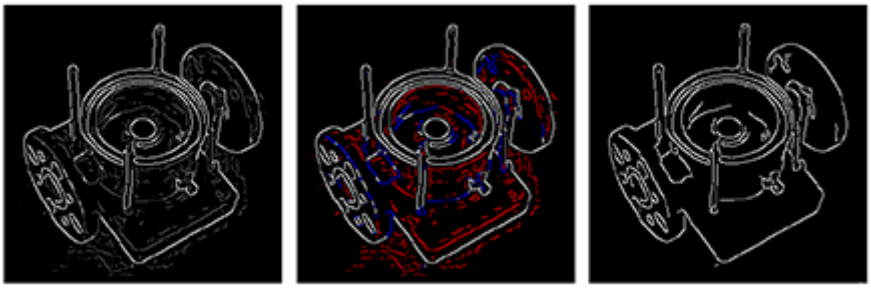
5. Edge tracking by hysteresis



(a) Edges after non-maximum suppression (b) Double thresholding

Figure 9: Example of Double Thresholding

Strong edges are now included in the final edge image and weak edges are included if and only if they are connected to strong edges.



(a) Double thresholding (b) Edge tracking by hysteresis (c) Final output

Figure 10: Example of edge tracking by hysteresis

K. Morphological Opening

Morphological operations are applied to images, most specifically, their pixel values. These operations use a small shape to transform the image called the structuring element. One morphological operation that is commonly used is the Opening operation. An opening of an image consists of eroding the image and consequently dilating it. [21]

L. Image Difference

The Image Difference Algorithm is part of the flatjavasrc package of the HIPR Project. Its authors are Judy Robertson and Timothy Sharman. The algorithm is used to find

the difference between two images. The input images should be in grayscale and should be transformed to their respective array of pixels. The seven steps of the algorithm are:

1. Assume the image is grey level (hence RR=GG=BB).
2. Use value 0x000000ff to get the BB value.
3. Do this for both input images.
4. Apply the operation (eg subtract).
5. Add 128 and then scale.
6. Clip to lie from 0 to 255. Call this value 0xCD.
7. Create int value 0xffCDCDCD.

IV. Design and Implementation

A. Use-Case Diagram

The system has two users: the public user and the morphology expert.

The user can download the updated reference database of leaves and network file or upload a leaf image for classification. Once the user has uploaded a leaf image, he can classify the leaf image using the neural network. The list of probable classifications will be shown to the user and he has the option whether to enter additional information about the leaf and reduce the list of results based on the properties he has supplied.

The morphology expert can either: upload a new classified leaf image, or delete a leaf image from the reference database. Once he uploads a new classified leaf image, the neural network is trained and then uploaded.

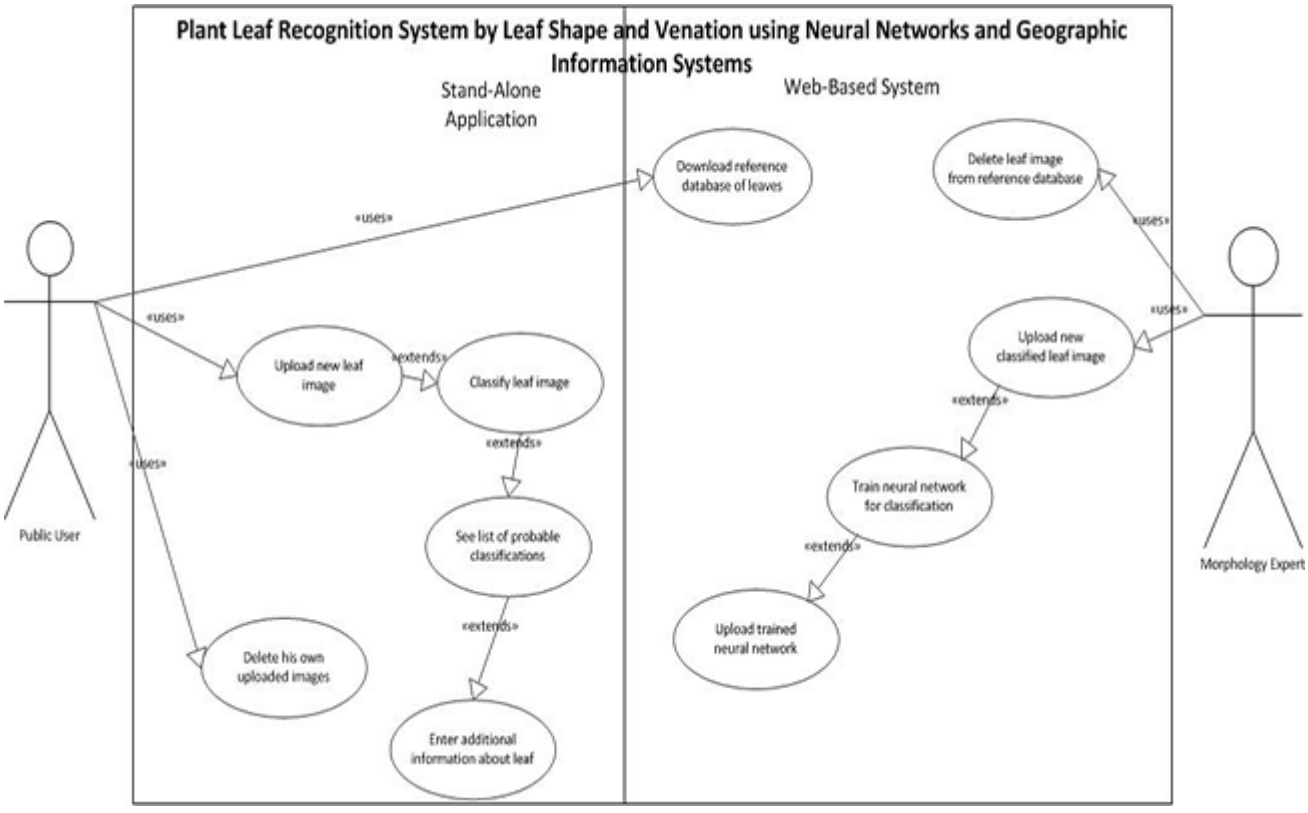


Figure 11: Use-Case Diagram

B. Context Diagram

The system has two users that are interacting with the sytem: the public user and the morphology expert. The public user has the main functions of downloading and uploading. He can download the reference database of leaves and network file and upload his own leaf images for classification. The morphology expert has the main functions of uploading leaf images that will constitute the reference database of leaves and uploading the trained network.

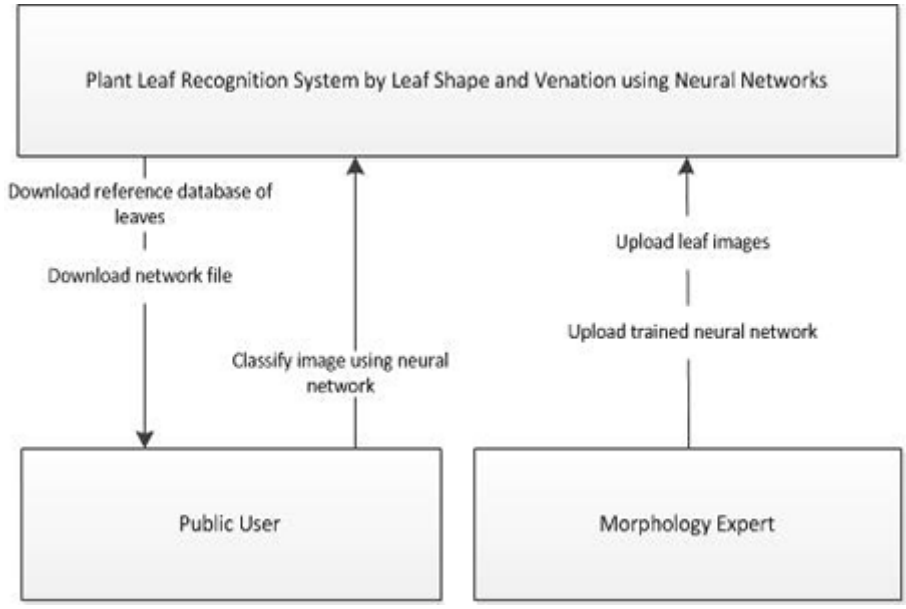


Figure 12: Context Diagram

C. Flowchart

The user first selects whether to upload a leaf image for classification or download the reference database of leaves and updated network file. When the user decides on the former, he can now classify the leaf image. The image will undergo normalization to be able to extract the first set of features to be used as input for the network. Afterwards, the edge will be detected and the shape descriptor will be utilized for the next set of features. Lastly, the vein extraction will be performed to be able to complete the features needed for the network to classify the leaf image. Once the network has classified the leaf image as to what species it most likely is, the system outputs the list of plant species and

their corresponding percentage of match to the leaf image. The user then has the option to input additional information about the leaf. If he supplies a leaf property, the list of results will be reduced accordingly.

When the expert uses the system, he can upload a new species or a new leaf image of an existing species in the database. He can train the neural network and input optional parameters to get the best results. Once the network is stable, he can save the network and upload it afterwards. He can also delete leaf images that were previously uploaded.

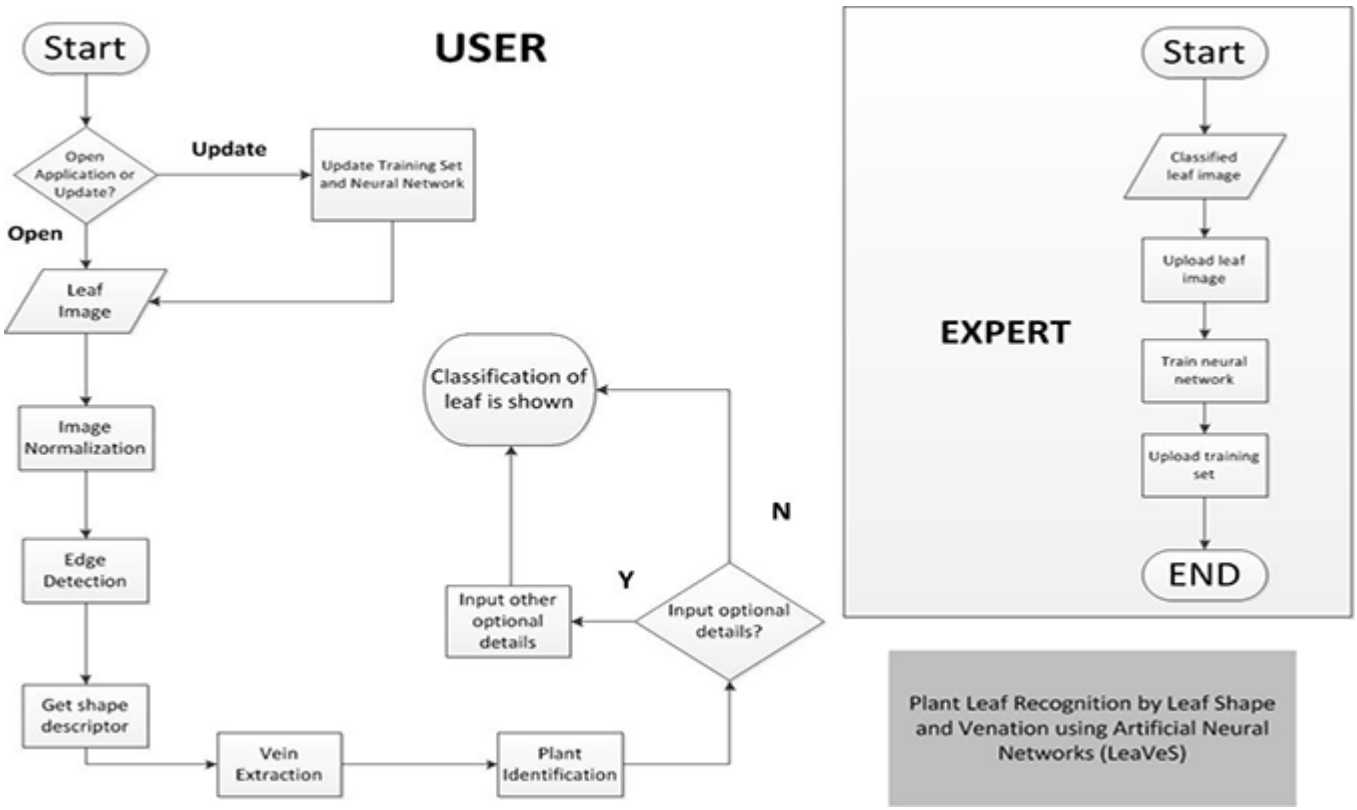


Figure 13: Flowchart

D. Neural Network Design

The neural network used is the Multilayer Perceptron model with back propagation. As is with usual MLPs, the neural network was designed to have an input layer, a hidden layer, and an output layer. The hidden layer consisted of only one layer of nodes. The input layer has 189 nodes which represented the data extracted from the leaf image. 4 nodes are the central moments of the leaf image, these moments are invariant

to translation, rotation and scaling extracted using Hu's moment features from [1]. The next 180 nodes are the radii of the leaf edge image from the centroid, extracted using the centroid-radii model from [1]. The last 5 nodes are from the vein extraction process described in [4].

The hidden layer consists of 147 nodes. This is a good estimate since the recommended number of nodes in the hidden layer of an MLP should be 2/3 of the number of input nodes and number of output nodes. While the output layer has 32 nodes which represent each species currently stored in the database.

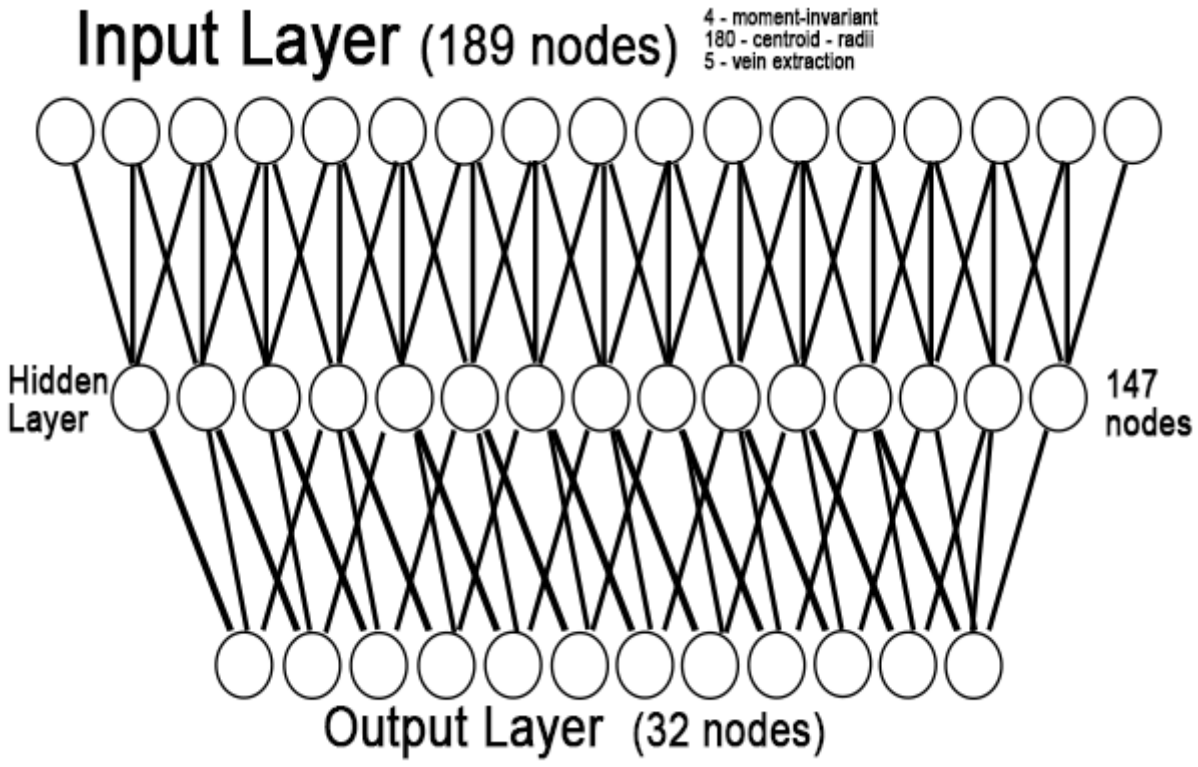


Figure 14: Neural Network Design

E. Neural Network Implementation

1. Data Preprocessing

   The only accepted data by the system is a .jpg file that is a leaf image. Once the user uploads the said image, preprocessing is done on the image to be able to extract data from it. The different preprocessing steps done to the image is:

(a) Moments Extraction

Following the model of Hu's moment features on [1], only the 4 central moments are to be used as the data for the network. These moments are extracted from the original leaf image, and these are invariant to translation, rotation and scaling.

(b) Canny Edge Detection

The edge detection algorithm used is the same with [**?**]. The extraction of data, however, was improved to further model the details of the edge of the leaf image. It was modified to measure radii at 2°intervals instead of 10°to be able to include the important features between 10 °.

(c) Veins Extraction

In order to extract the veins of the leaf image, several steps must first be undertaken. The leaf image should first be converted to grayscale. Then the image would undergo morphological opening with a flat, circular structuring element with varying radii to improve the visualization of the veins. This would then undergo an algorithm named **Image Difference** with the other image being the grayscale image of the leaf. The resulting image resembles the leaf vein image.

2. Data Input

The input layer consists of the 4 central moments, 180 radii from the leaf edge image, and 5 vein features from the veins extraction algorithm. These are then fed to the neural network either for training or for testing.

3. Network Training

Once the data is separated to follow the 70-30 rule of training and testing, and the 70% of the leaf images are already uploaded to the database together with its respective data, the training can now commence. To be able to get the best results from the network, the learning rate, max error, and hidden nodes are chosen carefully. A good estimate for the learning rate would be in the range of 0.2 to 0.8

but a lower learning rate, according to [22] avoids oscillations in the training but converges slower compared to training the network with a higher learning rate.

The max error is chosen so as the smaller the max error, the more accurate the network turns out to be. And although the recommended number of hidden nodes is 2/3 of the input nodes and the output nodes, it could be any integer in between the input and output nodes.

4. Network Testing

The data from the user's uploaded leaf image is extracted in the same way as in training, but the testing requires that a neural network file be downloaded first into the user's own computer. This will be utilized to classify the leaf image by showing the user with its' percentage match to each species in the database also downloaded.

V. Architecture

A. System Architecture

The application - both for the expert user and for the public user - was written in Java. For the main functions of detecting the edge and the veins image, the classes implemented are also in Java. Consequently, the extraction of features of both descriptors are in Java as well. The MySQL database queries were conveniently written in Java as well. The application connects to the MySQL database via Java Database Connectivity (JDBC). For the design of the neural network, the Multilayer Perceptron of neuroph was utilized.

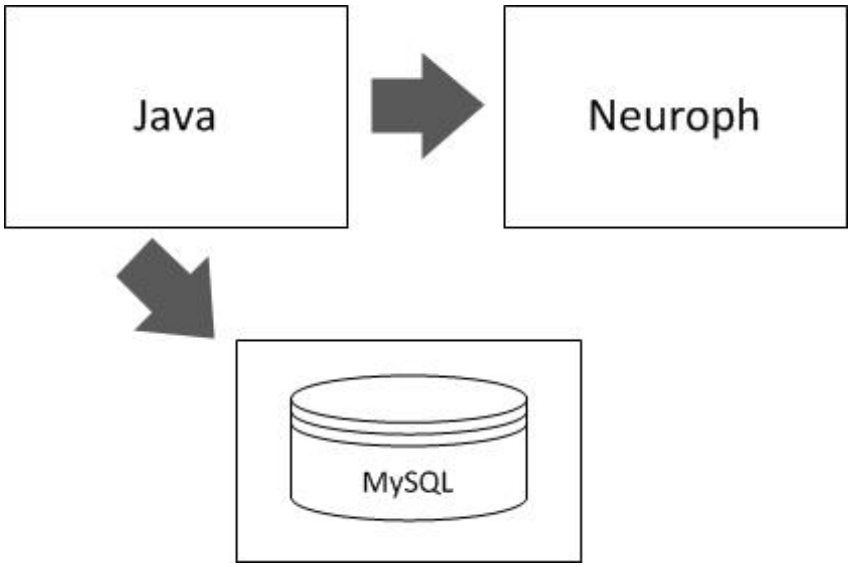


Figure 15: LeaVeS System Architecture

B. Technical Architecture

The system is downloadable and used on a computer. The storage for the collection of leaf images is the user's computer memory. The reference database of leaves and the neural network is downloadable via a reliable Internet connection.

Minimum System Requirements

1. 1 GB RAM
2. 1.60GHz processor
3. 32-bit operating system
4. Microsoft Windows 2000/XP/Vista/7, Linux
5. JRE (Java Runtime Environment)
6. Reliable internet connection

VI. Results

A. Home Screen

At once, the splash screen will display as seen in 16 and will open the home screen afterwards. The home screen of the LeaVeS application is shown in Figure 17. Upon opening of the application, the user is prompted to enter login details if the user is a morphology expert or to freely download the updated reference database of leaves and network file or to upload a leaf image for classification.

Figure 16: LeaVeS' Splash Screen

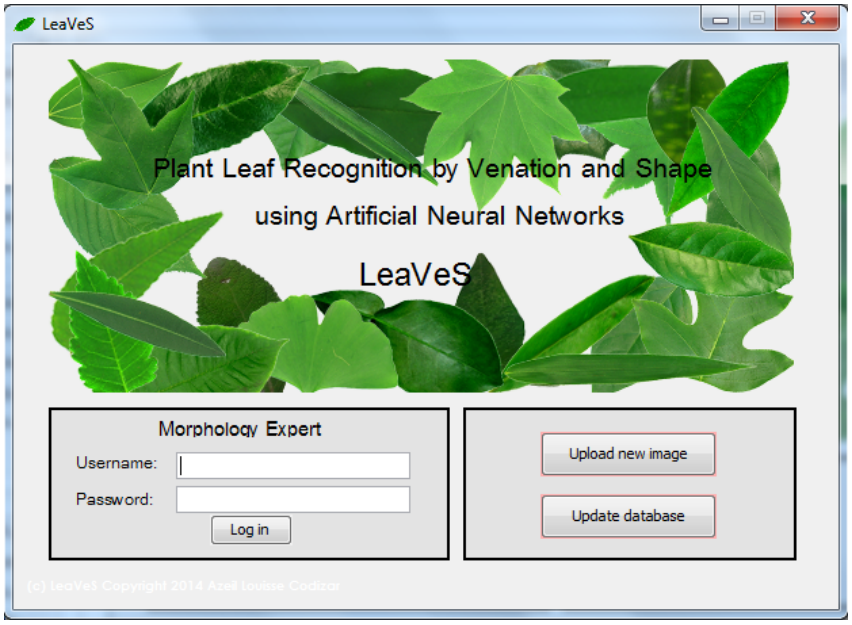


Figure 17: LeaVeS Home Screen

B. Public User

1. **Update reference database of leaves and network file**

   The user can update the reference database of leaves by pressing the *Update database* button on the home screen following Figure 18. The system will prompt the user of the update as seen in 19 If the download was successful, the system will prompt the user as can be seen in Figure 20.

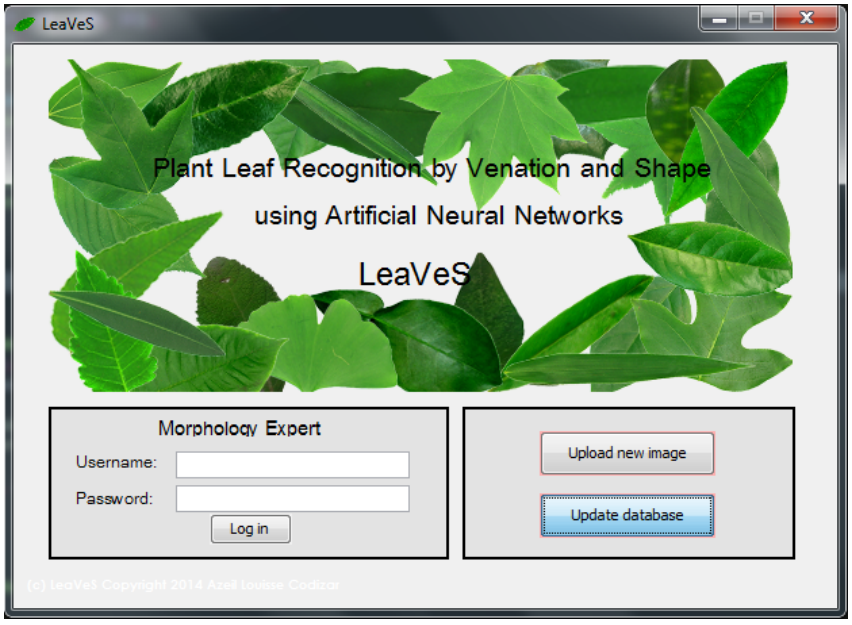


Figure 18: Update database

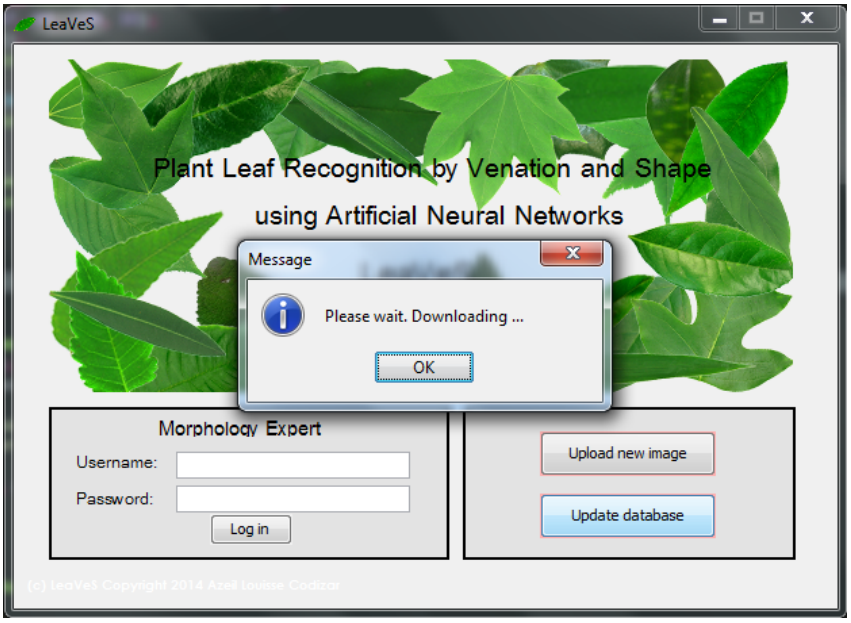


Figure 19: Downloading prompt

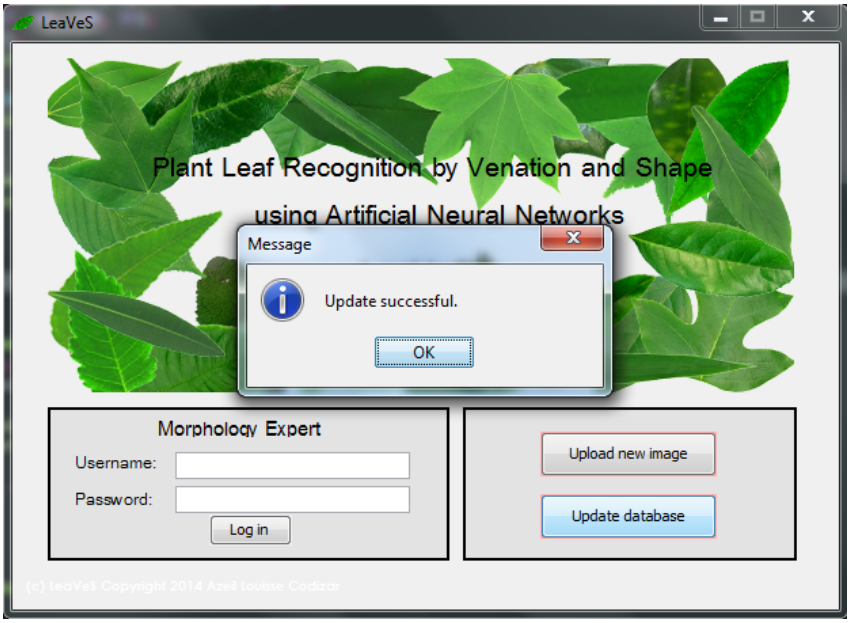


Figure 20: Update successful

2. **Uploading an image for classification**

The user can now upload a new leaf image for classification as seen in 21. The leaf image has to be in his own collection of leaf images. The instructions and guidelines on uploading a leaf image will be prompted as seen in Figure 22. The window for choosing a leaf image will be shown as can be seen in Figure 23.

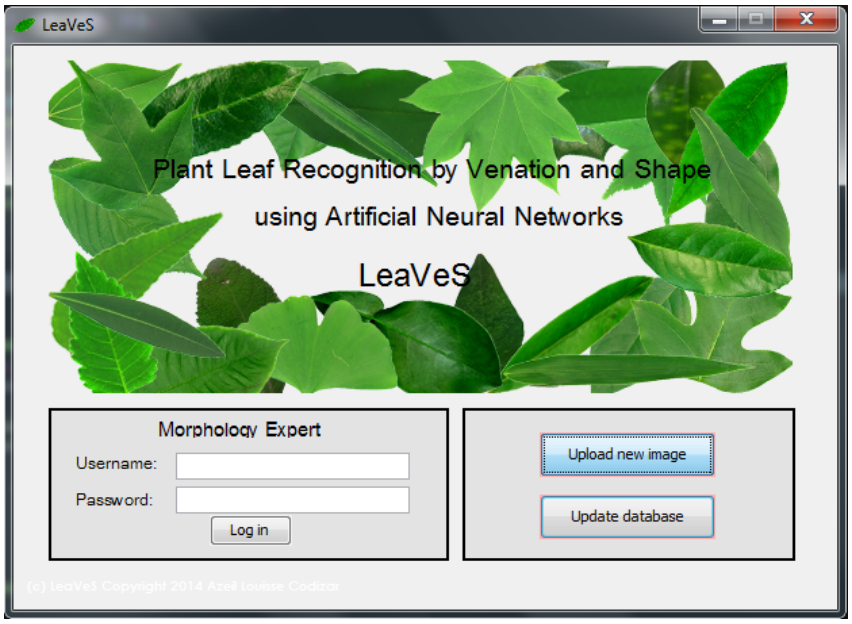


Figure 21: Upload leaf image for classification

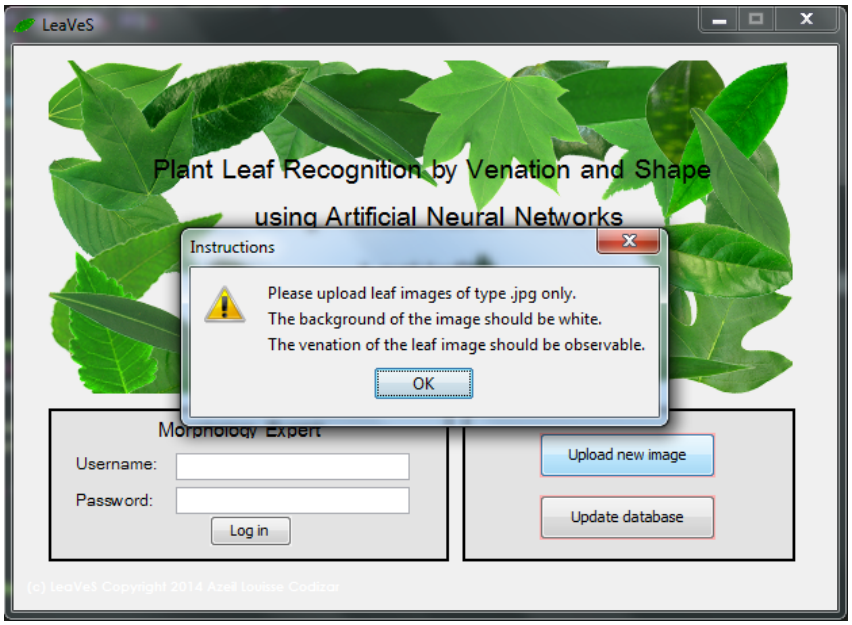


Figure 22: View instructions and guidelines

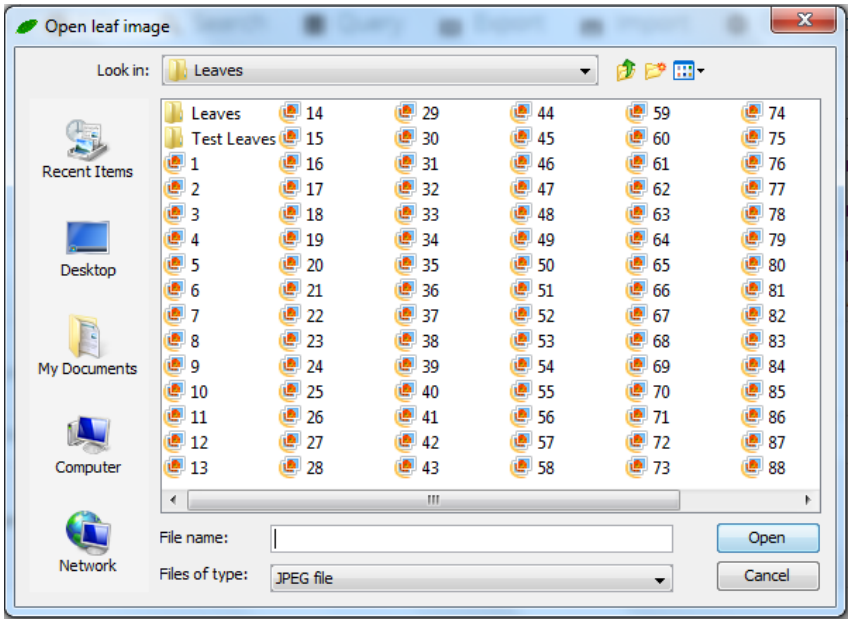


Figure 23: Select a leaf image

Once the user has selected a leaf image, the Public User screen will appear as seen in Figure 24. The user now has three options: to detect and view leaf edge image, to detect and view leaf vein image, or to classify the leaf image.

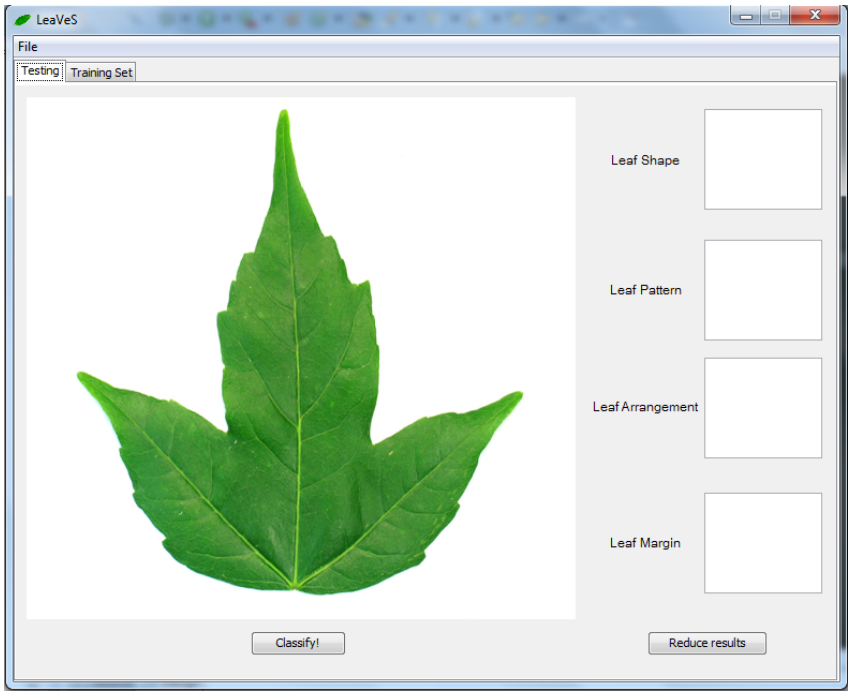


Figure 24: Public User Screen

If he chooses the first option, he can select the *View edges image* option in his menu bar, following Figure 25. The leaf edge image will then appear as seen in Figure 26.

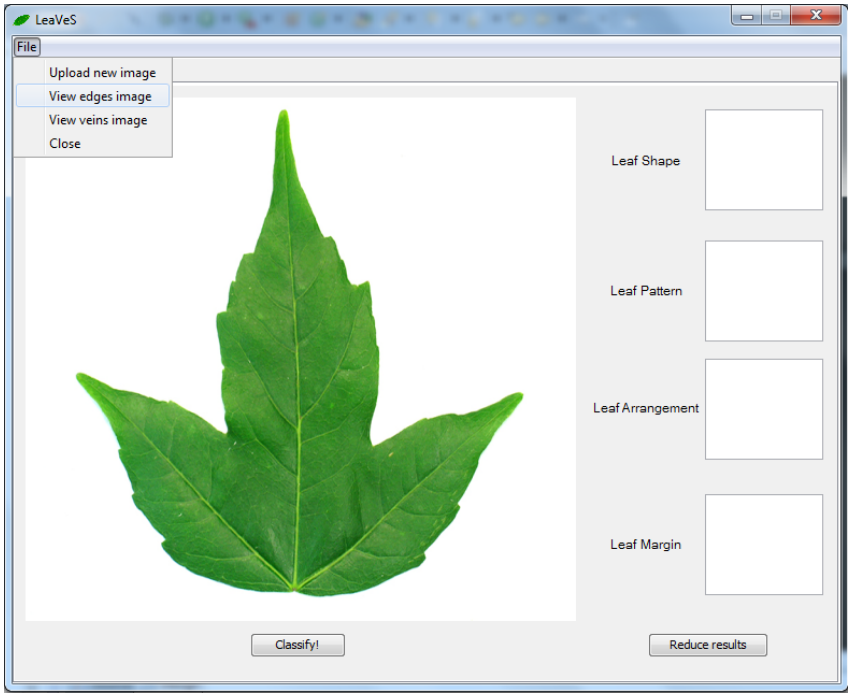


Figure 25: View leaf edge image

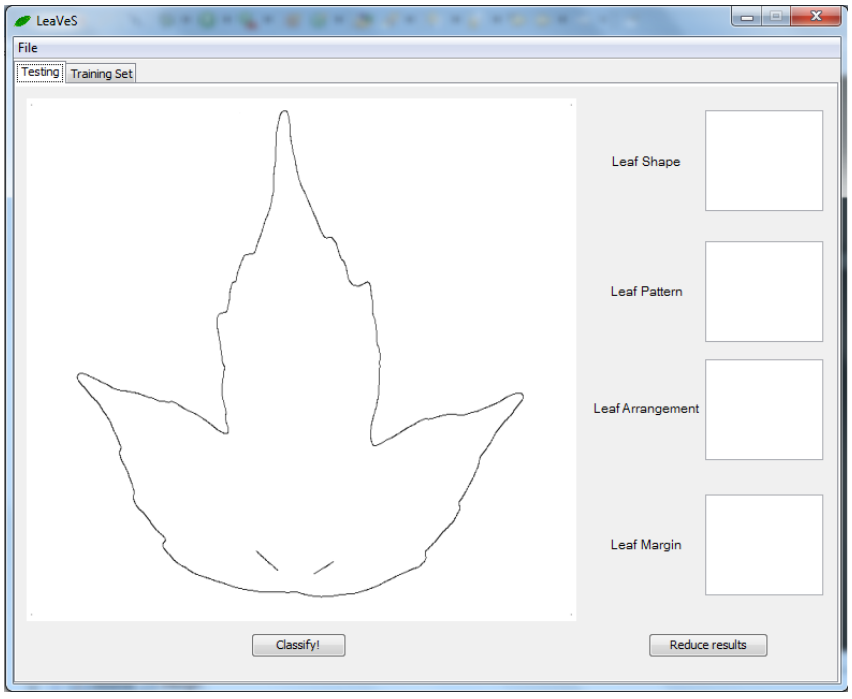


Figure 26: Leaf edge image

But if he decides on the second option, he can select the *View veins image* option in his menu bar as can be seen in Figure 27. Consequently, the leaf veins image will appear like the one seen in Figure 28.

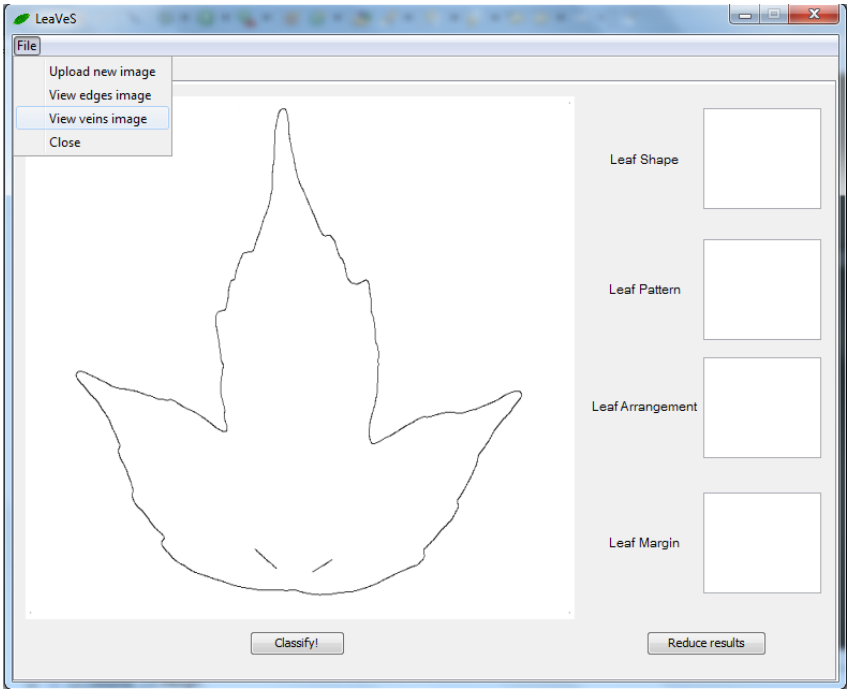


Figure 27: View leaf veins image

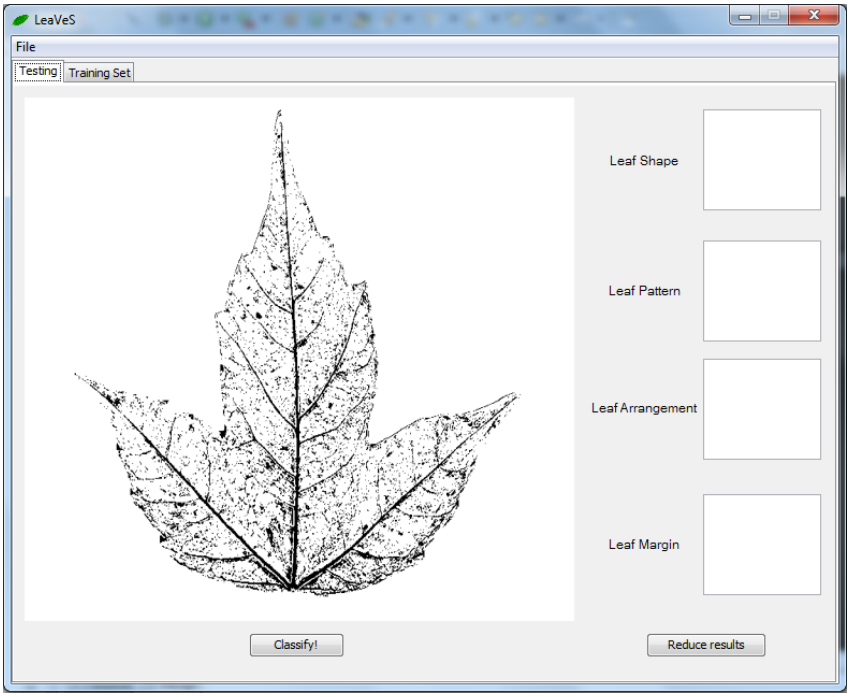


Figure 28: Leaf veins image

3. **Classify a leaf image**

Or the user can opt to classify the leaf image right away by pressing the *Classify!* button below the leaf image as seen in Figure 29. Once the system has finished extracting features, utilizing the network, and identifying which plant species it most likely is, the system prompts the user that it has classified the leaf image as can be seen in Figure 30.

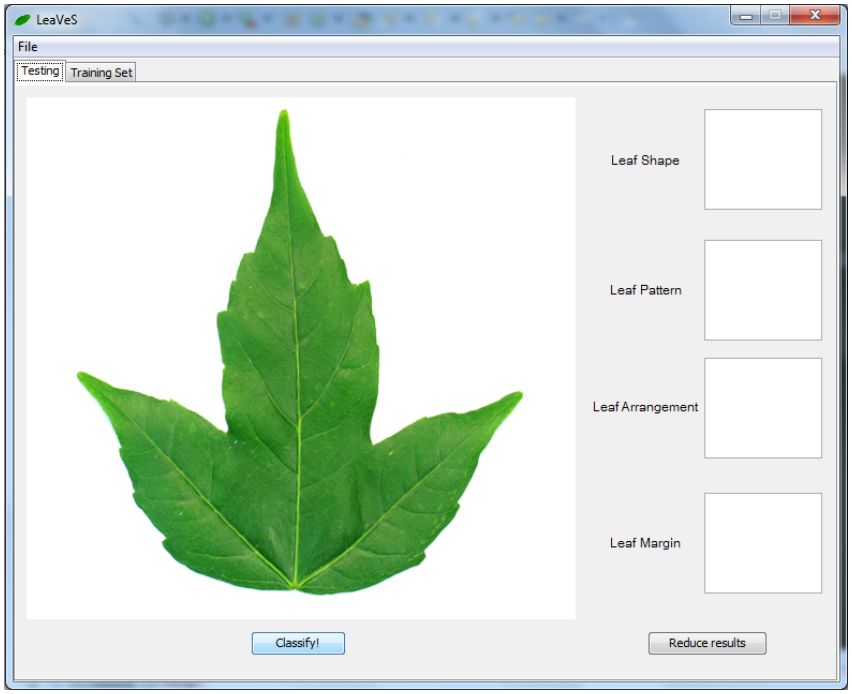


Figure 29: Classify leaf image

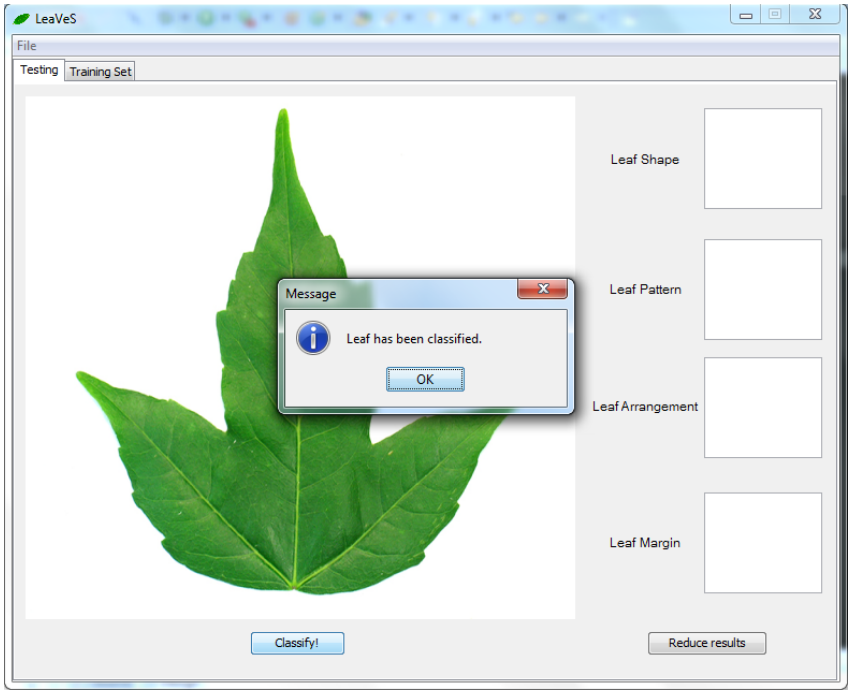


Figure 30: Leaf image classified

4. **See list of probable classifications** The user will now be seeing a list of the probable matches with their corresponding percentage of match as seen in Figure 31. All the species uploaded in the database will be in the list and will be arranged in decreasing order of percentage match to the leaf image.

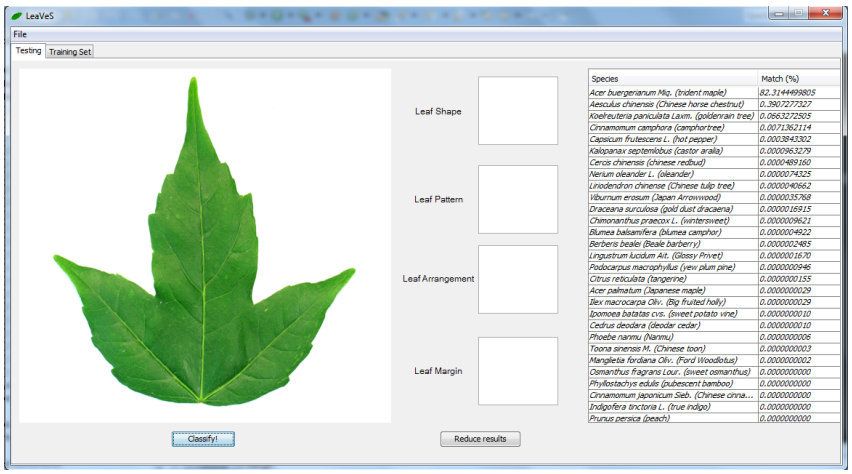


Figure 31: List of probable classifications

5. **Enter optional additional information about leaf**

Once the list of probable classifications have been browsed, the user can decide to enter optional additional information that he knows about the leaf to refine the list and output only the species that have the property or properties that he supplied. This can be reflected in Figures 32 and 33.

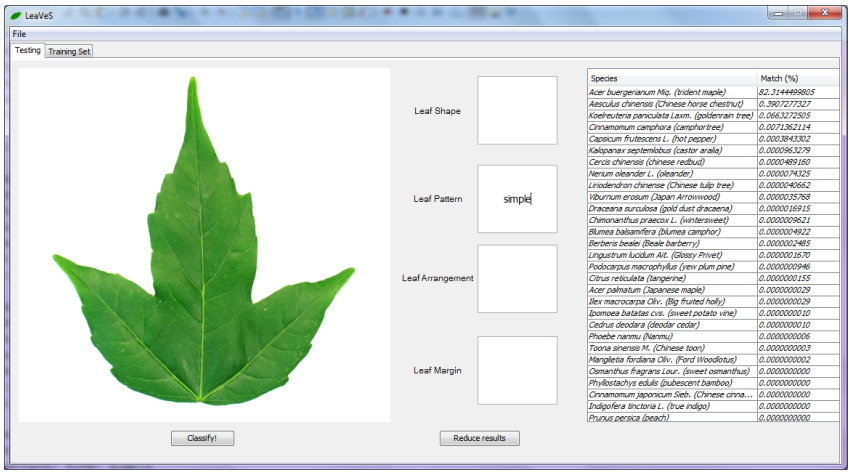


Figure 32: Enter optional additional information about leaf

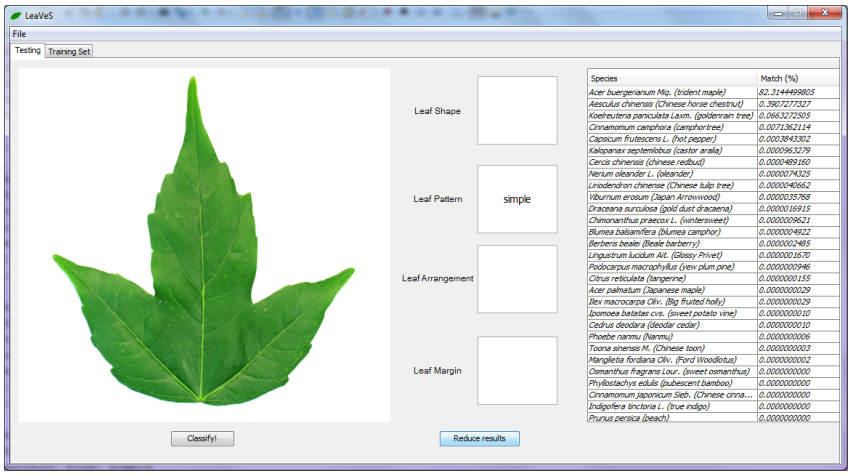


Figure 33: Enter optional additional information about leaf

The list will now be populated only with the species that match the property entered by the user. An example of this is seen in Figure 34.

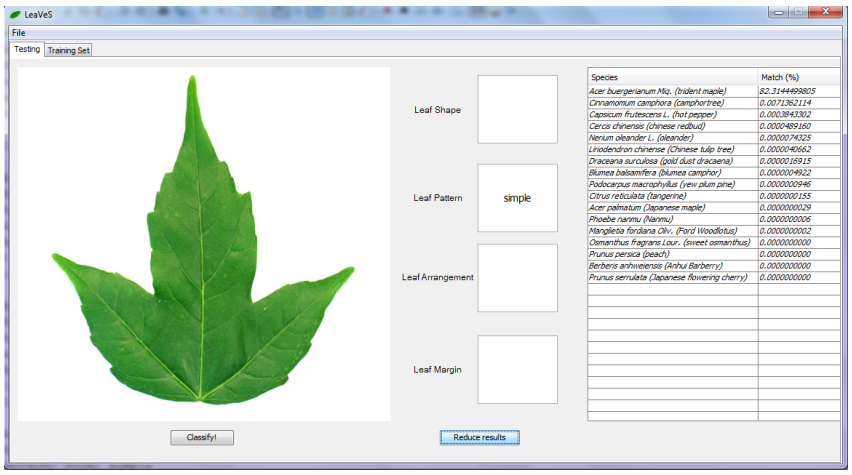


Figure 34: Reduced list of plant species

6. **View training set**

The user can also view the list of leaf images uploaded by the expert but cannot perform any changes on the said list. The *Training Set* tab being described is seen in Figure 35.

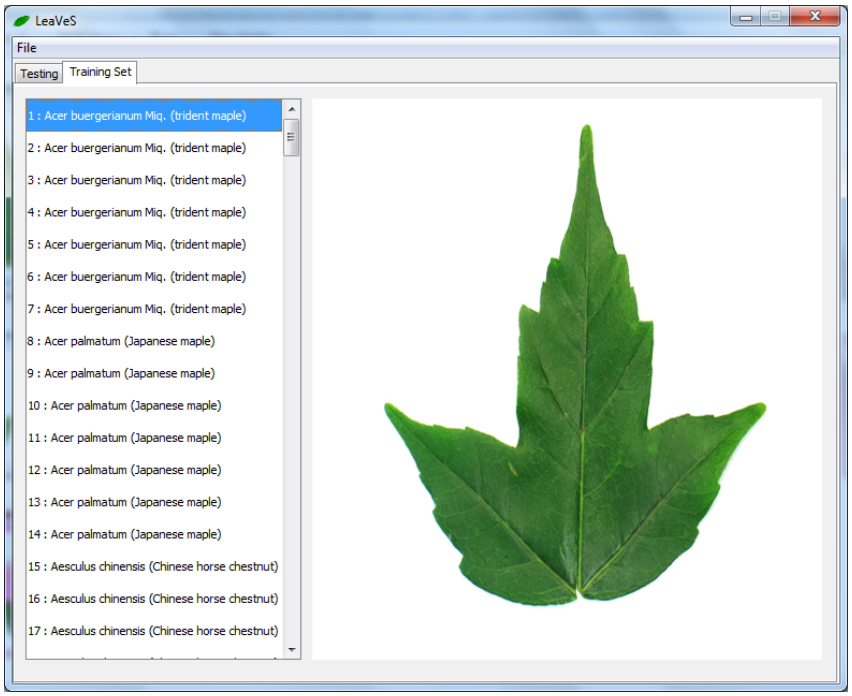


Figure 35: View database of leaves

7. **Delete uploaded leaf image**

If the user wants to delete his uploaded leaf image, he can upload a new leaf image in place of the previous leaf image. This can be done by selecting the *Upload new leaf image* option in the menu bar as seen in Figure 36.

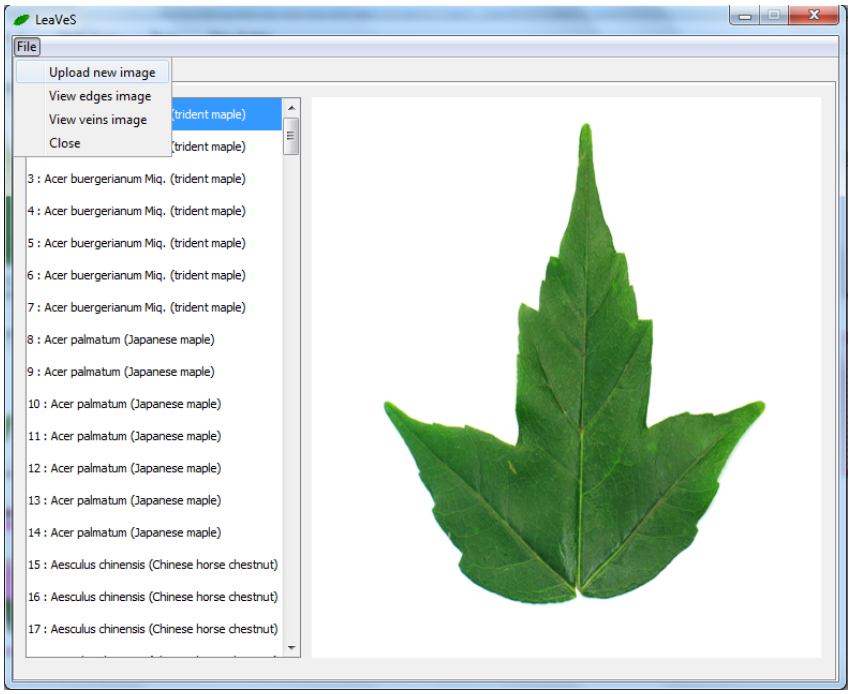


Figure 36: Deleting a leaf image

8. **Upload a new leaf image for classification**

The instructions and guidelines will be shown again and this is reflected in Figure 37. This will then lead to the window where the user can select which image to be uploaded as seen in Figure 38. And as the previous selection of leaf image, it will proceed to the user's screen where he has again, the three options.

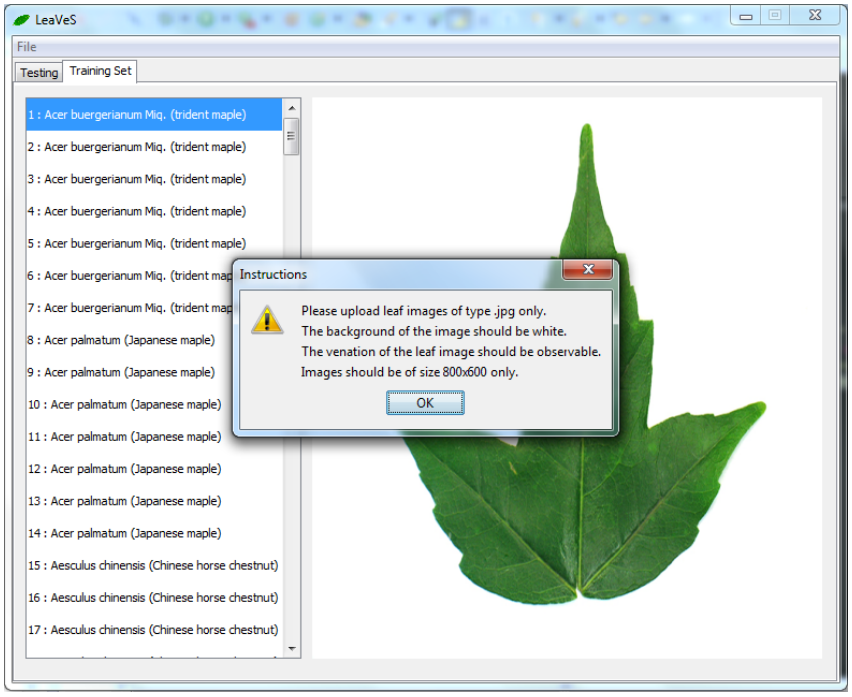


Figure 37: Instructions and guidelines

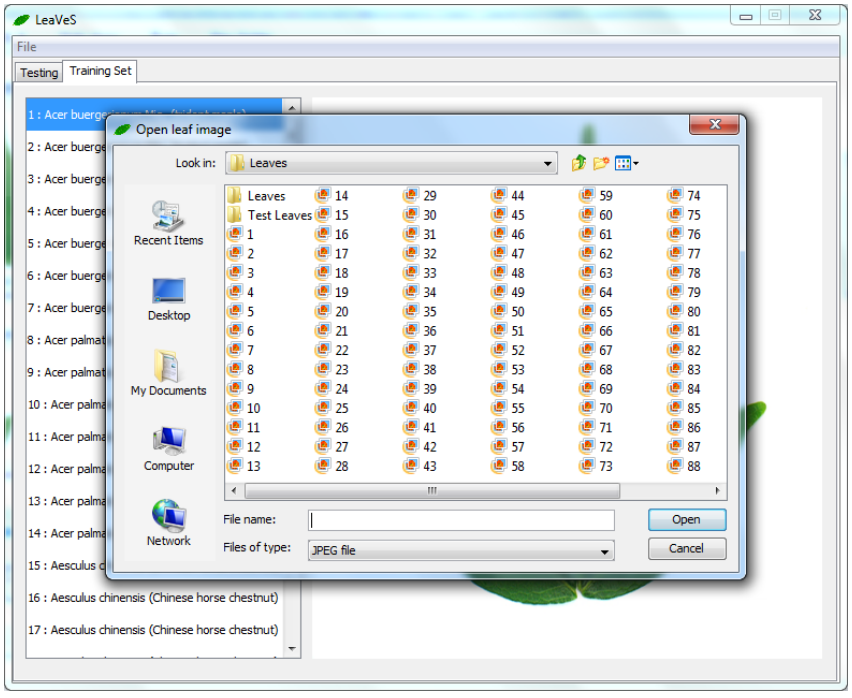


Figure 38: Upload new leaf image

C. Expert User

1. Expert Menu

Once the login details have been supplied, the morphology expert is taken to the expert screen. He is given a menu bar as seen in Figure 39 to be able to adjust the

parameters of edge detection and vein detection that will be most appropriate for the current leaf image and to save and upload the updated network.

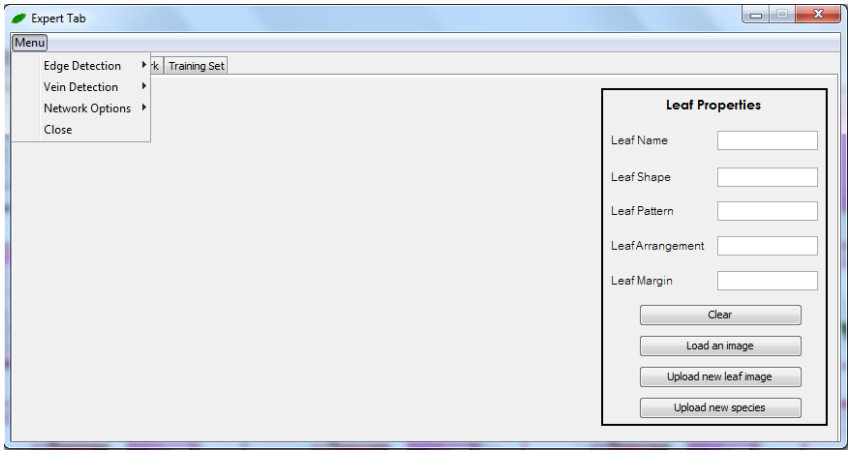


Figure 39: Expert Menu

2. **Load a leaf image**

Figure 40 shows the *Leaf Species* tab for the expert user. Here, the expert can choose whether to load a leaf image, supply the leaf properties he has identified, and upload it as a new species or to upload it as a leaf image of an existing species in the database. Note that the leaf properties need not be complete for the leaf image to be uploaded as a new species. This can be done by pressing the *Load an image* button as seen in 41.

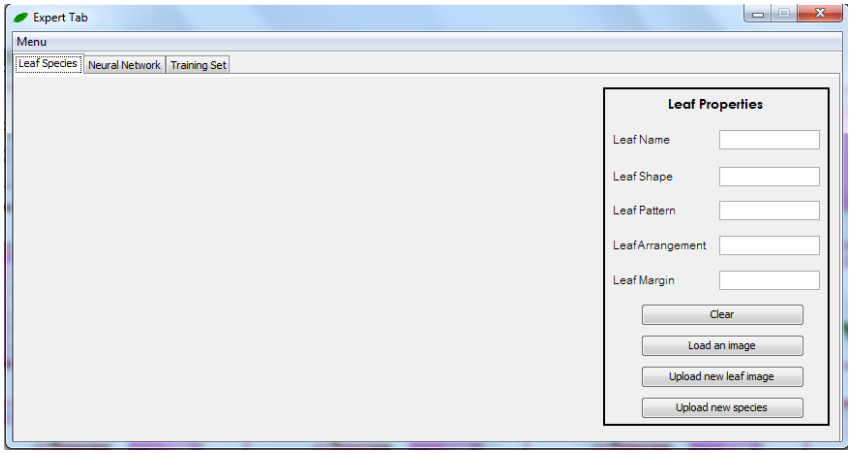


Figure 40: Load leaf images

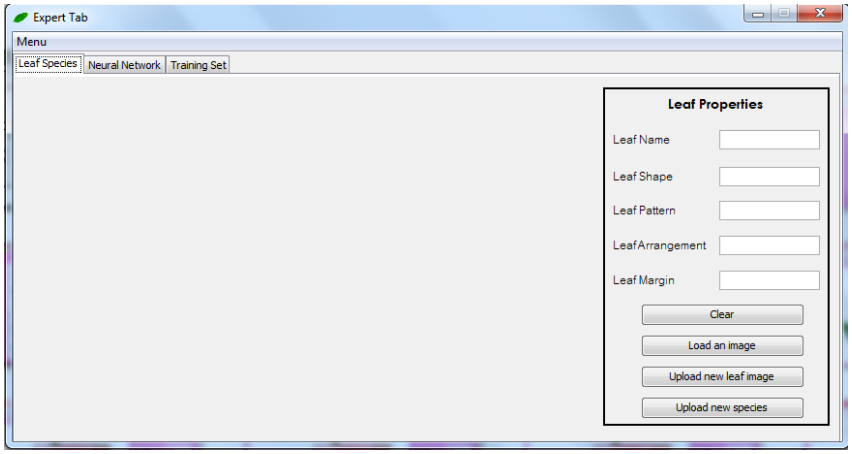


Figure 41: Load an image to upload

Once the expert has opted to load an image, the instructions will be shown on the screen before he can upload a leaf image. The instructions are shown in Figure 42. Then the expert will be able to select the leaf image to be uploaded as seen in Figure 43.

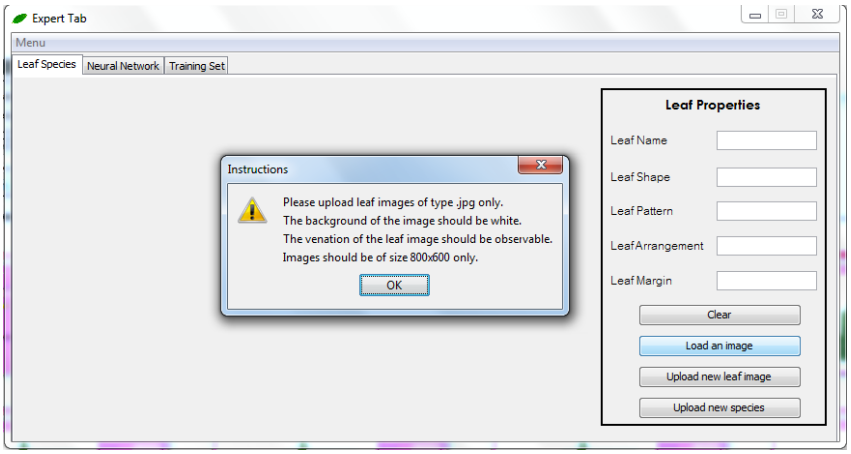


Figure 42: Instructions and Guidelines on Uploading

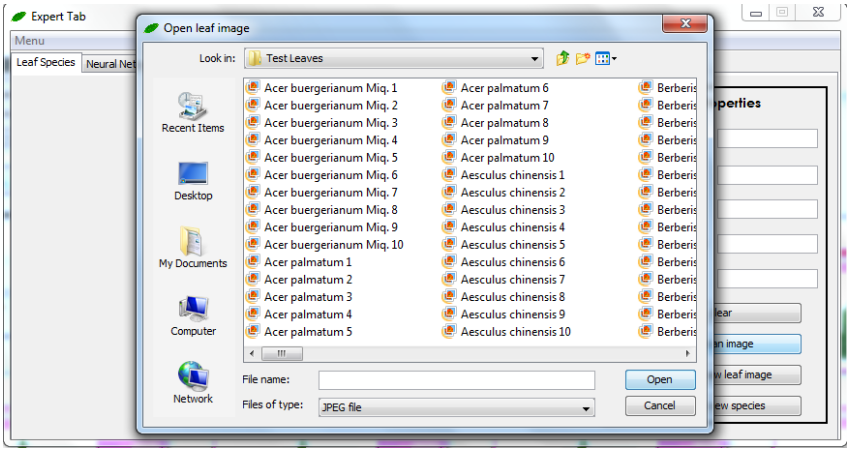


Figure 43: Choosing an Image

Upon selection of a leaf image, the expert will be taken back to his screen with the preview of the leaf image on the left and the panel of leaf properties on the right as can be seen in Figure 44. The expert can enter one or more of the additional information of the plant species that he wants to upload. An example is shown in Figure 45.

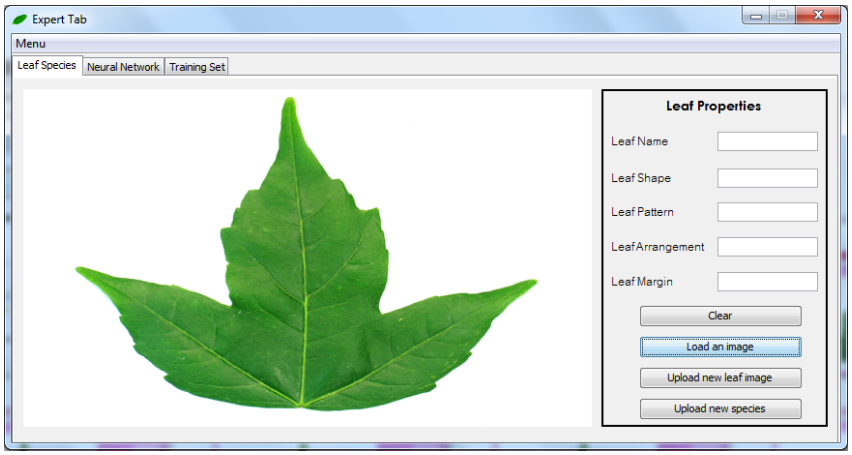


Figure 44: Preview of leaf image

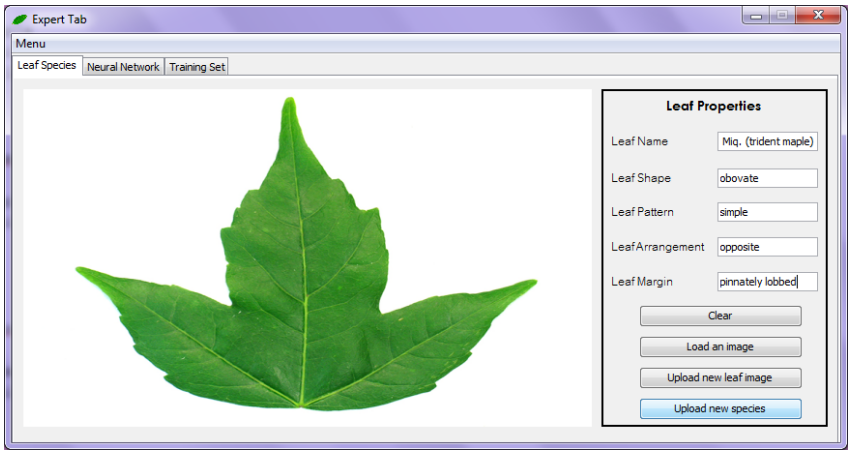


Figure 45: Enter leaf properties

3. **Upload a leaf image to add to database**

After providing the leaf properties, the morphology expert can now choose whether to upload a new leaf image of an existing plant species in the database or add a new plant species to the database of species. If he chooses the latter, after pressing the *Add new leaf species* button a prompt window will appear like the one seen in Figure 46. But if the former is chosen, after pressing the *Add new leaf image* button the prompt window would look like the one in Figure **??**.

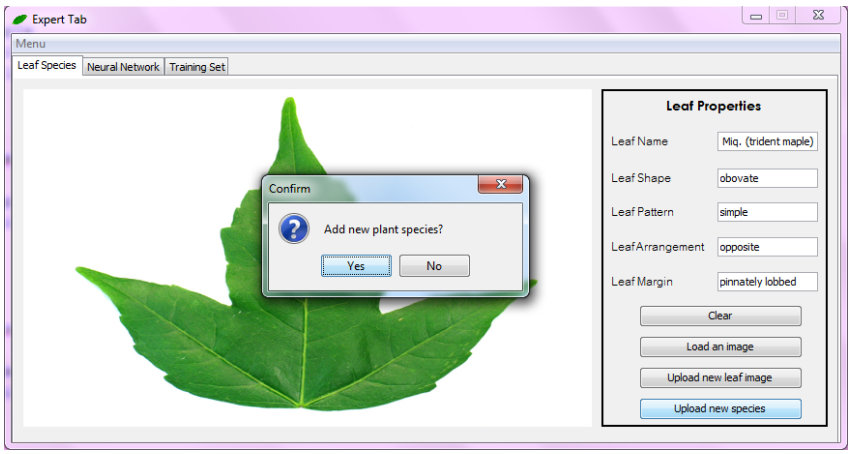


Figure 46: Uploading a new plant species

If the expert succeeds in uploading the new plant species, the system will inform him by showing a window as seen in Figure 47 and the window seen in Figure 48 which will also be seen when he successfully uploads a new leaf image.

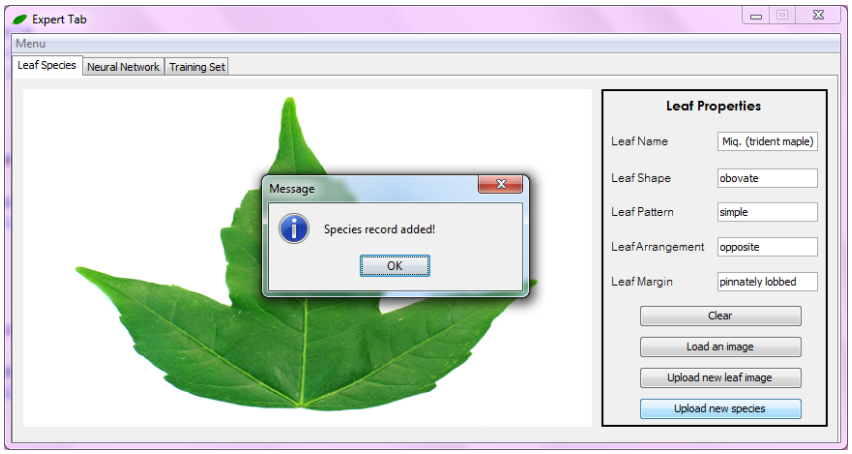


Figure 47: Uploaded new plant species

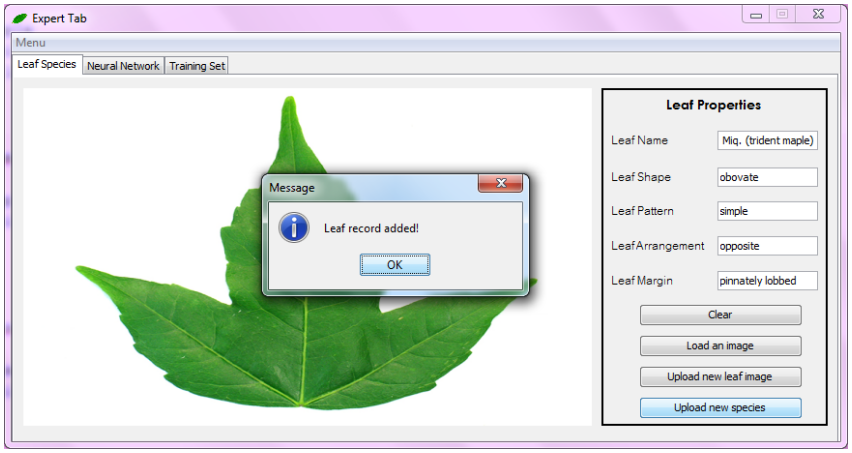


Figure 48: Uploaded new leaf image

4. **Adjusting detection parameters**

The user can choose to adjust the low threshold of the canny edge detector by choosing it in the menu bar as seen in Figure 49. The prompt window will appear as Figure 50.

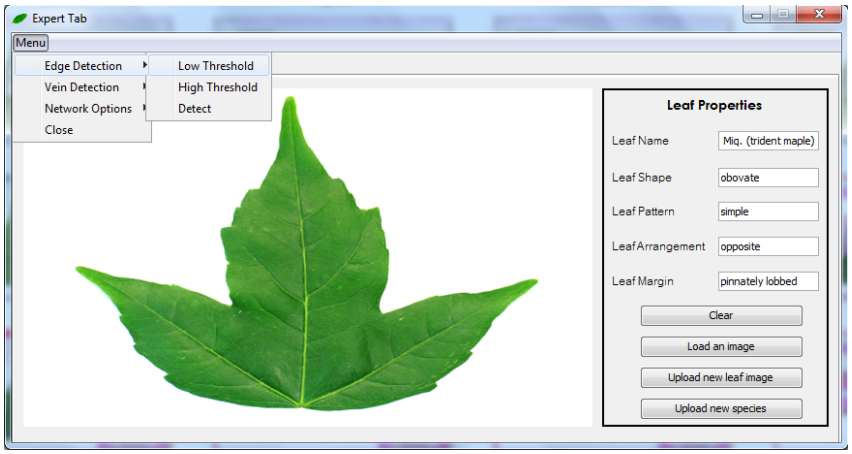


Figure 49: Adjusting the low threshold

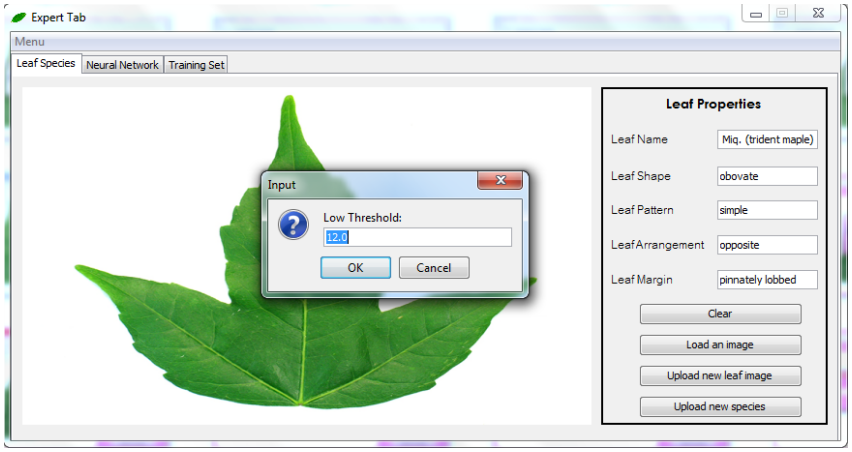


Figure 50: Input low threshold value

After supplying the low threshold, the expert can also adjust the high threshold as seen in Figure 51. The prompt window will then appear as Figure 52.

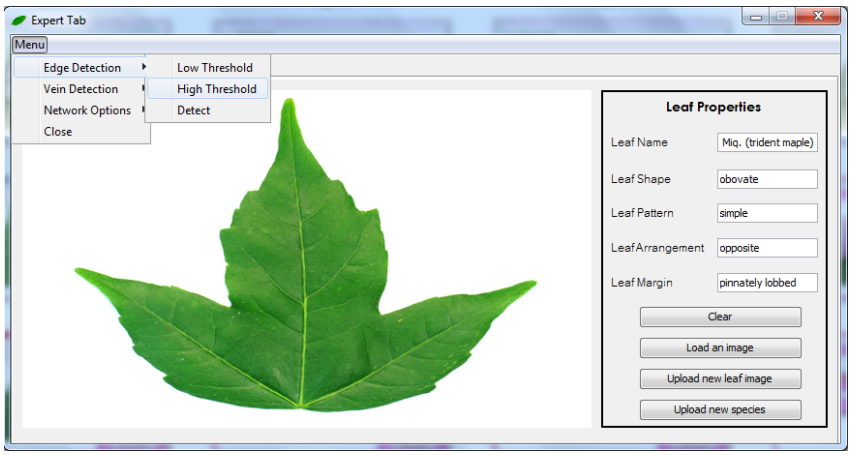


Figure 51: Adjusting the high threshold

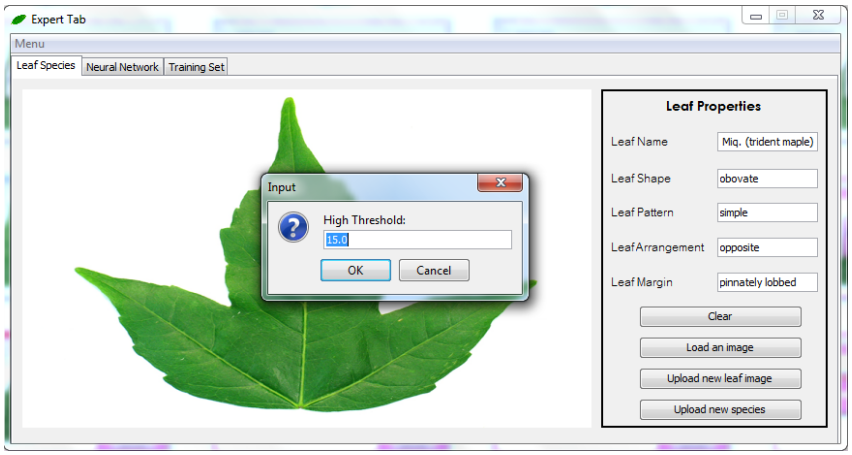


Figure 52: Input high threshold value

Finally, the user can detect the leaf edge by choosing the detect item in the *Edge detection* menu in the menu bar following Figure 53. The detected leaf image will be shown and will replace the original leaf image in its position like the screen shown in Figure 54.

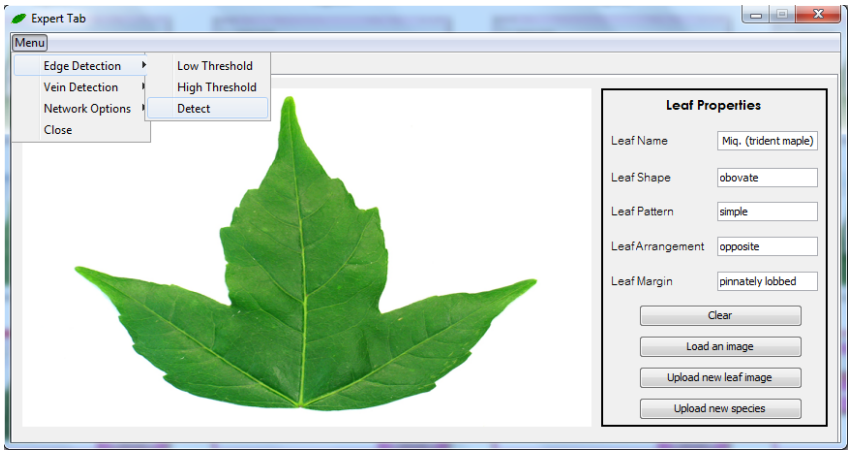


Figure 53: Detect edge

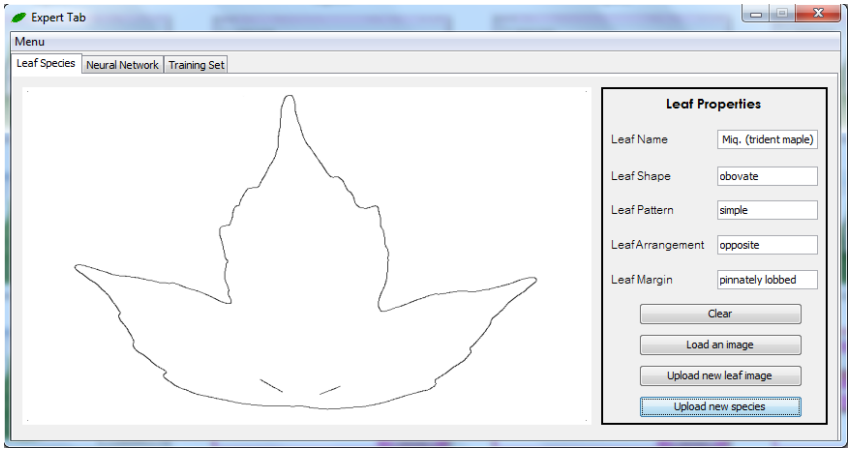


Figure 54: View detected leaf image edge

The expert can also adjust the radius of the morphological opening to be performed on the leaf image. The range of the radius is from 1-4. He can choose to adjust the radius from selecting the *Opening radius* option in the *Vein detection* menu in the menu bar as depicted in Figure 55. After selecting the option, the user will be prompted as in Figure 56.

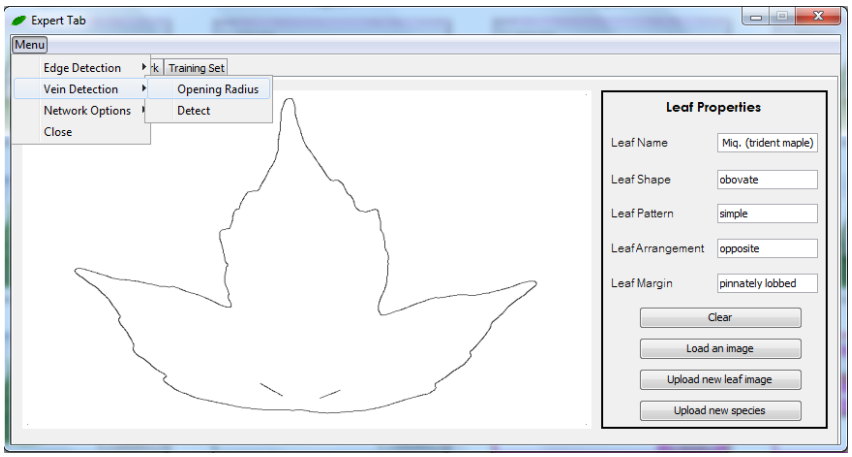


Figure 55: Adjust radius of morphological opening

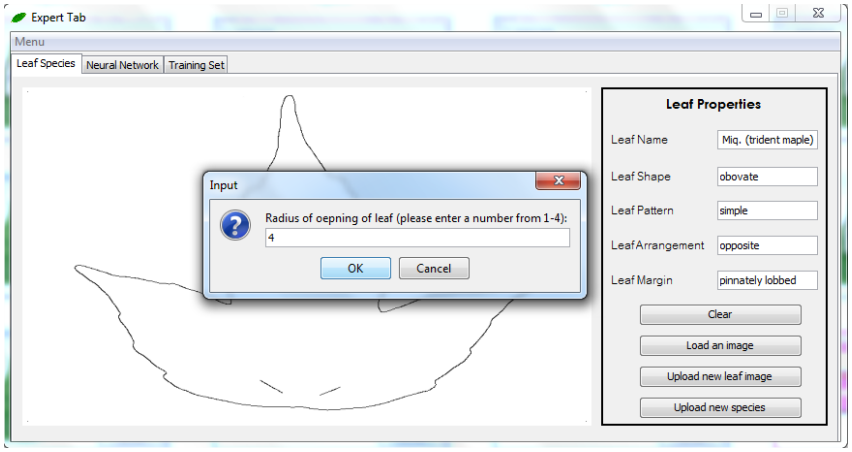


Figure 56: Input opening radius value

And then the expert can detect the vein image by choosing *detect* in the *Vein detection* menu in the menu bar as seen in Figure 57. The detected vein image will be depicted on the screen as shown in Figure 58.

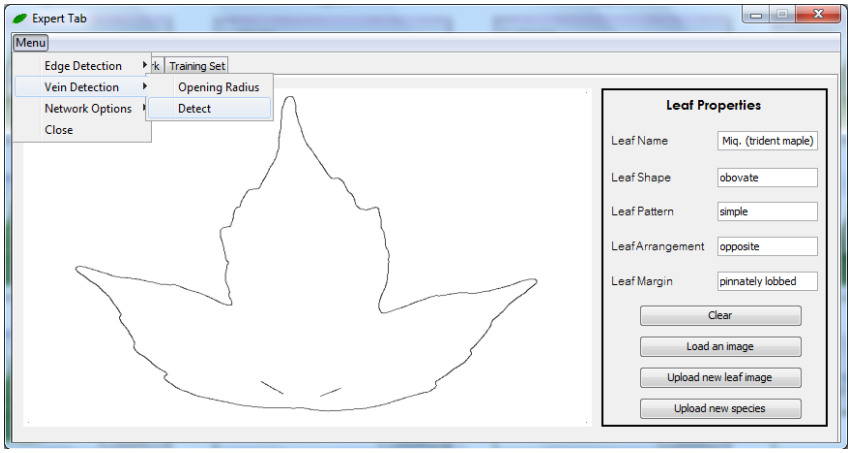


Figure 57: Detect vein

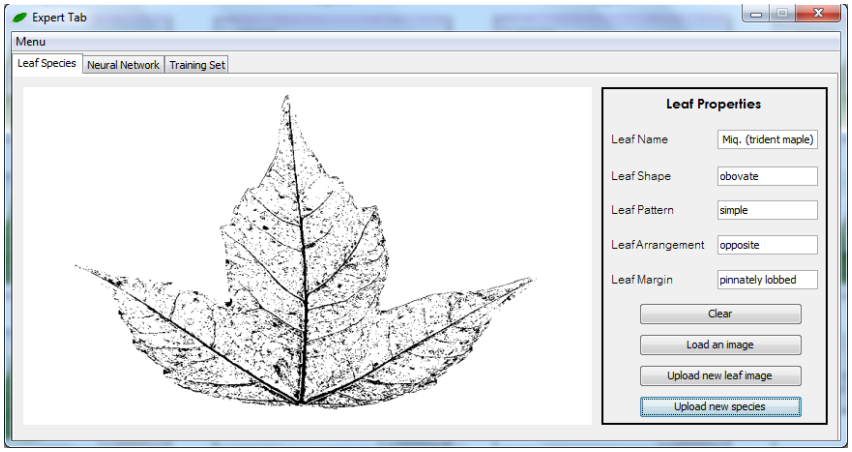


Figure 58: View detected leaf vein image

5. **Train Neural Network**

The *Neural Network* tab consists of an error graph depicting the network error, the summary of the database of leaves, and the network parameters. The summary of the database of leaves consists of the number of leaf images and the corresponding number of species in the database of leaves. The network parameters are composed of the number of input, hidden, and output nodes, the learning rate, and the max error of the network. The adjustable parameters are the number of hidden nodes, the learning rate, and the max error. The training commences upon pressing the Train button as seen in Figure 59.

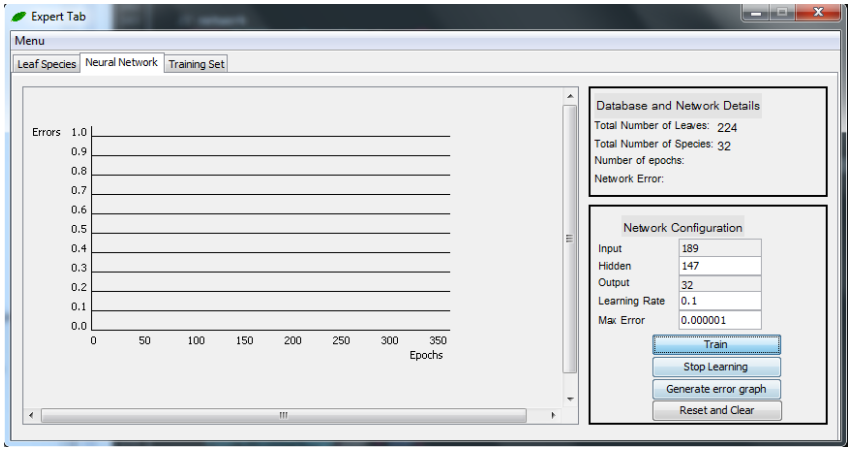


Figure 59: Train neural network

6. **Upload neural network**

The network will stop learning manually by pressing the *Stop Learning* button or once the max error has been surpassed as can be seen in Figure 60. The maximum error set was 0.00001 and the network error converged to this value. Once the network stopped learning, the expert can refresh the error graph by pressing the *Generate error graph* button as seen in Figure 61.

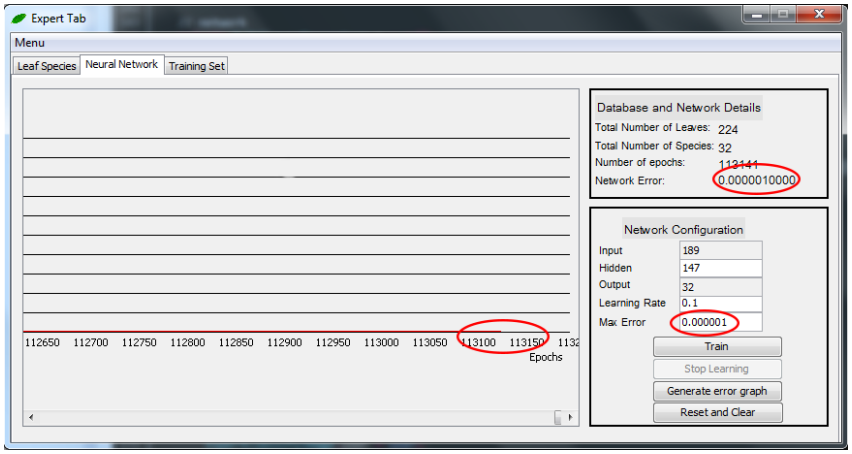


Figure 60: Neural network converged to maximum error

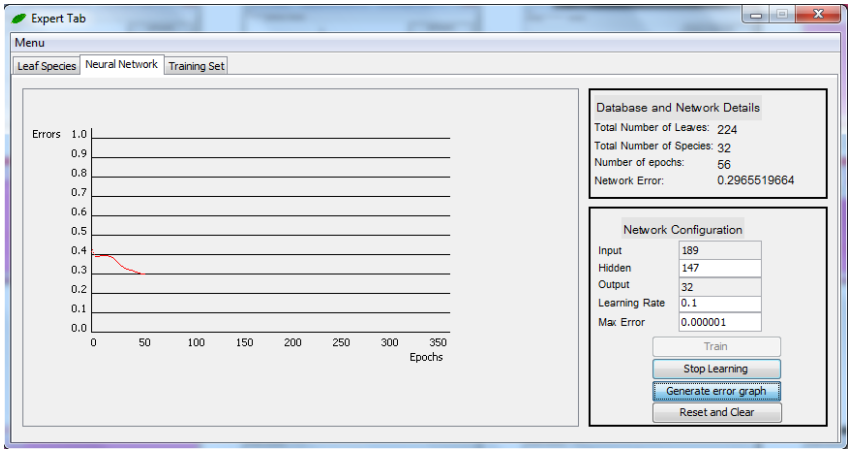


Figure 61: Generate error graph

When the expert is satisfied with the network he can opt to save the file by selecting *Save* in the *Network options* menu in the menu bar as seen in Figure 62 and Figure 63.

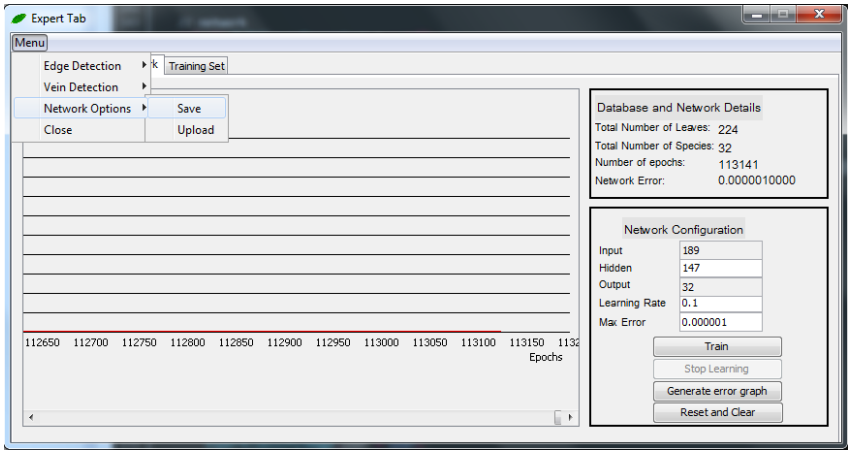


Figure 62: Save neural network to file

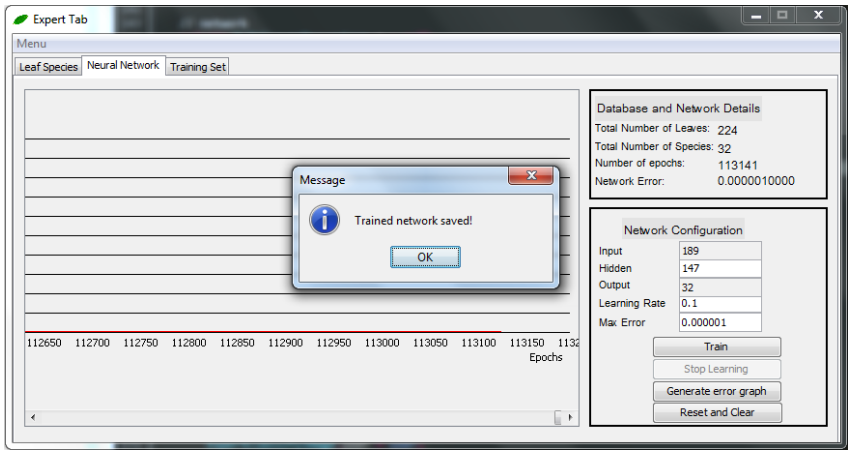


Figure 63: Saved neural network

After saving the neural network file, the expert can upload it to the reference database for the public user to download. The system will inform the user of successful uploading like the one seen in Figure 64 and Figure 65.

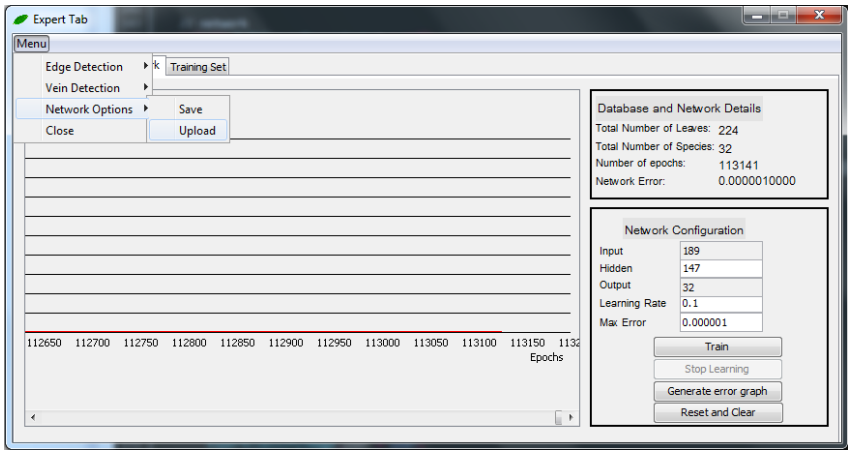


Figure 64: Upload neural network

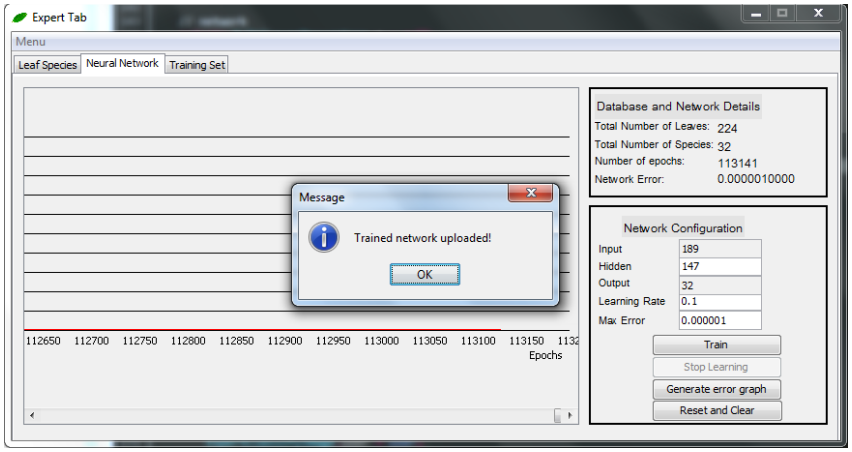


Figure 65: Uploaded neural network

7. **Delete leaf image**

The expert can also view the list of uploaded images on the reference database and browse through the images as seen in Figure 66. He can choose to update the list or delete a leaf image. When the *Update* button is pressed as seen in Figure 67, the system will prompt as seen in Figure 68.

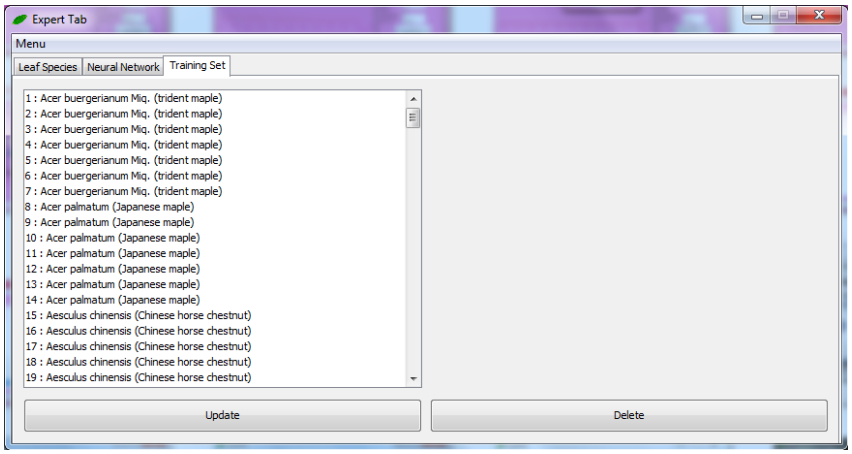


Figure 66: View training set

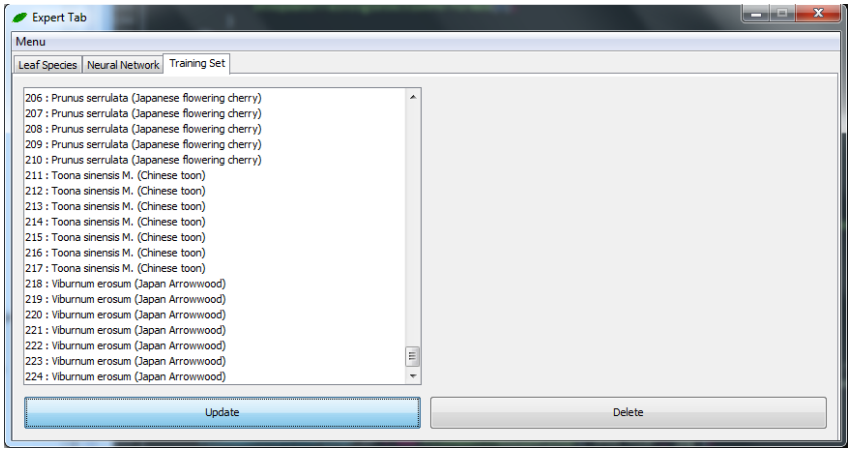


Figure 67: Update training set

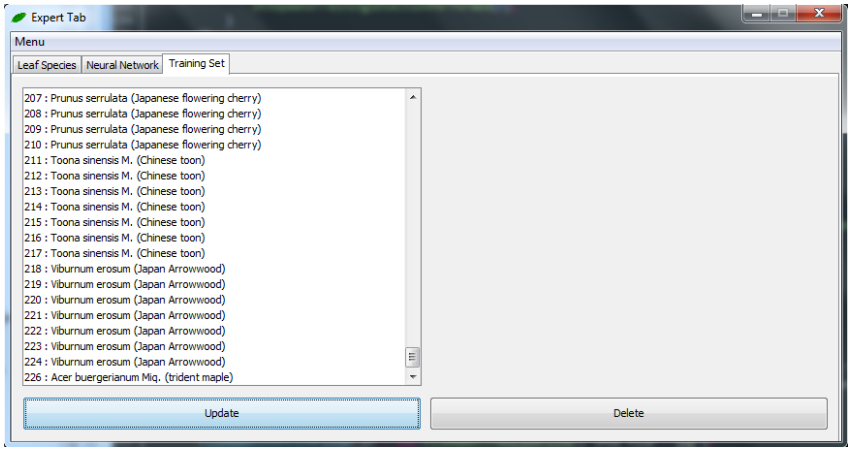


Figure 68: Update training set

The leaf image will be deleted by selecting the leaf image to be deleted on the list and pressing the *Delete* button as illustrated in Figure 69. If the deletion is successful, the system prompts the user as seen in Figure 70 and updates the list accordingly.

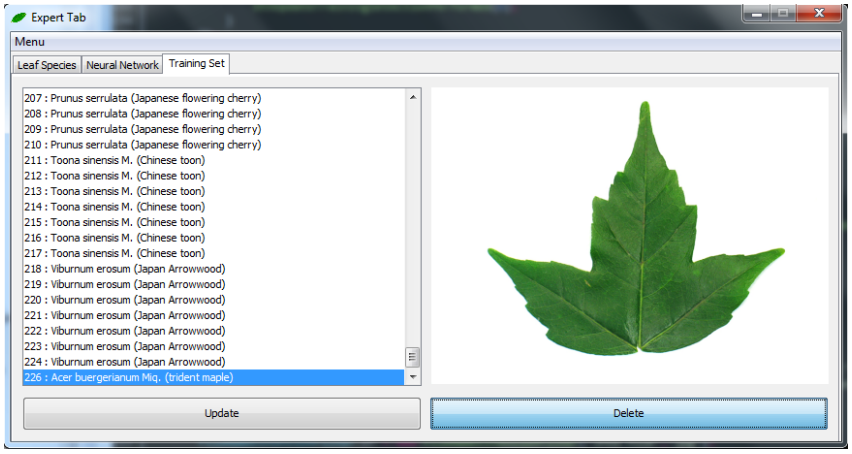


Figure 69: Delete leaf image

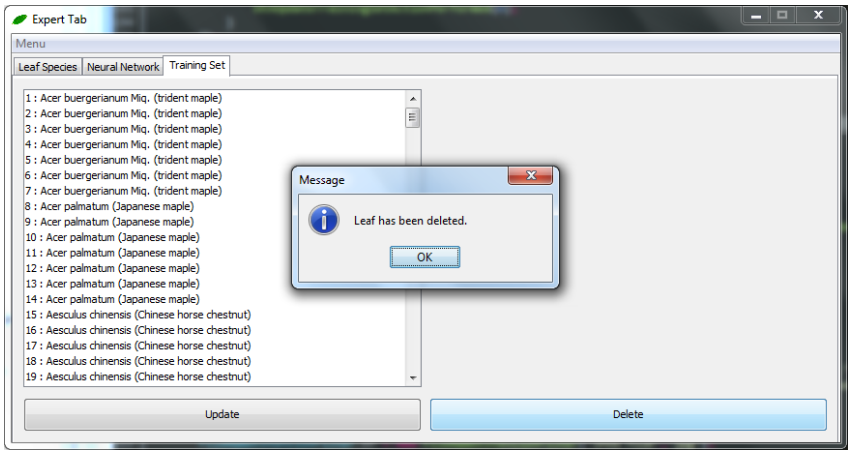


Figure 70: Leaf image deleted

VII. Discussions

The Leaf Recognition Application by Leaf Shape and Venation using Artificial Neural Networks (LeaVeS) is a stand-alone application that aims to classify a plant species by a leaf image alone. The application has two users: the public user and the morphology expert. While LeafRapp [7] has the same goal, the said application only utilizes of the leaf's edge to be able to classify it. LeaVeS detects the leaf shape and venation to make distinct the leaf images that almost have the same shape but differ in venation.

The former shape descriptor measured distances at 10°and to be able to address this, the shape descriptor is now measured at 2°intervals. On another note, the venation was extracted using the method implemented in [4]. A flat, structuring element was used with varying radii to be able to describe the venation completely. This set of features are the input of the neural network for classification. Once the leaf has been classified, the public user can enter optional additional information about the leaf to be able to refine the search. Upon entering the information, the updated list of classifications is shown.

The neural network used by the system is Multilayer Perceptron. Multilayer Perceptron was the choice of artificial neural network for its simplicity and basic training parameters. The network was implemented from the open-source package of neuroph. The MLP consisted of three layers where the input layer consists of 189 nodes and the output consists of 32. The nodes of the input layer represent the number of features extracted and the number of nodes of the input layer is the number of species in the database. The training set consisted of 224 leaves - 7 leaves per species and the test set consisted of 96 leaves - 3 for each species. The training was done by the morphology expert and uploaded for the public users' use.

To be able to perform the abovementioned functions, the application was written in Java and connected to a database where the leaf images, species details, and network files are stored. To connect to this MySQL database, the Java Database Connectivity (JDBC) was utilized. The database was designed in such a way that there were only three tables and one referenced another. Therefore, the database was modeled as basicly as it is possible.

The application hastens the plant recognition process because it makes readily available a system to classify leaf images that would take a normal morphology expert hours to do. Given a leaf image, the morphologu expert would need additional information, like the leaf's pattern, arrangement, or if the plant has any flowers and sometimes even a physical specimen to be able to correctly classify a plant. It is a convenient way to identify a leaf image for people who doesn't have taxonomic training.

The main issue encountered during the development of the application is the preprocessing. As with all neural network applications, the preprocessing consists of an estimated 70% of the whole system. Without preprocessing, the neural network cannot be implemented, and consequentially, the other minor functionalities. For the leaf venation, the operations done to be able to extract it was very complicated as from the one proposed but to be able to describe it was relatively easy compared to the latter.

Another issue was the data source as the required number of species to be able to proceed with the application was 32. It was difficult to collect 10 samples of each, especially to identify each and every species. Some of the leaves were from [23] that is a program based on [4]. The remaining leaves that were manually collected were identified by a morphology expert. All the other related leaf properties were carefully researched and attributed to [24] and [25].

After the preprocessing stage, the training of the neural network was more complicated than expected. The challenge was not to train the network perse, but to get the optimal parameters for the network. After the long training, it was also a difficult task to test the network on each of the leaves on the test set and compute for the network's accuracy. The actual problem was not to train the network but to make sure that the trained network was efficient. The neural network has an average accuracy of 70-90% over the dataset that was compiled.

VIII. Conclusions

LeaVeS has been developed to allow a public user and a morphology expert freely download the software. This is in congruence to addressing the need for a reliable and convenient way to identify a plant based on its leaf image. It is made readily available so as to cater to the need for a fast system that enables the user to classify his own leaf images without the need for a taxonomic background.

As a guide, both the public user and the morphology expert are shown the instructions and guidelines on uploading the correct leaf image to be able to efficiently extract the features from it. The user and the expert can then select the leaf image to be uploaded and classified.

The morphology expert can upload a new leaf species and correspondingly, new leaf images to update the reference database of leaves. This is in preparation for training the neural network. The features that will be utilized by the network are extracted carefully from the leaf images using methods that were previously used by existing systems, but modified and developed. The said features that are extracted are the leaf shape and its corresponding descriptor, and the leaf venation and its descriptor as well.

The leaf shape was extracted using the Canny Edge algorithm [20] and the descriptor of the shape are the four central Hu moments, and radii using the Centroid-Radii model in [1]. On the other hand, the leaf vein was extracted using the method used in [4]. The method consisted of performing morphological opening and image difference on the input leaf image.

These features, along with the leaf images and the plant species, are stored in the reference database of leaves for the user to download and for the training of the neural network. The database is password protected but allows both the public user and the morphology expert access to it. However, only the morphology expert can perform changes on it.

The morphology expert then trains the neural network to find the most appropriate parameters for the network to be at its most accurate. Once the expert has deduced the right combination of the network parameters, he has the option to save and upload

the network. Once he uploads the network to the reference database, it is free to be downloaded by the public user to use in his own end system.

The expert also has the option to delete a leaf image from the reference database if he sees it fit. He can view the list of leaf images right in the system and delete it as well. When the expert deletes a leaf image, it will no longer be available for download the next time the public user chooses to update his own copy of the reference leaf images.

The public user, as mentioned above, can freely update the reference database of leaf images and the network anytime he wishes to do so. The network file, once downloaded, will be used to classify the leaf images that the user will upload. The uploaded leaf images of the user from his own end system will be processed by the system in the same way it processes the uploaded leaf images by the morphology expert.

Once the leaf image has been processed by the system, the neural network will be executed to be able to make sense of the extracted features from the leaf image. The network will provide the percentage of match of the leaf image to each species in the reference database and the system will allow the user to view the list of probable classifications. Once the list of probable classifications is made available to the user, he can opt to enter optional additional information about the leaf to refine the list of probable classifications to those species that exhibit the said information.

Both the public user and the morphology expert has the liberty to detect and view the leaf edge and vein images for their convenience. This is an added feature for the user and the morphology expert can adjust the parameters of the detection methods.

LeaVeS provides the user with a system that incorporates both the leaf shape and venation in order to classify a leaf image. It improves previously implemented shape descriptors by a developing upon the former and enhancing it to accomodate more features that can improve the performance of the network. The vein descriptors that is an improvement of the existing systems are implemented so as to simplify the extraction of the veins data and to address the need for differentiation between leaf images that almost resemble the same leaf shape. The data is then fed into the neural network to aid in the efficient classification of the image.

The average accuracy of the system based on the highest percentage of match would be 70-90%. But if the 2nd and 3rd highest percentage of match would be considered, the lower bound of the average accuracy would go up by as much as 15% and the system could classify correctly 95% of the species in the given dataset. Compared to the previous methods, the accuracy of the system is lower but the dataset used by the system is more populated than the previous systems. Also, the 70-30 learning rule was implemented as opposed to 50-50 which allowed a smaller margin of error because there were more leaves to test a specific species with.

IX. Recommendations

LeaVeS can be improved by adding another feature of the leaf that will prove to be most definitive among others. In [15], aside from the leaf shape and the venation, the leaf dent was also extracted. This would prove to be efficient on a dataset that is comprised of leaves that are almost the same in shape and venation. Also, a projection histogram could be utilized to determine the main vein, and to differentiate the different leaf regions. [8]. This could add to the leaf vein descriptor and to the numerical data that is extracted from the leaf vein image.

However, to avoid overfeeding the network and confusing it with so much numerical data, Principal Component Analysis could be performed to determine which of the numerical data should be kept and those that should be disposed. [4]. This would be an efficient technique to model the leaf and its vital features alone.

Another path to tread is to improve on the classifier used. Although MLP is favored for its flexibility in classifying non-linear data [12], there is reason to believe that the Probabilistic Neural Network used in [4] also has its advantages over MLP. The main difference would be the transfer functions in the hidden layer.

Given the presumption that MLP is still the best option for the problem, there could be explorations on its best implementation. This is in accordance to the criterion of training time and efficiency, as well. Other open source applications that offer MLP training and testing may have its edge over the one implemented in LeaVeS, neuroph.

X. Bibliography

[1] J. Chaki and R. Parekh, "Plant leaf recognition using shape based features and neural network classifiers," *International Journal of Advanced Computer Science and Applications*, vol. 2, no. 10, p. 7, 2011.

[2] H. Hajjdiab and I. A. Maskari, "Plant species recognition using leaf contours," *IEEE*, 2011.

[3] L. Gao, X. Lin, M. Zhong, and J. Zeng, "A neural network classifier based on prior evolution and iterative approximation used for leaf recognition," in *Sixth International Conference on Natural Computation*, 2010.

[4] S. Wu, F. Bao, E. Xu, Y.-X. Wang, Y.-F. Chang, and Q.-L. Xiang, "A leaf recognition algorithm for plant classification using probabilistic neural network," *IEEE*, 2007.

[5] S. Zhang and Y.-K. Lei, "Modified locally linear discriminant embedding for plant leaf recognition," *Elsevier - Neurocomputing*, 2011.

[6] J. Cope, D. Corney, J. Clark, P. Remagnino, and P. Wilkin, "Plant species identification using digital morphometrics: a review," *Elsevier*, 2011.

[7] J. P. Sosa, "Plant leaf recognition application (leafrapp)," April 2013.

[8] K. Lee and K.-S. Hong, "An implementation of leaf recognition system using leaf vein and shape," *International Journal of Bio-Science and Bio-Technology*, 2013.

[9] P. Obico, "Interview," October 2014.

[10] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years," *IEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[11] S. Ghorpade, J. Ghorpade, and S. Mantri, "Pattern recognition using neural networks," *International Journal of Computer Science and Information Technology*, 2010.

[12] L. Noriega, *Multilayer Perceptron Tutorial*.

[13] J.-X. Du, X.-F. Wang, and G.-J. Zhang, "Leaf shape based plant species recognition," *Elsevier - Applied Mathematics and Computation*, 2007.

[14] C.-Y. Gwo, C.-H. Wei, and Y. Li, "Rotary matching of edge features for leaf recognition," *Elsevier - Computers and Electronics in Agriculture*, 2013.

[15] *Feature Extraction and Automatic Recognition of Plant Leaf Using Artificial Neural Network*, ch. Neural Networks and Genetic Algorithms. Center for Computing Research of IPN, 2006.

[16] "Leaf classification," October 2013.

[17] "Plant identification: Examining leaves," October 2013.

[18] "Leaf - venation," October 2013.

[19] N. J. Nilsson, *Introduction to Machine Learning*.

[20] "Canny edge detection," March 23 2009.

[21] "Morphological image processing," December 2014.

[22] M. Negnevitsky, *Artificial Intelligence A Guide to Intelligent Systems*, ch. Artificial Neural Networks. Pearson Education Limited, 2002.

[23] "Flavia," November 2013.

[24] "Kwantlen polytechnic university school of horticulture plant database," January 2014.

[25] "Hortipedia the gardeninfoportal," January 2014.

XI. Appendix

A. Source Code

1. AbstractOperation.java

```
import java.awt.image.BufferedImage;

/**
 * Abstract morphological base
 * class.
 *
 * @author Tomas Toss
 *
 */
public abstract class AbstractOperation
        implements MorphologicalOperation {

        protected int shapeSize;

        /**
         * Constructs a structuring
         * element shape. The values
         * in the structuring
         * element array are either
         * set to 1 or 0, 1 if that
         * position in the
         * structuring element array
         * is a part of the
         * structuring element shape
         *
         * @param shape
         *           The shape of the
         *           structuring
         *           element, see @
         *           MorphologicalOperation
         *           .
         *           STRUCTURING_ELEMENT_SHAPE
         *           } for available
         *           shapes
         * @param shapeSize
         *           The size from
         *           the middle of
         *           the shape to
         *           exterior of the
         *           shape. (Total
         *           size =
         *           2*shapeSize+1)
         * @return The constructed
         *          structuring
         *          element
         */
        protected short[][] constructShape(
                STRUCTURING_ELEMENT_SHAPE shape,
                int shapeSize) {

                int size = 2 * shapeSize + 1;
                short[][] structElem =
                        new short[size][size];
                switch (shape) {
                        case SQUARE:

                                for (int i = 0; i < size; i++) {
                                        for (int j = 0; j < size; j++) {
                                                structElem[i][j] = 1;
                                        }
                                }
                                break;
                        case VERTICAL_LINE:
                                for (int i = 0; i < size; i++) {
                                        structElem[i][shapeSize] = 1;
                                }
                                break;
                        case HORIZONTAL_LINE:
                                for (int i = 0; i < size; i++) {
                                        structElem[shapeSize][i] = 1;
                                }
                                break;
                        default:
                                for (int i = 0; i < size; i++) {
                                        for (int j = 0; j < size; j++) {
                                                structElem[i][j] = 1;
                                        }
                                }
                }
                return structElem;
        }

        /**
         * Execute the morphological
         * operation on the supplied
         * image
         *
```

```
         * @param img
         *          The image to
         *          operate on
         * @return The morphological
         *         adjusted image
         */
        @Override
        public abstract BufferedImage
                execute(BufferedImage img);

        @Override
        public int getShapeSize() {
                return shapeSize;
        }
}
```

2. CannyEdgeDetector.java

```
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.Arrays;

/**
 * <p>
 * <em>This software has been released into the public domain.
 * <strong>Please read the notes in this source file for additional information.
 * </strong></em>
 * </p>
 *
 * <p>
 * This class provides a
 * configurable implementation
 * of the Canny edge detection
 * algorithm. This classic
 * algorithm has a number of
 * shortcomings, but remains
 * an effective tool in many
 * scenarios.
 * <em>This class is designed
 * for single threaded use only.</em>
 * </p>
 *
 * <p>
 * Sample usage:
 * </p>
 *
 * <pre>
 * <code>
 * //create the detector
 * CannyEdgeDetector detector = new CannyEdgeDetector();
 * //adjust its parameters as desired
 * detector.setLowThreshold(0.5f);
 * detector.setHighThreshold(1f);
 * //apply it to an image
 * detector.setSourceImage(frame);
 * detector.process();
 * BufferedImage edges = detector.getEdgesImage();
 * </code>
 * </pre>
 *
 * <p>
 * For a more complete
 * understanding of this edge
 * detector's parameters
 * consult an explanation of
 * the algorithm.
 * </p>
 *
 * @author Tom Gibara
 *
 */

public class CannyEdgeDetector {

        // statics
        private final static float GAUSSIAN_CUT_OFF =
                0.005f;
        private final static float MAGNITUDE_SCALE =
                100F;
        private final static float MAGNITUDE_LIMIT =
                1000F;
        private final static int MAGNITUDE_MAX =
                (int) (MAGNITUDE_SCALE * MAGNITUDE_LIMIT);

        // fields
        private ArrayList<Integer> outline =
                new ArrayList<Integer>();
        private int centroidX; // modification
        private int centroidY; // modification

        private int height;
        private int width;
        private int picsize;
        private int[][] pixelArr; // SOSA
        private int[] data;
        private int[] magnitude;
        private BufferedImage sourceImage;
        private BufferedImage edgesImage;
```

```
private float gaussianKernelRadius;
private float lowThreshold;
private float highThreshold;
private int gaussianKernelWidth;
private boolean contrastNormalized;

private float[] xConv;
private float[] yConv;
private float[] xGradient;
private float[] yGradient;

// constructors

/**
 * Constructs a new detector
 * with default parameters.
 */

public CannyEdgeDetector(){
        lowThreshold = 2.5f;
        highThreshold = 7.5f;
        gaussianKernelRadius = 2f;
        gaussianKernelWidth = 16;
        contrastNormalized = false;
}

// accessors

/**
 * The image that provides
 * the luminance data used
 * by this detector to
 * generate edges.
 *
 * @return the source image,
 *         or null
 */

public BufferedImage getSourceImage() {
        return sourceImage;
}

/**
 * Specifies the image that
 * will provide the
 * luminance data in which
 * edges will be detected. A
 * source image must be set
 * before the process method
 * is called.
 *
 * @param image
 *          a source of
 *          luminance data
 */

public void setSourceImage(
        BufferedImage image) {
        sourceImage = image;
}

/**
 * Obtains an image
 * containing the edges
 * detected during the last
 * call to the process
 * method. The buffered
 * image is an opaque image
 * of type BufferedImage.
 * TYPE_INT_ARGB in which
 * edge pixels are white and
 * all other pixels are
 * black.
 *
 * @return an image
 *         containing the
 *         detected edges,
 *         or null if the
 *         process method
 *         has not yet been
 *         called.
 */

public BufferedImage getEdgesImage() {
        return edgesImage;
}

/**
 * Sets the edges image.
 * Calling this method will
 * not change the operation
 * of the edge detector in
 * any way. It is intended
 * to provide a means by
 * which the memory
 * referenced by the
 * detector object may be
 * reduced.
```

```
     *
     * @param edgesImage
     *            expected (though
     *            not required) to
     *            be null
     */

    public void setEdgesImage(
            BufferedImage edgesImage) {
            this.edgesImage = edgesImage;
    }

    /**
     * The low threshold for
     * hysteresis. The default
     * value is 2.5.
     *
     * @return the low
     *          hysteresis
     *          threshold
     */

    public float getLowThreshold() {
            return lowThreshold;
    }

    /**
     * Sets the low threshold
     * for hysteresis. Suitable
     * values for this parameter
     * must be determined
     * experimentally for each
     * application. It is
     * nonsensical (though not
     * prohibited) for this
     * value to exceed the high
     * threshold value.
     *
     * @param threshold
     *            a low hysteresis
     *            threshold
     */

    public void setLowThreshold(
            float threshold) {
            if (threshold < 0)
                    throw new IllegalArgumentException();
            lowThreshold = threshold;
    }

    /**
     * The high threshold for
     * hysteresis. The default
     * value is 7.5.
     *
     * @return the high
     *          hysteresis
     *          threshold
     */

    public float getHighThreshold() {
            return highThreshold;
    }

    /**
     * Sets the high threshold
     * for hysteresis. Suitable
     * values for this parameter
     * must be determined
     * experimentally for each
     * application. It is
     * nonsensical (though not
     * prohibited) for this
     * value to be less than the
     * low threshold value.
     *
     * @param threshold
     *            a high
     *            hysteresis
     *            threshold
     */

    public void setHighThreshold(
            float threshold) {
            if (threshold < 0)
                    throw new IllegalArgumentException();
            highThreshold = threshold;
    }

    /**
     * The number of pixels
     * across which the Gaussian
     * kernel is applied. The
     * default value is 16.
     *
     * @return the radius of the
     *          convolution
     *          operation in
     *          pixels
```

```
 */

public int getGaussianKernelWidth() {
        return gaussianKernelWidth;
}

/**
 * The number of pixels
 * across which the Gaussian
 * kernel is applied. This
 * implementation will
 * reduce the radius if the
 * contribution of pixel
 * values is deemed
 * negligable, so this is
 * actually a maximum
 * radius.
 *
 * @param gaussianKernelWidth
 *           a radius for the
 *           convolution
 *           operation in
 *           pixels, at least
 *           2.
 */

public void setGaussianKernelWidth(
        int gaussianKernelWidth) {
        if (gaussianKernelWidth < 2)
                throw new IllegalArgumentException();
        this.gaussianKernelWidth =
                gaussianKernelWidth;
}

/**
 * The radius of the
 * Gaussian convolution
 * kernel used to smooth the
 * source image prior to
 * gradient calculation. The
 * default value is 16.
 *
 * @return the Gaussian
 *          kernel radius in
 *          pixels
 */

public float
        getGaussianKernelRadius() {
        return gaussianKernelRadius;
}

/**
 * Sets the radius of the
 * Gaussian convolution
 * kernel used to smooth the
 * source image prior to
 * gradient calculation.
 *
 * @return a Gaussian kernel
 *          radius in pixels,
 *          must exceed 0.1f.
 */

public void setGaussianKernelRadius(
        float gaussianKernelRadius) {
        if (gaussianKernelRadius < 0.1f)
                throw new IllegalArgumentException();
        this.gaussianKernelRadius =
                gaussianKernelRadius;
}

/**
 * Whether the luminance
 * data extracted from the
 * source image is
 * normalized by linearizing
 * its histogram prior to
 * edge extraction. The
 * default value is false.
 *
 * @return whether the
 *          contrast is
 *          normalized
 */

public boolean isContrastNormalized() {
        return contrastNormalized;
}

/**
 * Sets whether the contrast
 * is normalized
 *
 * @param contrastNormalized
 *           true if the
 *           contrast should
 *           be normalized,
 *           false otherwise
```

```
 */

public void setContrastNormalized(
        boolean contrastNormalized) {
        this.contrastNormalized =
                contrastNormalized;
}

// methods
// modification
public void preprocess() {
        width = sourceImage.getWidth();
        height = sourceImage.getHeight();
        picsize = width * height;
        initArrays();
        readLuminance();

}

public void process() {
        width = sourceImage.getWidth();
        height = sourceImage.getHeight();
        picsize = width * height;
        initArrays();
        readLuminance();
        if (contrastNormalized)
                normalizeContrast();
        computeGradients(
                gaussianKernelRadius,
                gaussianKernelWidth);
        int low =
                Math.round(lowThreshold
                        * MAGNITUDE_SCALE);
        int high =
                Math.round(highThreshold
                        * MAGNITUDE_SCALE);
        performHysteresis(low, high);
        thresholdEdges();
        writeEdges(data);
}

// private utility methods
private void initArrays() {
        if (data == null
                || picsize != data.length) {
                data = new int[picsize];
                magnitude = new int[picsize];

                xConv = new float[picsize];
                yConv = new float[picsize];
                xGradient = new float[picsize];
                yGradient = new float[picsize];
        }
}

void
        computeGradients(
                float kernelRadius,
                int kernelWidth) {

        // generate the gaussian
        // convolution masks
        float kernel[] =
                new float[kernelWidth];
        float diffKernel[] =
                new float[kernelWidth];
        int kwidth;
        for (kwidth = 0; kwidth < kernelWidth; kwidth++) {
                float g1 =
                        gaussian(kwidth, kernelRadius);
                if (g1 <= GAUSSIAN_CUT_OFF
                        && kwidth >= 2)
                        break;
                float g2 =
                        gaussian(kwidth - 0.5f,
                                kernelRadius);
                float g3 =
                        gaussian(kwidth + 0.5f,
                                kernelRadius);
                kernel[kwidth] =
                        (g1 + g2 + g3)
                                / 3f
                                / (2f * (float) Math.PI
                                        * kernelRadius * kernelRadius);
                diffKernel[kwidth] = g3 - g2;
        }

        int initX = kwidth - 1;
        int maxX = width - (kwidth - 1);
        int initY = width * (kwidth - 1);
        int maxY =
                width * (height - (kwidth - 1));

        // perform convolution in
        // x and y
        // directions
        for (int x = initX; x < maxX; x++) {
                for (int y = initY; y < maxY; y +=
                        width) {
```

```
                int index = x + y;
                float sumX =
                        data[index] * kernel[0];
                float sumY = sumX;
                int xOffset = 1;
                int yOffset = width;
                for (; xOffset < kwidth;) {
                        sumY +=
                                kernel[xOffset]
                                        * (data[index - yOffset] + data[index
                                                + yOffset]);
                        sumX +=
                                kernel[xOffset]
                                        * (data[index - xOffset] + data[index
                                                + xOffset]);
                        yOffset += width;
                        xOffset++;
                }

                yConv[index] = sumY;
                xConv[index] = sumX;
        }

}

for (int x = initX; x < maxX; x++) {
        for (int y = initY; y < maxY; y +=
                width) {
                float sum = 0f;
                int index = x + y;
                for (int i = 1; i < kwidth; i++)
                        sum +=
                                diffKernel[i]
                                        * (yConv[index - i] - yConv[index
                                                + i]);

                xGradient[index] = sum;
        }

}

for (int x = kwidth; x < width
        - kwidth; x++) {
        for (int y = initY; y < maxY; y +=
                width) {
                float sum = 0.0f;
                int index = x + y;
                int yOffset = width;
                for (int i = 1; i < kwidth; i++) {
                        sum +=
                                diffKernel[i]
                                        * (xConv[index - yOffset] - xConv[index
                                                + yOffset]);
                        yOffset += width;
                }

                yGradient[index] = sum;
        }

}

initX = kwidth;
maxX = width - kwidth;
initY = width * kwidth;
maxY = width * (height - kwidth);
for (int x = initX; x < maxX; x++) {
        for (int y = initY; y < maxY; y +=
                width) {
                int index = x + y;
                int indexN = index - width;
                int indexS = index + width;
                int indexW = index - 1;
                int indexE = index + 1;
                int indexNW = indexN - 1;
                int indexNE = indexN + 1;
                int indexSW = indexS - 1;
                int indexSE = indexS + 1;

                float xGrad = xGradient[index];
                float yGrad = yGradient[index];
                float gradMag =
                        hypot(xGrad, yGrad);

                // perform non-maximal
                // supression
                float nMag =
                        hypot(xGradient[indexN],
                                yGradient[indexN]);
                float sMag =
                        hypot(xGradient[indexS],
                                yGradient[indexS]);
                float wMag =
                        hypot(xGradient[indexW],
                                yGradient[indexW]);
                float eMag =
                        hypot(xGradient[indexE],
                                yGradient[indexE]);
                float neMag =
                        hypot(xGradient[indexNE],
```

```
                                        yGradient[indexNE]);
                        float seMag =
                                hypot(xGradient[indexSE],
                                        yGradient[indexSE]);
                        float swMag =
                                hypot(xGradient[indexSW],
                                        yGradient[indexSW]);
                        float nwMag =
                                hypot(xGradient[indexNW],
                                        yGradient[indexNW]);
                        float tmp;

                        if (xGrad * yGrad <= 0 /*



                        ? Math.abs(xGrad) >= Math
                                .abs(yGrad) /* (2) */
                        ? (tmp =
                                Math.abs(xGrad * gradMag)) >= Math
                                .abs(yGrad * neMag
                                        - (xGrad + yGrad) * eMag) /*



                                && tmp > Math.abs(yGrad
                                        * swMag - (xGrad + yGrad)
                                        * wMag) /*
                                                                        * (4 )
                                                                        */
                                : (tmp =
                                        Math.abs(yGrad * gradMag)) >= Math
                                        .abs(xGrad * neMag
                                                - (yGrad + xGrad) * nMag) /*



                                        && tmp > Math.abs(xGrad
                                                * swMag - (yGrad + xGrad)
                                                * sMag) /*
                                                                                * ( 4 )
                                                                                */
                                        : Math.abs(xGrad) >= Math
                                                .abs(yGrad) /* (2) */
                                        ? (tmp =
                                                Math.abs(xGrad * gradMag)) >= Math
                                                .abs(yGrad * seMag
                                                        + (xGrad - yGrad) * eMag) /*



                                                && tmp > Math.abs(yGrad
                                                        * nwMag + (xGrad - yGrad)
                                                        * wMag) /*
                                                                                        * ( 4 )
                                                                                        */
                                                : (tmp =
                                                        Math.abs(yGrad * gradMag)) >= Math
                                                        .abs(xGrad * seMag
                                                                + (yGrad - xGrad)
                                                                * sMag) /*
                                                                                                * ( 3
                                                                                                * )
                                                                                                */
                                                        && tmp > Math.abs(xGrad
                                                                * nwMag
                                                                + (yGrad - xGrad)
                                                                * nMag) /*
                                                                                                * ( 4
                                                                                                * )
                                                                                                */
                        ) {
                                magnitude[index] =
                                        gradMag >= MAGNITUDE_LIMIT ? MAGNITUDE_MAX
                                                : (int) (MAGNITUDE_SCALE * gradMag);
                        } else {
                                magnitude[index] = 0;
                        }
                }
        }
}

private float hypot(float x, float y) {
        return (float) Math.hypot(x, y);
}

private float gaussian(float x,
        float sigma) {
        return (float) Math.exp(-(x * x)
                / (2f * sigma * sigma));
}

private void performHysteresis(
        int low, int high) {
        Arrays.fill(data, 0);
```

```
        int offset = 0;
        for (int y = 0; y < height; y++) {
                for (int x = 0; x < width; x++) {
                        if (data[offset] == 0
                                && magnitude[offset] >= high) {
                                follow(x, y, offset, low);
                        }
                        offset++;
                }
        }
}

private void follow(int x1, int y1,
        int i1, int threshold) {
        int x0 = x1 == 0 ? x1 : x1 - 1;
        int x2 =
                x1 == width - 1 ? x1 : x1 + 1;
        int y0 = y1 == 0 ? y1 : y1 - 1;
        int y2 =
                y1 == height - 1 ? y1 : y1 + 1;

        data[i1] = magnitude[i1];
        for (int x = x0; x <= x2; x++) {
                for (int y = y0; y <= y2; y++) {
                        int i2 = x + y * width;
                        if ((y != y1 || x != x1)
                                && data[i2] == 0
                                && magnitude[i2] >= threshold) {
                                follow(x, y, i2, threshold);
                                return;
                        }
                }
        }
}

// modification
public int getWidth() {
        return sourceImage.getWidth();
}

// modification
public int getHeight() {
        return sourceImage.getHeight();
}

// modification
public int getCentroidX() {
        return centroidX;
}

// modification
public int getCentroidY() {
        return centroidY;
}

// modification
public int[][] get2DPixels() {
        return pixelArr;
}

// modification
public int[] getData() {
        return data;
}

// modification
public ArrayList<Integer>
        getOutline() {
        return outline;
}

// modification-inverted
// colors
private void thresholdEdges() {
        int sumX = 0;
        int sumY = 0;
        int blackPixelsCount = 0;
        pixelArr = new int[width][height];
        for (int i = 0; i < picsize; i++) {
                data[i] =
                        data[i] > 0 ? 0xff000000 : -1;
                pixelArr[i % width][i / width] =
                        data[i];
                if (data[i] == 0xff000000) {
                        sumX += i % width;
                        sumY += i / width;
                        outline.add(i);
                        ++blackPixelsCount;
                }
        }
        data[114 * 115] = 55;
        centroidX = sumX / blackPixelsCount;
        centroidY = sumY / blackPixelsCount;
}

private int luminance(float r,
        float g, float b) {
        return Math.round(0.299f * r
```

```
                        + 0.587f * g + 0.114f * b);
        }

        public void readLuminance() {
                int type = sourceImage.getType();
                if (type == BufferedImage.TYPE_INT_RGB
                        || type == BufferedImage.TYPE_INT_ARGB) {
                        int[] pixels =
                                (int[]) sourceImage.getData()
                                        .getDataElements(0, 0, width,
                                                height, null);
                        for (int i = 0; i < picsize; i++) {
                                int p = pixels[i];
                                int r = (p & 0xff0000) >> 16;
                                int g = (p & 0xff00) >> 8;
                                int b = p & 0xff;
                                data[i] = luminance(r, g, b);
                        }
                } else if (type == BufferedImage.TYPE_BYTE_GRAY) {
                        byte[] pixels =
                                (byte[]) sourceImage.getData()
                                        .getDataElements(0, 0, width,
                                                height, null);
                        for (int i = 0; i < picsize; i++) {
                                data[i] = (pixels[i] & 0xff);
                        }
                } else if (type == BufferedImage.TYPE_USHORT_GRAY) {
                        short[] pixels =
                                (short[]) sourceImage.getData()
                                        .getDataElements(0, 0, width,
                                                height, null);
                        for (int i = 0; i < picsize; i++) {
                                data[i] =
                                        (pixels[i] & 0xffff) / 256;
                        }
                } else if (type == BufferedImage.TYPE_3BYTE_BGR) {
                        byte[] pixels =
                                (byte[]) sourceImage.getData()
                                        .getDataElements(0, 0, width,
                                                height, null);
                        int offset = 0;
                        for (int i = 0; i < picsize; i++) {
                                int b = pixels[offset++] & 0xff;
                                int g = pixels[offset++] & 0xff;
                                int r = pixels[offset++] & 0xff;
                                data[i] = luminance(r, g, b);
                        }
                } else {
                        throw new IllegalArgumentException(
                                "Unsupported image type: "
                                        + type);
                }
        }

        private void normalizeContrast() {
                int[] histogram = new int[256];
                for (int i = 0; i < data.length; i++) {
                        histogram[data[i]]++;
                }
                int[] remap = new int[256];
                int sum = 0;
                int j = 0;
                for (int i = 0; i < histogram.length; i++) {
                        sum += histogram[i];
                        int target = sum * 255 / picsize;
                        for (int k = j + 1; k <= target; k++) {
                                remap[k] = i;
                        }
                        j = target;
                }

                for (int i = 0; i < data.length; i++) {
                        data[i] = remap[data[i]];
                }
        }

        private void writeEdges(int pixels[]) {
                if (edgesImage == null) {
                        edgesImage =
                                new BufferedImage(width,
                                        height,
                                        BufferedImage.TYPE_INT_ARGB);
                }
                edgesImage.getWritableTile(0, 0)
                        .setDataElements(0, 0, width,
                                height, pixels);
        }
}
```

3. CentroidRadii.java

```
import java.util.ArrayList;

public class CentroidRadii {

        private final int QUADRANT = 4;
        private final int POINTS = 45;
        private ArrayList<Double> radii =
```

```
        new ArrayList<Double>();
private ArrayList<Integer> outline =
        new ArrayList<Integer>();
private double[][] degrees;
private double[][] degRadii;

private int width;
private Double centroidX;
private Double centroidY;

// constructor
public CentroidRadii(int centroidX ,
        int centroidY , int width){
        this.centroidX = (double) centroidX;
        this.centroidY = (double) centroidY;
        this.width = width;
}

// outline, arraylist of
// black pixels
public void setOutline(
        ArrayList<Integer> outline) {
        this.outline = outline;
}

// get Euclidean distance
public double
        getDistance(int x, int y) {
        x -= centroidX;
        y -= centroidY;
        return Math.sqrt(x * x + y * y);
}

// get set of radii
public ArrayList<Double>
        getRadiiSet() {
        return radii;
}

// normalize radii
public void normalizeArray() {
        double longest = 0;
        for (int i = 0; i < QUADRANT; i++) {
                for (int j = 0; j < POINTS; j++) {
                        radii.add(degRadii[i][j]);
                        longest =
                                degRadii[i][j] > longest ? degRadii[i][j]
                                        : longest;
                }
        }
        longest = longest > 0 ? longest : 1;
        for (int i = 0; i < radii.size(); i++) {
                radii.set(i, radii.get(i)
                        / longest);
        }
}

// get quadrant of point
// with respect to
// origin(centroidX,
// centroidY)
public int getQuadrant(int x, int y) {
        if (x < centroidX) {
                if (y < centroidY) {
                        return 2;
                } else if (y >= centroidY) {
                        return 3;
                }
        } else if (x > centroidX) {
                if (y <= centroidY) {
                        return 1;
                } else if (y > centroidY) {
                        return 4;
                }
        } else { // 90 degrees,
                                                // 270 degrees
                if (y < centroidY) {
                        return 2;
                } else if (y > centroidY) {
                        return 4;
                }
        }
        return 0;
}

// initialize 4 x 45 array
public void initDeg() {
        degRadii =
                new double[QUADRANT][POINTS];
        degrees =
                new double[QUADRANT][POINTS];
        for (int i = 0; i < QUADRANT; i++) {
                for (int j = 0; j < POINTS; j++) {
                        degRadii[i][j] = 1.0;
                        degrees[i][j] = 10.0 * (j + 1);
                }
        }
        degrees[1][0] = 100.0;
        // at zero degrees
```

```
                degrees[3][0] = 100.0;
                // at 180 degrees

        }

        // update degrees
        public void updateDeg(double deg,
                int x, int y) {
                int tempDeg = 0;
                tempDeg =
                        deg == 0 ? 0
                                : (int) (deg / 2) * 2;
                int xIndex =
                        getQuadrant(x, y) <= 0 ? 0
                                : getQuadrant(x, y) - 1;
                int yIndex =
                        tempDeg == 90 ? 0 : tempDeg / 2;
                if (tempDeg == 90 && xIndex < 2) {
                        xIndex = 1;
                } else if (tempDeg == 90
                        && xIndex > 1) {
                        xIndex = 3;
                } else if (tempDeg == 0
                        && (xIndex == 0 || x == 3)) {
                        xIndex = 0;
                } else if (tempDeg == 0
                        && (xIndex == 1 || x == 2)) {
                        xIndex = 2;
                }
                double d = degrees[xIndex][yIndex];
                if (Math.abs(tempDeg - deg) <= Math
                        .abs(d - tempDeg)) {
                        degrees[xIndex][yIndex] = deg;
                        degRadii[xIndex][yIndex] =
                                getDistance(x, y);
                }
        }

        // obtain 36 points
        public void determineRadii() {
                initDeg();
                for (int i = 0; i < outline.size(); i++) {
                        int x = outline.get(i) % width;
                        int y = outline.get(i) / width;
                        Double slope =
                                Math.abs((y - centroidY)
                                        / (x - centroidX));
                        Double deg =
                                Math.toDegrees(Math
                                        .atan((slope)));
                        updateDeg(deg, x, y);
                }
                normalizeArray();
        }
}
```

4. ClassifyImage.java

```
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;
import java.sql.Blob;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javax.imageio.ImageIO;
import javax.swing.DefaultListModel;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.SwingConstants;
import javax.swing.UIManager;
import javax.swing.border.EmptyBorder;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.filechooser.FileNameExtensionFilter;
```

```
import javax.swing.table.DefaultTableModel;

import org.neuroph.core.NeuralNetwork;
import org.neuroph.core.learning.DataSet;
import org.neuroph.core.learning.DataSetRow;

@SuppressWarnings("serial")
public class ClassifyImage extends
        JFrame implements
        ListSelectionListener {

        // canny edge configurations
        private float lowThresh = 12f;
        private float highThresh = 15f;
        private float gaussRadius = 2f;
        private int gaussWidth = 16;

        // labels and textfields
        private JPanel panel;
        private JTextField leafMargin;
        private JTextField leafArrangement;
        private JTextField leafPattern;
        private JLabel lblNewLabel =
                new JLabel("");
        private ImageUtils imageUtil =
                new ImageUtils();
        private JTextField leafShape;

        // network
        private int inputCount = 189;
        private int outputCount = 32;

        // results
        private float th = 0;

        // login details
        private static String url =
                "jdbc:mysql://localhost:3306/leaves";
        private static String user = "root";
        private static String pass = "leaves";

        // source image
        @SuppressWarnings("unused")
        private String sourceFileName;
        private Image img;

        // output
        private ArrayList<OutputSpecies> os =
                new ArrayList<OutputSpecies>();

        // table
        private JTable resultsTable =
                new JTable();

        // DefaultModel
        private DefaultTableModel tableModel;
        private DefaultListModel<String> tsListModel =
                new DefaultListModel<String>();

        // image lists
        JScrollPane scrollPane_1 =
                new JScrollPane();
        JLabel imageLabel = new JLabel("");
        DefaultListModel<String> imageListModel;
        JList<String> imageList;
        List<ImageIcon> images =
                new ArrayList<ImageIcon>();

        // list c = collection, s =
        // scroll
        private JList<String> tsCSList =
                new JList<String>(tsListModel);

        // progress bar
        JProgressBar prog;

        /**
         * Create the frame.
         *
         * @param ImageIcon
         *            (image)
         * @throws IOException
         */

        public ClassifyImage(
                final Image image , String url ,
                String user , String pass)
                throws IOException{
                img = image;
                imageUtil
                        .setSourceImage((BufferedImage) image);
                setResizable(false);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 776, 630);
                setLocationRelativeTo(null);
                setTitle("LeaVeS");
                File iconFile =
                        new File(
                                "C:/Users/ACER/workspace/Leaves180/images/icon.png");
```

```
setIconImage(ImageIO.read(iconFile));

JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);

JMenu mnFile = new JMenu("File");
menuBar.add(mnFile);

JMenuItem mntmUploadNewImage =
        new JMenuItem("Upload new image");
mnFile.add(mntmUploadNewImage);

JMenuItem mntmViewEdges =
        new JMenuItem("View edges image");
mnFile.add(mntmViewEdges);

JMenuItem mntmViewVeins =
        new JMenuItem("View veins image");
mnFile.add(mntmViewVeins);

mntmUploadNewImage
        .addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        mntmUploadNewImageActionPerformed(e);

                }

        });

mntmViewEdges
        .addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        CannyEdgeDetector cEdge;
                        try {
                                cEdge = extract(image);
                                cEdge.process();
                                BufferedImage edges =
                                        cEdge.getEdgesImage();
                                Image lEdge =
                                        edges.getScaledInstance(
                                                lblNewLabel.getWidth(),
                                                lblNewLabel.getHeight(),
                                                Image.SCALE_SMOOTH);
                                lblNewLabel
                                        .setIcon(new ImageIcon(
                                                lEdge));
                        } catch (IOException e1) {
                                e1.printStackTrace();
                        }
                }

        });

mntmViewVeins
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        Opening open = new Opening(4);
                        try {
                                imageUtil
                                        .convertToGrayscale();

                                BufferedImage grey =
                                        imageUtil.getGrayImage();
                                BufferedImage opened =
                                        open.execute(grey);
                                BufferedImage veins =
                                        imageUtil.getVeinImage(
                                                grey, opened);

                                Image lVein =
                                        veins.getScaledInstance(
                                                lblNewLabel.getWidth(),
                                                lblNewLabel.getHeight(),
                                                Image.SCALE_SMOOTH);
                                lblNewLabel
                                        .setIcon(new ImageIcon(
                                                lVein));
                        } catch (IOException e2) {
                                e2.printStackTrace();
                        }
                }

        });

JMenuItem mntmClose =
        new JMenuItem("Close");
mnFile.add(mntmClose);

JTabbedPane tabbedPane =
        new JTabbedPane(JTabbedPane.TOP);
tabbedPane
        .setBounds(0, 0, 857, 404);
```

```
getContentPane().add(tabbedPane);

mntmClose
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        System.exit(0);
                }

        });

panel = new JPanel();
panel.setBorder(new EmptyBorder(5,
        5, 5, 5));
panel.setLayout(null);
tabbedPane.addTab("Testing", null,
        panel, null);

lblNewLabel.setBounds(10, 11, 512,
        487);
panel.add(lblNewLabel);
Image img =
        image.getScaledInstance(
                lblNewLabel.getWidth(),
                lblNewLabel.getHeight(),
                Image.SCALE_SMOOTH);
lblNewLabel.setIcon(new ImageIcon(
        img));

JButton btnClassify =
        new JButton("Classify!");
btnClassify
        .addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(
                        ActionEvent e) {

                        try {

                                CannyEdgeDetector canny =
                                        extract(image);
                                getMoments(canny);
                                getRadii(canny);

                                ImageTools tools =
                                        new ImageTools();

                                Opening open =
                                        new Opening(1);
                                Opening open2 =
                                        new Opening(2);
                                Opening open3 =
                                        new Opening(3);
                                Opening open4 =
                                        new Opening(4);
                                imageUtil
                                        .convertToGrayscale();

                                BufferedImage grey =
                                        imageUtil.getGrayImage();
                                BufferedImage opened =
                                        open.execute(grey);
                                ImageIcon veinsImage =
                                        new ImageIcon(imageUtil
                                                .getVeinImage(grey,
                                                        opened));

                                BufferedImage greyBinary =
                                        new BufferedImage(
                                                grey.getWidth(),
                                                grey.getHeight(),
                                                BufferedImage.TYPE_BYTE_BINARY);
                                // Draw the image
                                // on
                                // to the buffered
                                // image
                                Graphics2D bGr =
                                        greyBinary
                                                .createGraphics();
                                bGr.drawImage(grey, 0, 0,
                                        null);
                                bGr.dispose();

                                ImageIcon src =
                                        new ImageIcon(greyBinary);

                                BufferedImage opened2 =
                                        open2.execute(grey);
                                ImageIcon veinsImage2 =
                                        new ImageIcon(imageUtil
                                                .getVeinImage(grey,
                                                        opened2));

                                BufferedImage opened3 =
                                        open3.execute(grey);
                                ImageIcon veinsImage3 =
                                        new ImageIcon(imageUtil
```

```
                        .getVeinImage(grey,
                                opened3));

                BufferedImage opened4 =
                        open4.execute(grey);
                ImageIcon veinsImage4 =
                        new ImageIcon(imageUtil
                                .getVeinImage(grey,
                                        opened4));

                int[] imageForArea =
                        tools.examineInput(src);
                int totalArea =
                        getTotalArea(imageForArea);

                int[] veinsForArea =
                        tools
                                .examineInput(veinsImage);
                int areaVeins1 =
                        getTotalArea(veinsForArea);

                int[] veinsForArea2 =
                        tools
                                .examineInput(veinsImage2);
                int areaVeins2 =
                        getTotalArea(veinsForArea2);

                int[] veinsForArea3 =
                        tools
                                .examineInput(veinsImage3);
                int areaVeins3 =
                        getTotalArea(veinsForArea3);

                int[] veinsForArea4 =
                        tools
                                .examineInput(veinsImage4);
                int areaVeins4 =
                        getTotalArea(veinsForArea4);

                getVeinFeatures(totalArea,
                        areaVeins1, areaVeins2,
                        areaVeins3, areaVeins4);

                // ProjectionHistogram
                // hist = new
                // ProjectionHistogram();
                // int[] hist1D =
                // getHistogram(hist,
                // veinsImage,
                // false);
                // Image img =
                // createImage(new
                // MemoryImageSource(veinsImage
                // .getIconWidth(),
                // veinsImage.getIconHeight(),
                // hist1D, 0,
                // veinsImage.getIconWidth()));
                // BufferedImage
                // histogram = new
                // BufferedImage(img
                // .getWidth(null),
                // img.getHeight(null),
                // BufferedImage.TYPE_INT_RGB);
                //
                // // Draw the
                // image
                // on to the
                // buffered
                // image
                // Graphics2D bGr1
                // =
                // histogram.createGraphics();
                // bGr1.drawImage(img,
                // 0, 0, null);
                // bGr1.dispose();
                //
                // Image histo =
                // histogram.getScaledInstance(
                // lblNewLabel.getWidth(),
                // lblNewLabel.getHeight(),
                // Image.SCALE_SMOOTH);
                // lblNewLabel.setIcon(new
                // ImageIcon(histo));
                recognize();
        } catch (IOException
                | ClassNotFoundException
                | SQLException e1) {
                e1.printStackTrace();
        }
        prog =
                new JProgressBar(0, 100);
        prog.setValue(0);
        prog.setStringPainted(true);

        JOptionPane
                .showMessageDialog(null,
                        "Leaf has been classified.");
        setSize(1151, 630);
        setLocationRelativeTo(null);
```

```
                        revalidate();
                        repaint();

                    }
                });
        btnClassify.setBounds(219, 509, 89,
                23);
        panel.add(btnClassify);

        JPanel panel1 = new JPanel();
        panel1.setBounds(532, 11, 220, 487);
        panel.add(panel1);
        panel1.setLayout(null);

        JLabel lblLeafMargin =
                new JLabel("Leaf Margin");
        lblLeafMargin.setBounds(0, 368,
                110, 94);
        lblLeafMargin
                .setForeground(UIManager
                        .getColor("Button.focus"));
        lblLeafMargin.setFont(new Font(
                "Cordia New", Font.BOLD, 18));
        lblLeafMargin
                .setHorizontalAlignment(SwingConstants.CENTER);
        panel1.add(lblLeafMargin);

        leafMargin = new JTextField();
        leafMargin.setFont(new Font(
                "Tahoma", Font.PLAIN, 14));
        leafMargin.setBounds(110, 369, 110,
                94);
        leafMargin
                .setToolTipText("Enter leaf margin to refine search");
        leafMargin
                .setHorizontalAlignment(SwingConstants.CENTER);
        panel1.add(leafMargin);
        leafMargin.setColumns(10);

        JLabel lblLeafArrangement =
                new JLabel("Leaf Arrangement");
        lblLeafArrangement.setBounds(0,
                241, 110, 94);
        lblLeafArrangement
                .setFont(new Font("Cordia New",
                        Font.BOLD, 18));
        lblLeafArrangement
                .setForeground(UIManager
                        .getColor("Button.focus"));
        lblLeafArrangement
                .setHorizontalAlignment(SwingConstants.CENTER);
        panel1.add(lblLeafArrangement);

        leafArrangement = new JTextField();
        leafArrangement.setBounds(110, 243,
                110, 94);
        leafArrangement.setFont(new Font(
                "Tahoma", Font.PLAIN, 14));
        leafArrangement
                .setHorizontalAlignment(SwingConstants.CENTER);
        leafArrangement
                .setToolTipText("Enter leaf arrangement to refine search");
        panel1.add(leafArrangement);
        leafArrangement.setColumns(10);

        JLabel lblLeafPattern =
                new JLabel("Leaf Pattern");
        lblLeafPattern.setBounds(0, 132,
                110, 94);
        lblLeafPattern
                .setForeground(UIManager
                        .getColor("Button.focus"));
        lblLeafPattern.setFont(new Font(
                "Cordia New", Font.BOLD, 18));
        lblLeafPattern
                .setHorizontalAlignment(SwingConstants.CENTER);
        panel1.add(lblLeafPattern);

        leafPattern = new JTextField();
        leafPattern.setFont(new Font(
                "Tahoma", Font.PLAIN, 14));
        leafPattern.setBounds(110, 133,
                110, 94);
        leafPattern
                .setToolTipText("Enter leaf pattern to refine search");
        leafPattern
                .setHorizontalAlignment(SwingConstants.CENTER);
        panel1.add(leafPattern);
        leafPattern.setColumns(10);

        leafShape = new JTextField();
        leafShape.setFont(new Font(
                "Tahoma", Font.PLAIN, 14));
        leafShape
                .setHorizontalAlignment(SwingConstants.CENTER);
        leafShape.setBounds(110, 11, 110,
                94);
        panel1.add(leafShape);
        leafShape.setColumns(10);
```

```
JLabel lblLeafShape =
        new JLabel("Leaf Shape");
lblLeafShape
        .setHorizontalAlignment(SwingConstants.CENTER);
lblLeafShape.setFont(new Font(
        "Cordia New", Font.BOLD, 18));
lblLeafShape
        .setForeground(Color.BLACK);
lblLeafShape.setBounds(0, 11, 110,
        94);
panel1.add(lblLeafShape);

JScrollPane scrollPane =
        new JScrollPane();
scrollPane.setBounds(793, 11, 486,
        487);
panel.add(scrollPane);

resultsTable.setFont(new Font(
        "Tahoma", Font.ITALIC, 11));
resultsTable
        .setRowSelectionAllowed(false);
resultsTable.setBorder(null);
tableModel =
        new DefaultTableModel(
                new Object[] { "Species",
                        "Match (%)" }, 0);
initTable();
resultsTable.setModel(tableModel);

tsCSList.setFixedCellHeight(30);
tsCSList.setFixedCellWidth(280);
tsCSList
        .setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
tsCSList
        .addListSelectionListener(new ListSelectionListener() {
                @Override
                public void valueChanged(
                        ListSelectionEvent event) {
                        if (tsListModel.size() > 0
                                && tsCSList
                                        .getSelectedValue() != null) {
                                sourceFileName =
                                        "species/"
                                                + tsCSList
                                                        .getSelectedValue()
                                                        .toString();
                        }
                }
        });

scrollPane
        .setViewportView(resultsTable);

JButton btnReduceResults =
        new JButton("Reduce results");
btnReduceResults.setBounds(589,
        509, 112, 23);
panel.add(btnReduceResults);
btnReduceResults
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        try {
                                reduce();
                        } catch (
                                ClassNotFoundException
                                | SQLException e1) {
                                e1.printStackTrace();
                        }
                        setSize(1151, 630);
                        setLocationRelativeTo(null);
                        revalidate();
                        repaint();

                }

        });

JPanel panel_1 = new JPanel();
tabbedPane.addTab("Training Set",
        null, panel_1, null);
panel_1.setLayout(null);

try {
        displayTSet();
} catch (ClassNotFoundException
        | SQLException | IOException e1) {
        e1.printStackTrace();
}

imageList.setFixedCellHeight(30);
imageList.setFixedCellWidth(235);
imageList
        .setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
revalidate();
```

```
        repaint();

        scrollPane_1.setBounds(10, 11, 257,
                523);
        panel_1.add(scrollPane_1);

        imageLabel.setBounds(277, 11, 475,
                523);
        panel_1.add(imageLabel);

}

// display training set
// collection
public void displayTSet()
        throws ClassNotFoundException,
        SQLException, IOException {
        imageListModel =
                new DefaultListModel<String>();
        imageListModel.clear();
        Blob b;
        ArrayList<Leaf> leaves =
                new ArrayList<Leaf>();
        SQL s = new SQL(url, user, pass);
        s.connect();
        leaves = s.queryLeaf();
        for (Leaf l : leaves) {
                b = l.getImg();

                byte[] bytes =
                        b.getBytes(1, (int) b.length());
                BufferedImage bimg =
                        ImageIO
                                .read(new ByteArrayInputStream(
                                        bytes));
                ImageIcon icon =
                        new ImageIcon(bimg);
                imageListModel.addElement(l
                        .getLID()
                        + " : "
                        + s.queryNameByID(l.getID()));
                images.add(icon);
        }
        imageList =
                new JList<String>(imageListModel);
        imageList
                .addListSelectionListener(this);
        scrollPane_1
                .setViewportView(imageList);
        revalidate();
        repaint();

}

public void valueChanged(
        ListSelectionEvent arg0) {
        if (arg0.getSource() == imageList) {
                SQL s = new SQL(url, user, pass);
                try {
                        s.connect();
                        int index =
                                imageList.getSelectedIndex();
                        ImageIcon icon =
                                images.get(index);
                        Image image =
                                icon.getImage()
                                        .getScaledInstance(
                                                imageLabel.getWidth(),
                                                imageLabel.getHeight(),
                                                Image.SCALE_SMOOTH);
                        icon.setImage(image);
                        imageLabel.setIcon(icon);
                } catch (ClassNotFoundException
                        | SQLException e) {
                        e.printStackTrace();
                }
                revalidate();
                repaint();
        }
}

// initialize results tab
public void initTable() {
        for (int i = 0; i < outputCount; i++) {
                tableModel.addRow(new Object[] {
                        "", "" });
        }
}

// initialize Canny Edge
// Detector
protected CannyEdgeDetector extract(
        Image image) throws IOException {
        CannyEdgeDetector canny =
                new CannyEdgeDetector();
        canny.setLowThreshold(lowThresh);
        canny.setHighThreshold(highThresh);
        canny.computeGradients(gaussRadius,
                gaussWidth);
```

```
        canny
                .setSourceImage((BufferedImage) image);
        return canny;
}

// extract moments of edge
// image
public ArrayList<Double> getMoments(
        CannyEdgeDetector canny) {
        canny.process();
        int w = canny.getWidth();
        int h = canny.getHeight();
        int[] data = canny.getData();
        double[][] matrix =
                new double[h][w];

        for (int i = 0; i < h; i++) {
                for (int j = 0; j < w; j++) {
                        matrix[i][j] =
                                data[(i * w) + j];
                }
        }

        ArrayList<Double> m =
                new ArrayList<Double>();
        @SuppressWarnings("unused")
        Moments mom = new Moments();
        m.add(Moments
                .getNormalizedCentralMoment(0, 0,
                        matrix));
        m.add(Moments
                .getNormalizedCentralMoment(0, 1,
                        matrix));
        m.add(Moments
                .getNormalizedCentralMoment(1, 0,
                        matrix));
        m.add(Moments
                .getNormalizedCentralMoment(1, 1,
                        matrix));

        return m;
}

// extract radii of edge
// image
public ArrayList<Double> getRadii(
        CannyEdgeDetector canny) {
        ArrayList<Double> radii =
                new ArrayList<Double>();
        canny.process();
        CentroidRadii cr =
                new CentroidRadii(
                        canny.getCentroidX(),
                        canny.getCentroidY(),
                        canny.getWidth());
        cr.setOutline(canny.getOutline());
        cr.determineRadii();
        radii = cr.getRadiiSet();

        return radii;
}

// extract area of leaf vein
// image
public int getTotalArea(
        int[] veinImage) {
        int area = 0;

        for (int i = 0; i < veinImage.length; i++) {
                if (veinImage[i] != -1) {
                        area++;
                }
        }
        return area;
}

// extract vein features
public ArrayList<Double>
        getVeinFeatures(int totalArea,
                int veinsArea1, int veinsArea2,
                int veinsArea3, int veinsArea4) {
        ArrayList<Double> veinFeatures =
                new ArrayList<Double>();
        veinFeatures
                .add((double) veinsArea1
                        / (double) totalArea);
        veinFeatures
                .add((double) veinsArea2
                        / (double) totalArea);
        veinFeatures
                .add((double) veinsArea3
                        / (double) totalArea);
        veinFeatures
                .add((double) veinsArea4
                        / (double) totalArea);
        veinFeatures
                .add((double) veinsArea4
                        / (double) veinsArea1);
```

```
        return veinFeatures;
}

/**
 *
 * @param hist
 *          - a histogram
 *          instance
 * @param veinsImage
 *          - the leaf image
 * @param horizontal
 *          - orientation of
 *          the projection:
 *          horizontal if
 *          true, vertical
 *          if false
 * @return the projection
 *         histogram of the
 *         image
 */
public int[] getHistogram(
        ProjectionHistogram hist,
        ImageIcon veinsImage,
        boolean horizontal) {
        ImageTools tool = new ImageTools();
        int[] veins1D =
                tool.examineInput(veinsImage);
        int[] hist1D =
                horizontal ? hist
                        .getHorizontalProjection(
                                veins1D,
                                veinsImage.getIconWidth(),
                                veinsImage.getIconHeight())
                        : hist.getVerticalProjection(
                                veins1D,
                                veinsImage.getIconWidth(),
                                veinsImage.getIconHeight());
        return hist1D;
}

protected void
        mntmUploadNewImageActionPerformed(
                ActionEvent e) {
        JOptionPane
                .showMessageDialog(
                        null,
                        "Please upload leaf images of type .jpg only."
                                + "\nThe background of the image should be white."
                                + "\nThe venation of the leaf image should be observable."
                                + "\nImages should be of size 800x600 only.",
                        "Instructions",
                        JOptionPane.WARNING_MESSAGE);
        String dir =
                "D:/Zee/School stuff/SP/Leaves";
        JFileChooser fileChooser =
                new JFileChooser(dir);
        fileChooser
                .setDialogTitle("Open leaf image");
        FileNameExtensionFilter ff =
                new FileNameExtensionFilter(
                        "JPEG file", "jpg");
        fileChooser.setFileFilter(ff);
        fileChooser
                .setAcceptAllFileFilterUsed(false);
        fileChooser.getFileView();
        int returnVal =
                fileChooser.showOpenDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
                File file =
                        fileChooser.getSelectedFile();
                try {
                        Image image =
                                ImageIO.read(file);
                        this.dispose();
                        new ClassifyImage(image, url,
                                user, pass).setVisible(true);
                } catch (IOException ex) {
                        System.out
                                .println("Problem accessing file "
                                        + file.getAbsolutePath());
                }
        }

}

// recognition process:
// extraction of
// features, testing neural
// net
public void recognize()
        throws ClassNotFoundException,
        SQLException, IOException {
        testNeuralNet();
        sortSpecies();
        displayMatch();
}

// reduce list of matches
public void reduce()
```

```
        throws ClassNotFoundException,
        SQLException {
        SQL s = new SQL(url, user, pass);
        s.connect();
        ArrayList<OutputSpecies> reduce =
                new ArrayList<OutputSpecies>();
        for (OutputSpecies spec : os) {
                Species p =
                        s.queryByID(spec.getID());
                boolean match = true;
                // check each field
                if (!(leafShape.getText().equals(
                        "") || leafShape.getText()
                        .equals(null))) {
                        if (!p
                                .getShape()
                                .toLowerCase()
                                .equals(
                                        leafShape.getText()
                                                .toLowerCase()))
                                match = false;
                }
                if (!(leafPattern.getText()
                        .equals("") || leafPattern
                        .getText().equals(null))) {
                        if (!p
                                .getPattern()
                                .toLowerCase()
                                .equals(
                                        leafPattern.getText()
                                                .toLowerCase()))
                                match = false;
                }
                if (!(leafArrangement.getText()
                        .equals("") || leafArrangement
                        .getText().equals(null))) {
                        if (!p
                                .getArrangement()
                                .toLowerCase()
                                .equals(
                                        leafArrangement.getText()
                                                .toLowerCase()))
                                match = false;
                }
                if (!(leafMargin.getText()
                        .equals("") || leafMargin
                        .getText().equals(null))) {
                        if (!p
                                .getMargin()
                                .toLowerCase()
                                .equals(
                                        leafMargin.getText()
                                                .toLowerCase()))
                                match = false;
                }
                if (!match)
                        reduce.add(spec);
        }
        os.removeAll(reduce);
        emptyTable();
        displayMatch();
        s.closeConnection();
}

// load neural net
// configuration and test
public void testNeuralNet()
        throws ClassNotFoundException,
        IOException {

        File f =
                new File("file/Weights.nnet");
        if (f.exists()) {
                ArrayList<Double> feat =
                        new ArrayList<Double>();
                os.removeAll(os);
                double[] in =
                        new double[inputCount];
                int i = 0;

                CannyEdgeDetector c =
                        extract(img);
                feat.addAll(getMoments(c));
                feat.addAll(getRadii(c));

                ImageTools tools =
                        new ImageTools();

                Opening open = new Opening(1);
                Opening open2 = new Opening(2);
                Opening open3 = new Opening(3);
                Opening open4 = new Opening(4);
                imageUtil.convertToGrayscale();

                BufferedImage grey =
                        imageUtil.getGrayImage();
                BufferedImage opened =
                        open.execute(grey);
                ImageIcon veinsImage =
```

```
        new ImageIcon(
                imageUtil.getVeinImage(grey,
                        opened));

BufferedImage greyBinary =
        new BufferedImage(
                grey.getWidth(),
                grey.getHeight(),
                BufferedImage.TYPE_BYTE_BINARY);
// Draw the image on to
// the buffered
// image
Graphics2D bGr =
        greyBinary.createGraphics();
bGr.drawImage(grey, 0, 0, null);
bGr.dispose();

ImageIcon src =
        new ImageIcon(greyBinary);

BufferedImage opened2 =
        open2.execute(grey);
ImageIcon veinsImage2 =
        new ImageIcon(
                imageUtil.getVeinImage(grey,
                        opened2));

BufferedImage opened3 =
        open3.execute(grey);
ImageIcon veinsImage3 =
        new ImageIcon(
                imageUtil.getVeinImage(grey,
                        opened3));

BufferedImage opened4 =
        open4.execute(grey);
ImageIcon veinsImage4 =
        new ImageIcon(
                imageUtil.getVeinImage(grey,
                        opened4));

int[] imageForArea =
        tools.examineInput(src);
int totalArea =
        getTotalArea(imageForArea);

int[] veinsForArea =
        tools.examineInput(veinsImage);
int areaVeins1 =
        getTotalArea(veinsForArea);

int[] veinsForArea2 =
        tools.examineInput(veinsImage2);
int areaVeins2 =
        getTotalArea(veinsForArea2);

int[] veinsForArea3 =
        tools.examineInput(veinsImage3);
int areaVeins3 =
        getTotalArea(veinsForArea3);

int[] veinsForArea4 =
        tools.examineInput(veinsImage4);
int areaVeins4 =
        getTotalArea(veinsForArea4);

feat.addAll(getVeinFeatures(
        totalArea, areaVeins1,
        areaVeins2, areaVeins3,
        areaVeins4));

for (double a : feat) {
        in[i] = a;
        ++i;
}

DataSet testSet =
        new DataSet(inputCount);
testSet.addRow(in);
NeuralNetwork mlp =
        NeuralNetwork
                .load("file/Weights.nnet");

for (DataSetRow testSetRow : testSet
        .getRows()) {
        mlp.setInput(testSetRow
                .getInput());
        mlp.calculate();

        double[] networkOutput =
                mlp.getOutput();
        for (i = 0; i < networkOutput.length; i++) {
                OutputSpecies s =
                        new OutputSpecies();
                s.setID(i + 1);
                s.setOutput(networkOutput[i] * 100);
                os.add(s);
        }
}
```

```
                        reductionProp(true);
                } else {
                        JOptionPane
                                .showMessageDialog(
                                        null,
                                        "Network file not found. Please update trained network file.");
                }
        }

        // sort output species
        public void sortSpecies() {
                for (int i = 0; i < os.size(); i++) {
                        for (int j = 0; j < os.size() - 1; j++) {
                                if (os.get(j).getOutput() < os
                                        .get(j + 1).getOutput()) {
                                        OutputSpecies temp =
                                                os.get(j + 1);
                                        os.set(j + 1, os.get(j));
                                        os.set(j, temp);
                                }
                        }
                }
        }

        // populate results table
        public void displayMatch()
                throws SQLException,
                ClassNotFoundException {
                emptyTable();
                SQL s = new SQL(url, user, pass);
                s.connect();
                for (int i = 0; i < os.size(); i++) {
                        Species spec =
                                s.queryByID(os.get(i).getID());
                        if (os.get(i).getOutput() < th)
                                break;
                        if (tableModel.getRowCount() < os
                                .size()) {
                                tableModel.addRow(new Object[] {
                                        "", "" });
                                outputCount++;
                        }
                        tableModel.setValueAt(
                                spec.getName(), i, 0);
                        tableModel.setValueAt(String
                                .format("%.10f", os.get(i)
                                        .getOutput()), i, 1);
                }
                s.closeConnection();
        }

        // disable/enable reduction
        public void
                reductionProp(boolean flag) {
                leafShape.setEditable(flag);
                leafPattern.setEditable(flag);
                leafArrangement.setEditable(flag);
                leafMargin.setEditable(flag);
        }

        // initialize table
        public void emptyTable() {
                for (int i = 0; i < outputCount; i++) {
                        tableModel.setValueAt("", i, 0);
                        tableModel.setValueAt("", i, 1);
                }
        }
}
```

5. Dilation.java

```
import java.awt.image.BufferedImage;
import java.awt.image.Raster;
import java.awt.image.WritableRaster;

/**
 * Dilation operation for
 * grayscaled images. The
 * dilation operand will for
 * every pixel search for the
 * pixel in a neighborhood
 * around it, i.e. structuring
 * element, with the highest
 * luminace value. The pixel
 * under investigation will
 * then be set to the latter
 * value. The end result will
 * be a dilation of bright
 * structures in the image.
 *
 * @author Tomas
 *
 */
public class Dilation extends
        AbstractOperation {

        @SuppressWarnings("unused")
        private short[][] structElem;
```

```
private STRUCTURING_ELEMENT_SHAPE shape;

public Dilation(){
        shapeSize = 2;
        shape =
                STRUCTURING_ELEMENT_SHAPE.SQUARE;
        this.structElem =
                constructShape(shape, shapeSize);
}

public Dilation(
        STRUCTURING_ELEMENT_SHAPE shape ,
        int shapeSize){
        this.shape = shape;
        super.shapeSize = shapeSize;
        this.structElem =
                constructShape(shape, shapeSize);
}

/**
 * @see AbstractOperation
 */
@Override
public BufferedImage execute(
        BufferedImage img) {
        if (img.getType() != BufferedImage.TYPE_BYTE_GRAY)
                throw new IllegalArgumentException(
                        "The image must be of type TYPE_BYTE_GRAY");
        BufferedImage dilatedImg =
                new BufferedImage(img.getWidth(),
                        img.getHeight(), img.getType());

        int sSize = 2 * shapeSize + 1;
        byte[] window;
        int newValue = 0;

        int imgWidth = img.getWidth();
        int imgHeight = img.getHeight();
        int filterWidth = imgWidth - sSize;
        int filterHeight =
                imgHeight - sSize;
        int lowerSide =
                imgHeight - shapeSize;
        int rightSide =
                imgWidth - shapeSize;
        Raster oldData = img.copyData(null);
        WritableRaster newData =
                dilatedImg.getRaster();

        // Dilate the center of
        // the image, leave
        // dilation near the
        // borders for
        // later
        for (int x = 0; x <= filterWidth; x++) {
                for (int y = 0; y <= filterHeight; y++) {
                        window =
                                (byte[]) oldData
                                        .getDataElements(x, y,
                                                sSize, sSize, null);
                        newValue = max(window);
                        newData.setSample(
                                x + shapeSize, y + shapeSize,
                                0, newValue);
                }
        }

        // Take care of dilation
        // of the left
        // border
        for (int x = 0; x < shapeSize; x++) {
                for (int y = 0; y <= filterHeight; y++) {
                        window =
                                (byte[]) oldData
                                        .getDataElements(0, y,
                                                sSize, sSize, null);
                        newValue = max(window);
                        newData.setSample(x, y
                                + shapeSize, 0, newValue);
                }
        }
        newData.setSamples(
                0,
                lowerSide,
                shapeSize,
                shapeSize,
                0,
                fillArray(shapeSize * shapeSize,
                        newValue));
        window =
                (byte[]) oldData.getDataElements(
                        0, 0, sSize, sSize, null);
        newValue = max(window);
        newData.setSamples(
                0,
                0,
                shapeSize,
                shapeSize,
                0,
```

```
                        fillArray(shapeSize * shapeSize,
                                newValue));

                // Take care of dilation
                // of the right
                // border
                for (int x = rightSide; x < imgWidth; x++) {
                        for (int y = 0; y <= filterHeight; y++) {
                                window =
                                        (byte[]) oldData
                                                .getDataElements(
                                                        filterWidth, y, sSize,
                                                        sSize, null);
                                newValue = max(window);
                                newData.setSample(x, y
                                        + shapeSize, 0, newValue);
                        }
                }
                newData.setSamples(
                        rightSide,
                        lowerSide,
                        shapeSize,
                        shapeSize,
                        0,
                        fillArray(shapeSize * shapeSize,
                                newValue));

                // Take care of dilation
                // of the lower
                // border
                for (int y = lowerSide - 1; y < imgHeight; y++) {
                        for (int x = 0; x <= filterWidth; x++) {
                                window =
                                        (byte[]) oldData
                                                .getDataElements(x,
                                                        filterHeight, sSize,
                                                        sSize, null);
                                newValue = max(window);
                                newData
                                        .setSample(x + shapeSize, y,
                                                0, newValue);
                        }
                }

                // Take care of dilation
                // of the upper
                // border
                for (int y = 0; y < shapeSize; y++) {
                        for (int x = 0; x <= filterWidth; x++) {
                                window =
                                        (byte[]) oldData
                                                .getDataElements(x, 0,
                                                        sSize, sSize, null);
                                newValue = max(window);
                                newData
                                        .setSample(x + shapeSize, y,
                                                0, newValue);
                        }
                }
                newData.setSamples(
                        rightSide,
                        0,
                        shapeSize,
                        shapeSize,
                        0,
                        fillArray(shapeSize * shapeSize,
                                newValue));

                // Dilation is done
                return dilatedImg;
        }

        final private int max(byte[] val) {
                int max = val[0];
                int end = val.length;
                int v = 0;
                for (int i = 1; i < end; i++) {
                        if (val[i] < 0)
                                v = 256 + val[i];
                        else
                                v = val[i];
                        if (v > max)
                                max = v;
                }
                return max;
        }

        final private int[] fillArray(
                int length, int value) {
                int[] arr = new int[length];
                for (int i = 0; i < arr.length; i++) {
                        arr[i] = value;
                }
                return arr;
        }

}
```

6. Erosion.java

```
import java.awt.image.BufferedImage;
import java.awt.image.Raster;
import java.awt.image.WritableRaster;

/**
 * Erosion operation for
 * grayscaled images. The
 * Erosion operand will for
 * every pixel search for the
 * pixel in a neighborhood
 * around it, i.e. structuring
 * element, with the lowest
 * luminace value. The pixel
 * under investigation will
 * then be set to the latter
 * value. The end result will
 * be a erosion of bright
 * structures in the image.
 *
 * @author Tomas
 *
 */
public class Erosion extends
        AbstractOperation {

        @SuppressWarnings("unused")
        private short[][] structElem;
        private STRUCTURING_ELEMENT_SHAPE shape;

        public Erosion(){
                shapeSize = 2;
                shape =
                        STRUCTURING_ELEMENT_SHAPE.SQUARE;
                this.structElem =
                        constructShape(shape, shapeSize);
        }

        public Erosion(
                STRUCTURING_ELEMENT_SHAPE shape ,
                int shapeSize){
                this.shape = shape;
                super.shapeSize = shapeSize;
                this.structElem =
                        constructShape(shape, shapeSize);
        }

        /**
         * @see AbstractOperation
         */
        @Override
        public BufferedImage execute(
                BufferedImage img) {
                if (img.getType() != BufferedImage.TYPE_BYTE_GRAY)
                        throw new IllegalArgumentException(
                                "The image must be of type TYPE_BYTE_GRAY");
                BufferedImage erodedImg =
                        new BufferedImage(img.getWidth(),
                                img.getHeight(), img.getType());

                int sSize = 2 * shapeSize + 1;
                byte[] window = null;
                int newValue = 0;

                int imgWidth = img.getWidth();
                int imgHeight = img.getHeight();
                int filterWidth = imgWidth - sSize;
                int filterHeight =
                        imgHeight - sSize;
                int lowerSide =
                        imgHeight - shapeSize;
                int rightSide =
                        imgWidth - shapeSize;
                Raster oldData = img.copyData(null);
                WritableRaster newData =
                        erodedImg.getRaster();

                // Erode the center of the
                // image, leave
                // dilation near the
                // borders for
                // later
                for (int x = 0; x <= filterWidth; x++) {
                        for (int y = 0; y <= filterHeight; y++) {
                                window =
                                        (byte[]) oldData
                                                .getDataElements(x, y,
                                                        sSize, sSize, null);
                                newValue = min(window);
                                newData.setSample(
                                        x + shapeSize, y + shapeSize,
                                        0, newValue);
                        }
                }

                // Take care of erosion of
                // the left border
                for (int x = 0; x < shapeSize; x++) {
```

```
                for (int y = 0; y <= filterHeight; y++) {
                        window =
                                (byte[]) oldData
                                        .getDataElements(0, y,
                                                sSize, sSize, null);
                        newValue = min(window);
                        newData.setSample(x, y
                                + shapeSize, 0, newValue);
                }
        }
        newData.setSamples(
                0,
                lowerSide,
                shapeSize,
                shapeSize,
                0,
                fillArray(shapeSize * shapeSize,
                        newValue));
        window =
                (byte[]) oldData.getDataElements(
                        0, 0, sSize, sSize, null);
        newValue = min(window);
        newData.setSamples(
                0,
                0,
                shapeSize,
                shapeSize,
                0,
                fillArray(shapeSize * shapeSize,
                        newValue));

        // Take care of erosion of
        // the right
        // border
        for (int x = rightSide; x < imgWidth; x++) {
                for (int y = 0; y <= filterHeight; y++) {
                        window =
                                (byte[]) oldData
                                        .getDataElements(
                                                filterWidth, y, sSize,
                                                sSize, null);
                        newValue = min(window);
                        newData.setSample(x, y
                                + shapeSize, 0, newValue);
                }
        }
        newData.setSamples(
                rightSide,
                lowerSide,
                shapeSize,
                shapeSize,
                0,
                fillArray(shapeSize * shapeSize,
                        newValue));

        // Take care of erosion of
        // the lower
        // border
        for (int y = lowerSide - 1; y < imgHeight; y++) {
                for (int x = 0; x <= filterWidth; x++) {
                        window =
                                (byte[]) oldData
                                        .getDataElements(x,
                                                filterHeight, sSize,
                                                sSize, null);
                        newValue = min(window);
                        newData
                                .setSample(x + shapeSize, y,
                                        0, newValue);
                }
        }

        // Take care of erosion of
        // the upper
        // border
        for (int y = 0; y < shapeSize; y++) {
                for (int x = 0; x <= filterWidth; x++) {
                        window =
                                (byte[]) oldData
                                        .getDataElements(x, 0,
                                                sSize, sSize, null);
                        newValue = min(window);
                        newData
                                .setSample(x + shapeSize, y,
                                        0, newValue);
                }
        }
        newData.setSamples(
                rightSide,
                0,
                shapeSize,
                shapeSize,
                0,
                fillArray(shapeSize * shapeSize,
                        newValue));

        return erodedImg;
}
```

```
        final public int min(byte[] val) {
                int min = 256;
                int end = val.length;
                int v = 0;
                for (int i = 0; i < end; i++) {
                        if (val[i] < 0)
                                v = 256 + val[i];
                        else
                                v = val[i];
                        if (v < min)
                                min = v;
                }
                return min;
        }

        final private int[] fillArray(
                int length, int value) {
                int[] arr = new int[length];
                for (int i = 0; i < arr.length; i++) {
                        arr[i] = value;
                }
                return arr;
        }
}
```

7. ErrorGraph.java

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.ArrayList;

import javax.swing.JPanel;

@SuppressWarnings("serial")
public class ErrorGraph extends JPanel {
        // properties of error graph
        private int width;
        private int height;
        private ArrayList<Double> err;

        // constructor
        public ErrorGraph(int width ,
                int height , ArrayList<Double> err){
                this.width = width;
                this.height = height;
                this.err = err;
        }

        // paint graph
        @Override
        public void
                paintComponent(Graphics g) {
                // draw string labels
                Graphics2D g2D = (Graphics2D) g;
                g2D.drawString("Errors", 10, 50);
                g2D.drawString("Epochs",
                        width - 50, height - 20);
                g.drawLine(70, 40, 70, height - 50);

                // draw horizontal lines
                double in = 0.0;
                for (int y = height - 50; y > 0
                        && in < 1; y -= 20, in += 0.1) {
                        g.drawLine(70, y, width - 10, y);
                        g2D.drawString(
                                String.format("%.1f", in), 50,
                                y);
                }

                // draw numerical labels
                int i = 0;
                for (int x = 70; x < width; x += 50, i++) {
                        g2D.drawString((i * 50) + "", x,
                                height - 35);
                }

                // plot points(errors)
                g2D.setColor(Color.red);
                for (i = 0; i < width - 70
                        && i < err.size(); i++) {
                        int x = i + 70;
                        double y = 250 - err.get(i) * 2;
                        g2D.drawLine(x, (int) y, x,
                                (int) y);
                }

        }
}
```

8. ExpertClassifyImage.java

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.Image;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Blob;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;

import javax.imageio.ImageIO;
import javax.swing.DefaultListModel;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JTextPane;
import javax.swing.JViewport;
import javax.swing.SwingConstants;
import javax.swing.SwingWorker;
import javax.swing.UIManager;
import javax.swing.border.MatteBorder;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.filechooser.FileNameExtensionFilter;

import org.neuroph.core.events.LearningEvent;
import org.neuroph.core.events.LearningEventListener;
import org.neuroph.core.learning.DataSet;
import org.neuroph.core.learning.DataSetRow;
import org.neuroph.core.learning.LearningRule;
import org.neuroph.nnet.MultiLayerPerceptron;
import org.neuroph.nnet.learning.BackPropagation;
import org.neuroph.util.TransferFunctionType;

public class ExpertClassifyImage extends
        JFrame implements
        LearningEventListener,
        ListSelectionListener {

        /**
         * Launch the application.
         */

        private static final long serialVersionUID =
                1L;
        private JFrame frame;

        // network errors
        private ArrayList<Double> errors =
                new ArrayList<Double>();

        // panel for error graph
        JPanel panel_3 = new ErrorGraph(450,
                300, errors);
        private JScrollPane scrollPane_1;

        // textfields for leaf
        // properties
        private JTextField leafNameField;
        private JTextField leafShapeField;
        private JTextField leafPatternField;
        private JTextField leafArrField;
        private JTextField leafMarginField;

        // label where image will be
        // displayed
        private JLabel lblNewLabel =
                new JLabel("");

        // labels for leaf
        // properties
        JLabel lblNewLabel_1 = new JLabel(
                "Leaf Name");
        JLabel lblLeafShape = new JLabel(
                "Leaf Shape");
        JLabel lblLeafPattern = new JLabel(
                "Leaf Pattern");
        JLabel lblLeafArrangement =
                new JLabel("Leaf Arrangement");
        JLabel lblLeafMargin = new JLabel(
                "Leaf Margin");
```

```
// fields for summary
JLabel leavesCount = new JLabel("");
JLabel speciesCount = new JLabel("");
JLabel netError = new JLabel("");
JTextField outputNodes =
        new JTextField();

// edge detector
// configurations
private float lowThresh = 12f;
private float highThresh = 15f;
private float gaussRadius = 2f;
private int gaussWidth = 16;

// vein areas
private int total = 0;
private int veinArea1 = 0;
private int veinArea2 = 0;
private int veinArea3 = 0;
private int veinArea4 = 0;

// source leaf image
private String img;
private Image image;

// blob for leaf image
@SuppressWarnings("unused")
private Blob b;

// vein detector
// configuration
private int rad = 4;

// textfields for network
// configurations
private JTextField inputNodes;
private JTextField hiddenNodes;
private JTextField maxError;
private JTextField learningRate;

// network buttons
JButton btnTrain = new JButton(
        "Train");
JButton btnStopLearning =
        new JButton("Stop Learning");

// network variables
private int epochs = 0;
JLabel epochsCount;

// flag to stop network
// learning
private boolean stopped = false;

// Thread
@SuppressWarnings("rawtypes")
private SwingWorker w;

// network
private MultiLayerPerceptron m = null;

// expected output
private double[] out;

// scroll pane for training
// set
private JScrollPane scrollPane =
        new JScrollPane();

// image lists
DefaultListModel<String> imageListModel;
JList<String> imageList;
JLabel imageLabel;
List<ImageIcon> images =
        new ArrayList<ImageIcon>();

// login details
private static String url =
        "localhost:3306/leaves";
private static String user = "root";
private static String pass = "leaves";

/**
 * Create the application.
 *
 * @throws IOException
 */

public ExpertClassifyImage(
        final String url ,
        final String user ,
        final String pass)
        throws IOException{
        ExpertClassifyImage.url = url;
        ExpertClassifyImage.user = user;
        ExpertClassifyImage.pass = pass;

        setBackground(new Color(0, 0, 0));
```

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setBounds(100, 100, 863, 450);
setLocationRelativeTo(null);
setResizable(false);
setTitle("Expert Tab");
File iconFile =
        new File(
                "C:/Users/ACER/workspace/Leaves180/images/icon.png");
setIconImage(ImageIO.read(iconFile));

SQL s = new SQL(url, user, pass);
try {
        s.connect();
        updateSummary();
} catch (ClassNotFoundException
        | SQLException e2) {
        e2.printStackTrace();
}

try {
        Thread.sleep(5000);
} catch (InterruptedException e) {
        System.out
                .println("An error occurred.");
}

JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);

JMenu mnMenu = new JMenu("Menu");
menuBar.add(mnMenu);

JMenu mnEdgeDetection =
        new JMenu("Edge Detection");
mnMenu.add(mnEdgeDetection);

JMenuItem mntmLowThreshold =
        new JMenuItem("Low Threshold");
mnEdgeDetection
        .add(mntmLowThreshold);
mntmLowThreshold
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        lowThresh =
                                Float
                                        .parseFloat(JOptionPane
                                                .showInputDialog(null,
                                                        "Low Threshold:",
                                                        12.0));
                }

        });

JMenuItem mntmHighThreshold =
        new JMenuItem("High Threshold");
mnEdgeDetection
        .add(mntmHighThreshold);
mntmHighThreshold
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        highThresh =
                                Float
                                        .parseFloat(JOptionPane
                                                .showInputDialog(null,
                                                        "High Threshold:",
                                                        15.0));

                }

        });

JMenuItem mntmDetect =
        new JMenuItem("Detect");
mnEdgeDetection.add(mntmDetect);
mntmDetect
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        CannyEdgeDetector canny =
                                extract();
                        canny.process();
                        BufferedImage edge =
                                canny.getEdgesImage();
                        Image img =
                                edge.getScaledInstance(
                                        lblNewLabel.getWidth(),
                                        lblNewLabel.getHeight(),
                                        Image.SCALE_SMOOTH);
                        lblNewLabel
                                .setIcon(new ImageIcon(img));
                }
```

```
        });

JMenu mnVeinDetection =
        new JMenu("Vein Detection");
mnMenu.add(mnVeinDetection);

JMenuItem mntmOpeningRadius =
        new JMenuItem("Opening Radius");
mnVeinDetection
        .add(mntmOpeningRadius);
mntmOpeningRadius
        .addActionListener(new ActionListener() {

@Override
public void actionPerformed(
        ActionEvent e) {
        rad = Integer.parseInt(JOptionPane
        .showInputDialog( null,
        "Radius of oepning of leaf (please enter a number from 1-4): ",
        4));
                        if (rad < 1 || rad > 4) {
                                JOptionPane
                                        .showMessageDialog(null,
                                                "You must enter a number from 1-4.");
                        }
                }
        });

JMenuItem mntmDetect_1 =
        new JMenuItem("Detect");
mnVeinDetection.add(mntmDetect_1);
mntmDetect_1
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        ImageUtils iu =
                                new ImageUtils();
                        iu.setSourceImage((BufferedImage) image);
                        try {
                                iu.convertToGrayscale();
                        } catch (IOException e1) {
                                e1.printStackTrace();
                        }
                        BufferedImage grayImage =
                                iu.getGrayImage();
                        Opening open =
                                new Opening(rad);
                        BufferedImage opened =
                                open.execute(grayImage);
                        BufferedImage veins =
                                iu.getVeinImage(grayImage,
                                        opened);
                        Image finalVeins =
                                veins.getScaledInstance(
                                        lblNewLabel.getWidth(),
                                        lblNewLabel.getHeight(),
                                        Image.SCALE_SMOOTH);
                        lblNewLabel
                                .setIcon(new ImageIcon(
                                        finalVeins));
                }

        });

JMenu mnNetworkOptions =
        new JMenu("Network Options");
mnMenu.add(mnNetworkOptions);

JMenuItem mntmSave =
        new JMenuItem("Save");
mnNetworkOptions.add(mntmSave);
mntmSave
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        mntmSaveActionPerformed(e);

                }

        });

JMenuItem mntmUpload =
        new JMenuItem("Upload");
mnNetworkOptions.add(mntmUpload);
mntmUpload
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        mntmUploadActionPerformed(e);

                }
```

```
        });

JMenuItem mntmClose =
        new JMenuItem("Close");
mnMenu.add(mntmClose);

getContentPane().setLayout(null);

JTabbedPane tabbedPane =
        new JTabbedPane(
                SwingConstants.TOP);
tabbedPane
        .setBounds(0, 0, 857, 404);
getContentPane().add(tabbedPane);

JPanel panel = new JPanel();
tabbedPane.addTab("Leaf Species",
        null, panel, null);
panel.setLayout(null);
lblNewLabel.setBounds(10, 11, 588,
        349);
panel.add(lblNewLabel);

JPanel panel_1 = new JPanel();
panel_1.setBorder(new MatteBorder(
        2, 2, 2, 2, new Color(0, 0, 0)));
panel_1
        .setBounds(608, 11, 234, 349);
panel.add(panel_1);
panel_1.setLayout(null);

JTextPane txtpnLeafProperties =
        new JTextPane();
txtpnLeafProperties.setBounds(64,
        4, 104, 25);
txtpnLeafProperties
        .setFont(new Font(
                "Century Gothic", Font.BOLD, 14));
txtpnLeafProperties
        .setEditable(false);
txtpnLeafProperties
        .setForeground(UIManager
                .getColor("CheckBox.focus"));
txtpnLeafProperties
        .setBackground(UIManager
                .getColor("Button.background"));
txtpnLeafProperties
        .setText("Leaf Properties");
panel_1.add(txtpnLeafProperties);

lblNewLabel_1
        .setForeground(UIManager
                .getColor("Button.focus"));
lblNewLabel_1.setFont(new Font(
        "Cordia New", Font.PLAIN, 18));
lblNewLabel_1.setBounds(10, 40, 69,
        25);
panel_1.add(lblNewLabel_1);

lblLeafShape
        .setForeground(UIManager
                .getColor("Button.focus"));
lblLeafShape.setFont(new Font(
        "Cordia New", Font.PLAIN, 18));
lblLeafShape.setBounds(10, 78, 69,
        25);
panel_1.add(lblLeafShape);

lblLeafPattern
        .setForeground(UIManager
                .getColor("Button.focus"));
lblLeafPattern.setFont(new Font(
        "Cordia New", Font.PLAIN, 18));
lblLeafPattern.setBounds(10, 113,
        69, 25);
panel_1.add(lblLeafPattern);

lblLeafArrangement
        .setForeground(UIManager
                .getColor("Button.focus"));
lblLeafArrangement
        .setFont(new Font("Cordia New",
                Font.PLAIN, 18));
lblLeafArrangement.setBounds(10,
        149, 109, 25);
panel_1.add(lblLeafArrangement);

lblLeafMargin
        .setForeground(UIManager
                .getColor("Button.focus"));
lblLeafMargin.setFont(new Font(
        "Cordia New", Font.PLAIN, 18));
lblLeafMargin.setBounds(10, 185,
        69, 25);
panel_1.add(lblLeafMargin);

leafNameField = new JTextField();
leafNameField.setBounds(120, 44,
        104, 20);
```

```
panel_1.add(leafNameField);
leafNameField.setColumns(10);

leafShapeField = new JTextField();
leafShapeField.setBounds(120, 82,
        104, 20);
panel_1.add(leafShapeField);
leafShapeField.setColumns(10);

leafPatternField = new JTextField();
leafPatternField.setBounds(120,
        117, 104, 20);
panel_1.add(leafPatternField);
leafPatternField.setColumns(10);

leafArrField = new JTextField();
leafArrField.setBounds(120, 153,
        104, 20);
panel_1.add(leafArrField);
leafArrField.setColumns(10);

leafMarginField = new JTextField();
leafMarginField.setColumns(10);
leafMarginField.setBounds(120, 189,
        104, 20);
panel_1.add(leafMarginField);

JButton btnLoadAnImage =
        new JButton("Load an image");
btnLoadAnImage.setBounds(39, 255,
        169, 23);
panel_1.add(btnLoadAnImage);
btnLoadAnImage
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        btnLoadImageActionPerformed(e);

                }

        });

JButton btnUploadImageAnd =
        new JButton(
                "Upload new leaf image");
btnUploadImageAnd.setBounds(39,
        287, 169, 23);
panel_1.add(btnUploadImageAnd);
btnUploadImageAnd
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        // add record to
                        // leaf
                        // table
                        if (!leafNameField.getText()
                                .equals("")
                                | !leafNameField.getText()
                                        .equals(null)) {
                                int response =
                                        JOptionPane
                                                .showConfirmDialog(
                                                        null,
                                                        "Add existing plant species?",
                                                        "Confirm",
                                                        JOptionPane.YES_NO_OPTION,
                                                        JOptionPane.QUESTION_MESSAGE);
                                if (response == JOptionPane.YES_NO_OPTION) {
                                        try {
                                                addToL();
                                                JOptionPane
                                                        .showMessageDialog(
                                                                null,
                                                                "Leaf record added!");
                                                updateSummary();
                                        } catch (
                                                ClassNotFoundException
                                                | SQLException
                                                | IOException e1) {
                                                JOptionPane
                                                        .showMessageDialog(
                                                                null,
                                                                "An error occurred!");
                                        }
                                }
                        } else {
                                JOptionPane
                                        .showMessageDialog(null,
                                                "Please input species name");
                        }
                }
        });

JButton btnClear =
        new JButton("Clear");
```

```
btnClear
        .setBounds(39, 223, 169, 23);
panel_1.add(btnClear);
btnClear
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        leafNameField.setText("");
                        leafShapeField.setText("");
                        leafPatternField.setText("");
                        leafArrField.setText("");
                        leafMarginField.setText("");
                        lblNewLabel.setIcon(null);
                        img = null;
                }

        });

JButton btnUploadNewSpecies =
        new JButton("Upload new species");
btnUploadNewSpecies.setBounds(39,
        319, 169, 23);
panel_1.add(btnUploadNewSpecies);
btnUploadNewSpecies
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        // add record to
                        // plant
                        // table and leaf
                        // table
                        int response =
                                JOptionPane
                                        .showConfirmDialog(
                                                null,
                                                "Add new plant species?",
                                                "Confirm",
                                                JOptionPane.YES_NO_OPTION,
                                                JOptionPane.QUESTION_MESSAGE);
                        if (response == JOptionPane.YES_NO_OPTION) {
                                try {
                                        addToP();
                                        addToL();
                                        JOptionPane
                                                .showMessageDialog(
                                                        null,
                                                        "Leaf record added!");
                                        updateSummary();
                                } catch (
                                        ClassNotFoundException
                                        | SQLException
                                        | IOException e3) {
                                        JOptionPane
                                                .showMessageDialog(
                                                        null,
                                                        "An error occurred!");
                                        e3.printStackTrace();
                                }
                        }

                }

        });

JPanel panel_2 = new JPanel();
tabbedPane.addTab("Neural Network",
        null, panel_2, null);
panel_2.setLayout(null);

panel_3.setLayout(null);
panel_3.setForeground(Color.BLACK);
panel_3
        .setPreferredSize(new Dimension(
                575, 349));

scrollPane_1 = new JScrollPane();
scrollPane_1
        .setForeground(Color.BLACK);
scrollPane_1.setBounds(10, 11, 575,
        349);
scrollPane_1.add(panel_3);
scrollPane_1
        .setViewportView(panel_3);
panel_2.add(scrollPane_1);

JLabel lblErrorGraph =
        new JLabel("Error Graph");
lblErrorGraph
        .setForeground(UIManager
                .getColor("Button.focus"));
lblErrorGraph.setFont(new Font(
        "Cordia New", Font.BOLD, 23));
panel_3.add(lblErrorGraph);

JPanel panel_4 = new JPanel();
```

```
panel_4.setBorder(new MatteBorder(
        2, 2, 2, 2, new Color(0, 0, 0)));
panel_4.setForeground(UIManager
        .getColor("Button.focus"));
panel_4
        .setBounds(595, 11, 247, 114);
panel_2.add(panel_4);
panel_4.setLayout(null);

JTextArea txtrReferenceDatabaseDetails =
        new JTextArea();
txtrReferenceDatabaseDetails
        .setEditable(false);
txtrReferenceDatabaseDetails
        .setBounds(6, 7, 173, 26);
txtrReferenceDatabaseDetails
        .setForeground(UIManager
                .getColor("Button.focus"));
txtrReferenceDatabaseDetails
        .setBackground(UIManager
                .getColor("Button.light"));
txtrReferenceDatabaseDetails
        .setFont(new Font("Cordia New",
                Font.BOLD, 18));
txtrReferenceDatabaseDetails
        .setText("Database and Network Details");
panel_4
        .add(txtrReferenceDatabaseDetails);

JLabel lblTotalNumberOf =
        new JLabel(
                "Total Number of Leaves:");
lblTotalNumberOf.setBounds(6, 29,
        132, 20);
lblTotalNumberOf.setFont(new Font(
        "Cordia New", Font.BOLD, 16));
lblTotalNumberOf
        .setForeground(UIManager
                .getColor("Button.focus"));
panel_4.add(lblTotalNumberOf);

JLabel lblTotalNumberOf_1 =
        new JLabel(
                "Total Number of Species:");
lblTotalNumberOf_1
        .setForeground(UIManager
                .getColor("Button.focus"));
lblTotalNumberOf_1
        .setFont(new Font("Cordia New",
                Font.BOLD, 16));
lblTotalNumberOf_1.setBounds(6, 48,
        132, 20);
panel_4.add(lblTotalNumberOf_1);

JLabel lblNumberOfEpochs =
        new JLabel("Number of epochs:");
lblNumberOfEpochs.setBounds(6, 65,
        103, 20);
panel_4.add(lblNumberOfEpochs);
lblNumberOfEpochs
        .setForeground(UIManager
                .getColor("Button.focus"));
lblNumberOfEpochs.setFont(new Font(
        "Cordia New", Font.BOLD, 16));

JLabel lblNetworkError =
        new JLabel("Network Error:");
lblNetworkError.setBounds(6, 84,
        90, 20);
panel_4.add(lblNetworkError);
lblNetworkError
        .setForeground(UIManager
                .getColor("Button.focus"));
lblNetworkError.setFont(new Font(
        "Cordia New", Font.BOLD, 16));

leavesCount.setFont(new Font(
        "Cordia New", Font.BOLD, 18));
leavesCount
        .setForeground(Color.BLACK);
leavesCount.setBounds(133, 35, 46,
        14);
panel_4.add(leavesCount);

speciesCount
        .setForeground(Color.BLACK);
speciesCount.setFont(new Font(
        "Cordia New", Font.BOLD, 18));
speciesCount.setBounds(133, 54, 46,
        14);
panel_4.add(speciesCount);

epochsCount = new JLabel("");
epochsCount
        .setForeground(Color.BLACK);
epochsCount.setFont(new Font(
        "Cordia New", Font.BOLD, 18));
epochsCount.setBounds(133, 71, 104,
        14);
```

```
panel_4.add(epochsCount);

netError.setForeground(Color.BLACK);
netError.setFont(new Font(
        "Cordia New", Font.BOLD, 18));
netError
        .setBounds(133, 87, 104, 14);
panel_4.add(netError);

JPanel panel_6 = new JPanel();
panel_6.setBounds(595, 133, 247,
        227);
panel_2.add(panel_6);
panel_6.setBorder(new MatteBorder(
        2, 2, 2, 2, new Color(0, 0, 0)));
panel_6.setLayout(null);

JTextArea txtrNetworkConfiguration =
        new JTextArea();
txtrNetworkConfiguration
        .setBackground(UIManager
                .getColor("Button.light"));
txtrNetworkConfiguration
        .setEditable(false);
txtrNetworkConfiguration
        .setForeground(UIManager
                .getColor("Button.focus"));
txtrNetworkConfiguration
        .setFont(new Font("Cordia New",
                Font.BOLD, 18));
txtrNetworkConfiguration
        .setText("Network Configuration");
txtrNetworkConfiguration.setBounds(
        34, 11, 127, 22);
panel_6
        .add(txtrNetworkConfiguration);

JLabel lblInputNodes =
        new JLabel("Input");
lblInputNodes
        .setForeground(UIManager
                .getColor("Button.focus"));
lblInputNodes.setFont(new Font(
        "Cordia New", Font.BOLD, 16));
lblInputNodes.setBounds(10, 35, 68,
        19);
panel_6.add(lblInputNodes);

JLabel lblHiddenNodes =
        new JLabel("Hidden");
lblHiddenNodes
        .setForeground(UIManager
                .getColor("Button.focus"));
lblHiddenNodes.setFont(new Font(
        "Cordia New", Font.BOLD, 16));
lblHiddenNodes.setBounds(10, 53,
        77, 19);
panel_6.add(lblHiddenNodes);

JLabel lblOutputNodes =
        new JLabel("Output");
lblOutputNodes
        .setForeground(UIManager
                .getColor("Button.focus"));
lblOutputNodes.setFont(new Font(
        "Cordia New", Font.BOLD, 16));
lblOutputNodes.setBounds(10, 70,
        38, 19);
panel_6.add(lblOutputNodes);

JLabel lblLearningRate =
        new JLabel("Learning Rate");
lblLearningRate
        .setForeground(UIManager
                .getColor("Button.focus"));
lblLearningRate.setFont(new Font(
        "Cordia New", Font.BOLD, 16));
lblLearningRate.setBounds(10, 89,
        77, 19);
panel_6.add(lblLearningRate);

JLabel lblMaxError =
        new JLabel("Max Error");
lblMaxError.setForeground(UIManager
        .getColor("Button.focus"));
lblMaxError.setFont(new Font(
        "Cordia New", Font.BOLD, 16));
lblMaxError.setBounds(10, 111, 68,
        14);
panel_6.add(lblMaxError);

btnTrain
        .setBounds(65, 133, 135, 23);
panel_6.add(btnTrain);
btnTrain
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
```

```
                        ActionEvent e) {
                        if (Integer
                                .parseInt(leavesCount
                                        .getText()) > 0) {
                                btnStopLearning
                                        .setEnabled(true);
                                btnTrain.setEnabled(false);
                                try {
                                        trainNetwork();
                                } catch (
                                        ClassNotFoundException
                                        | SQLException e1) {
                                        JOptionPane
                                                .showMessageDialog(
                                                        null,
                                                        "An error occurred!");
                                } catch (IOException e1) {
                                        JOptionPane
                                                .showMessageDialog(
                                                        null,
                                                        "An error occurred!");
                                }
                        } else {
                                JOptionPane
                                        .showMessageDialog(null,
                                                "Cannot train network.");
                        }

                }

        });

btnStopLearning.setBounds(65, 156,
        135, 23);
panel_6.add(btnStopLearning);
btnStopLearning
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        stopped = true;
                        btnTrain.setEnabled(true);
                        btnStopLearning
                                .setEnabled(false);
                }

        });

JButton btnGenerateErrorGraph =
        new JButton(
                "Generate error graph");
btnGenerateErrorGraph.setBounds(65,
        179, 135, 23);
panel_6.add(btnGenerateErrorGraph);
btnGenerateErrorGraph
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        createGraph(errors);
                        revalidate();
                        repaint();
                }

        });

JButton btnResetAndClear =
        new JButton("Reset and Clear");
btnResetAndClear.setBounds(65, 201,
        136, 23);
panel_6.add(btnResetAndClear);
btnResetAndClear
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        hiddenNodes.setText("20");
                        learningRate.setText("0.5");
                        maxError.setText("0.01");
                        createGraph(new ArrayList<Double>());
                        revalidate();
                        repaint();
                }

        });

inputNodes = new JTextField();
inputNodes.setEditable(false);
inputNodes
        .setForeground(Color.BLACK);
inputNodes
        .setHorizontalAlignment(SwingConstants.LEFT);
inputNodes.setText("189");
inputNodes
        .setBounds(93, 35, 86, 20);
panel_6.add(inputNodes);
```

```
inputNodes.setColumns(10);

hiddenNodes = new JTextField();
hiddenNodes
        .setForeground(Color.BLACK);
hiddenNodes
        .setHorizontalAlignment(SwingConstants.LEFT);
hiddenNodes.setText("20");
hiddenNodes.setBounds(93, 53, 86,
        20);
panel_6.add(hiddenNodes);
hiddenNodes.setColumns(10);

outputNodes.setEditable(false);
outputNodes
        .setForeground(Color.BLACK);
outputNodes
        .setHorizontalAlignment(SwingConstants.LEFT);
outputNodes.setBounds(93, 73, 86,
        20);
panel_6.add(outputNodes);
outputNodes.setColumns(10);

maxError = new JTextField();
maxError.setText("0.01");
maxError
        .setHorizontalAlignment(SwingConstants.LEFT);
maxError.setForeground(Color.BLACK);
maxError.setBounds(93, 109, 86, 20);
panel_6.add(maxError);
maxError.setColumns(10);

learningRate = new JTextField();
learningRate
        .setHorizontalAlignment(SwingConstants.LEFT);
learningRate
        .setForeground(Color.BLACK);
learningRate.setText("0.5");
learningRate.setBounds(93, 89, 86,
        20);
panel_6.add(learningRate);
learningRate.setColumns(10);

JPanel panel_5 = new JPanel();
tabbedPane.addTab("Training Set",
        null, panel_5, null);
panel_5.setLayout(null);

try {
        displayTSet();
} catch (ClassNotFoundException
        | SQLException | IOException e1) {
        e1.printStackTrace();
}

scrollPane.setBounds(10, 11, 412,
        308);
panel_5.add(scrollPane);

JButton btnUpdate =
        new JButton("Update");
btnUpdate.setBounds(10, 330, 412,
        35);
panel_5.add(btnUpdate);
btnUpdate
        .addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        try {
                                displayTSet();
                                revalidate();
                                repaint();
                        } catch (
                                ClassNotFoundException
                                | SQLException
                                | IOException e1) {
                                JOptionPane
                                        .showMessageDialog(null,
                                                "An error occurred.");
                        }
                }

        });

imageLabel = new JLabel("");
imageLabel.setBounds(432, 11, 410,
        306);
panel_5.add(imageLabel);

JButton btnDelete =
        new JButton("Delete");
btnDelete.setBounds(430, 330, 412,
        35);
panel_5.add(btnDelete);
btnDelete
        .addActionListener(new ActionListener() {
```

```
                        @Override
                        public void actionPerformed(
                                ActionEvent e) {
                                delete();
                                imageLabel.setIcon(null);
                                JOptionPane
                                        .showMessageDialog(null,
                                                "Leaf has been deleted.");
                        }

                });

        frame = new JFrame();
        frame.setBounds(100, 100, 450, 300);
        frame
                .setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

protected void
        mntmUploadActionPerformed(
                ActionEvent e) {
        int response =
                JOptionPane.showConfirmDialog(
                        frame, "Upload network?",
                        "Confirm",
                        JOptionPane.YES_NO_OPTION,
                        JOptionPane.QUESTION_MESSAGE);
        if (response == JOptionPane.YES_NO_OPTION) {
                try {
                        upload();
                        JOptionPane.showMessageDialog(
                                frame,
                                "Trained network uploaded!");
                } catch (FileNotFoundException
                        | ClassNotFoundException
                        | SQLException e1) {
                        JOptionPane
                                .showMessageDialog(frame,
                                        "Error. File cannot be uploaded.");
                        e1.printStackTrace();
                }
        }

}

protected void
        mntmSaveActionPerformed(
                ActionEvent e) {
        if (m != null) {
                int response =
                        JOptionPane.showConfirmDialog(
                                null, "Save network?",
                                "Confirm",
                                JOptionPane.YES_NO_OPTION,
                                JOptionPane.QUESTION_MESSAGE);
                if (response == JOptionPane.YES_NO_OPTION) {
                        m.save("file/Weights.txt");
                        JOptionPane.showMessageDialog(
                                null,
                                "Trained network saved!");
                }
        } else {
                JOptionPane
                        .showMessageDialog(null,
                                "Network has not yet been trained.");
        }

}

protected CannyEdgeDetector extract() {
        CannyEdgeDetector canny =
                new CannyEdgeDetector();
        canny.setLowThreshold(lowThresh);
        canny.setHighThreshold(highThresh);
        canny.computeGradients(gaussRadius,
                gaussWidth);
        canny
                .setSourceImage((BufferedImage) image);
        return canny;
}

public void loadImage(Image img) {
        Image image =
                img.getScaledInstance(
                        lblNewLabel.getWidth(),
                        lblNewLabel.getHeight(),
                        Image.SCALE_SMOOTH);
        lblNewLabel.setIcon(new ImageIcon(
                image));
        repaint();
        revalidate();
}

public void createGraph(
        ArrayList<Double> err) {

        if (epochs < 400) {
                panel_3 =
                        new ErrorGraph(450, 300, err);
```

```
                panel_3
                        .setPreferredSize(new Dimension(
                                450, 300));
        } else {
                panel_3 =
                        new ErrorGraph(epochs + 150,
                                300, err);
                panel_3
                        .setPreferredSize(new Dimension(
                                epochs + 150, 300));
        }

        panel_3.setForeground(Color.BLACK);

        scrollPane_1
                .setViewportView(panel_3);
        scrollPane_1.getViewport()
                .setScrollMode(
                        JViewport.SIMPLE_SCROLL_MODE);
        JScrollBar bar =
                scrollPane_1
                        .getHorizontalScrollBar();
        bar.setValue(bar.getMaximum());

        epochsCount.setText(Integer
                .toString(epochs));

        scrollPane_1.revalidate();
        scrollPane_1.repaint();

}

// display training set
// collection
public void displayTSet()
        throws ClassNotFoundException,
        SQLException, IOException {
        imageListModel =
                new DefaultListModel<String>();
        imageListModel.clear();
        Blob b;
        ArrayList<Leaf> leaves =
                new ArrayList<Leaf>();
        SQL s = new SQL(url, user, pass);
        s.connect();
        leaves = s.queryLeaf();
        for (Leaf l : leaves) {
                b = l.getImg();

                byte[] bytes =
                        b.getBytes(1, (int) b.length());
                BufferedImage bimg =
                        ImageIO
                                .read(new ByteArrayInputStream(
                                        bytes));
                ImageIcon icon =
                        new ImageIcon(bimg);
                imageListModel.addElement(l
                        .getLID()
                        + " : "
                        + s.queryNameByID(l.getID()));
                images.add(icon);
        }
        imageList =
                new JList<String>(imageListModel);
        imageList
                .addListSelectionListener(this);
        scrollPane
                .setViewportView(imageList);
        revalidate();
        repaint();

}

// add record to plant
public void addToP()
        throws ClassNotFoundException,
        SQLException, IOException {
        Species sp = new Species();
        sp.setName(leafNameField.getText());
        sp.setShape(leafShapeField
                .getText());
        sp.setPattern(leafPatternField
                .getText());
        sp.setArrangement(leafArrField
                .getText());
        sp.setMargin(leafMarginField
                .getText());

        SQL s = new SQL(url, user, pass);
        s.connect();
        s.addToPlant(sp, new File(img));
        JOptionPane.showMessageDialog(null,
                "Species record added!");
        s.closeConnection();
}

// add record to leaf
public void addToL()
```

```
        throws ClassNotFoundException,
        SQLException, IOException {

        SQL s = new SQL(url, user, pass);
        s.connect();
        CannyEdgeDetector c = extract();
        Leaf leaf = new Leaf();
        leaf.setID(s
                .queryByName(leafNameField
                        .getText()));
        leaf.setMoments(getMoments(c));
        leaf.setRadii(getRadii(c));
        getVeins();
        leaf.setVeins(getVeinFeatures(
                total, veinArea1, veinArea2,
                veinArea3, veinArea4));

        if (!s.addToLeaf(leaf,
                new File(img)))
                JOptionPane
                        .showMessageDialog(
                                null,
                                "Plant does not exist yet. Please add as new species.");
        s.closeConnection();
}

// extract moments of edge
// image
public ArrayList<Double> getMoments(
        CannyEdgeDetector canny) {
        canny.process();
        int w = canny.getWidth();
        int h = canny.getHeight();
        int[] data = canny.getData();
        double[][] matrix =
                new double[h][w];

        for (int i = 0; i < h; i++) {
                for (int j = 0; j < w; j++) {
                        matrix[i][j] =
                                data[(i * w) + j];
                }
        }

        ArrayList<Double> m =
                new ArrayList<Double>();
        @SuppressWarnings("unused")
        Moments mom = new Moments();
        m.add(Moments
                .getNormalizedCentralMoment(0, 0,
                        matrix));
        m.add(Moments
                .getNormalizedCentralMoment(0, 1,
                        matrix));
        m.add(Moments
                .getNormalizedCentralMoment(1, 0,
                        matrix));
        m.add(Moments
                .getNormalizedCentralMoment(1, 1,
                        matrix));

        return m;
}

// extract radii of edge
// image
public ArrayList<Double> getRadii(
        CannyEdgeDetector canny) {
        ArrayList<Double> radii =
                new ArrayList<Double>();
        canny.process();
        CentroidRadii cr =
                new CentroidRadii(
                        canny.getCentroidX(),
                        canny.getCentroidY(),
                        canny.getWidth());
        cr.setOutline(canny.getOutline());
        cr.determineRadii();
        radii = cr.getRadiiSet();

        return radii;
}

// get vein image for areas
public void getVeins() {
        ImageUtils iu = new ImageUtils();
        iu.setSourceImage((BufferedImage) image);
        try {
                iu.convertToGrayscale();
        } catch (IOException e1) {
                e1.printStackTrace();
        }
        BufferedImage grayImage =
                iu.getGrayImage();
        ImageTools tools = new ImageTools();
        int[] grayArray =
                tools.examineInput(new ImageIcon(
                        grayImage));
        total = getTotalArea(grayArray);
```

```
        Opening open1 = new Opening(1);
        BufferedImage opened1 =
                open1.execute(grayImage);
        BufferedImage veins1 =
                iu.getVeinImage(grayImage,
                        opened1);
        int[] veins1Array =
                tools.examineInput(new ImageIcon(
                        veins1));
        veinArea1 =
                getTotalArea(veins1Array);

        Opening open2 = new Opening(2);
        BufferedImage opened2 =
                open2.execute(grayImage);
        BufferedImage veins2 =
                iu.getVeinImage(grayImage,
                        opened2);
        int[] veins2Array =
                tools.examineInput(new ImageIcon(
                        veins2));
        veinArea2 =
                getTotalArea(veins2Array);

        Opening open3 = new Opening(3);
        BufferedImage opened3 =
                open3.execute(grayImage);
        BufferedImage veins3 =
                iu.getVeinImage(grayImage,
                        opened3);
        int[] veins3Array =
                tools.examineInput(new ImageIcon(
                        veins3));
        veinArea3 =
                getTotalArea(veins3Array);

        Opening open4 = new Opening(4);
        BufferedImage opened4 =
                open4.execute(grayImage);
        BufferedImage veins4 =
                iu.getVeinImage(grayImage,
                        opened4);
        int[] veins4Array =
                tools.examineInput(new ImageIcon(
                        veins4));
        veinArea4 =
                getTotalArea(veins4Array);

}

// extract area of leaf vein
// image
public int getTotalArea(
        int[] veinImage) {
        int area = 0;

        for (int i = 0; i < veinImage.length; i++) {
                if (veinImage[i] != -1) {
                        area++;
                }
        }
        return area;
}

// extract vein features
public ArrayList<Double>
        getVeinFeatures(int totalArea,
                int veinsArea1, int veinsArea2,
                int veinsArea3, int veinsArea4) {
        ArrayList<Double> veinFeatures =
                new ArrayList<Double>();
        veinFeatures
                .add((double) veinsArea1
                        / (double) totalArea);
        veinFeatures
                .add((double) veinsArea2
                        / (double) totalArea);
        veinFeatures
                .add((double) veinsArea3
                        / (double) totalArea);
        veinFeatures
                .add((double) veinsArea4
                        / (double) totalArea);
        veinFeatures
                .add((double) veinsArea4
                        / (double) veinsArea1);

        return veinFeatures;
}

// get summary
public void updateSummary()
        throws ClassNotFoundException,
        SQLException {
        SQL s = new SQL(url, user, pass);
        s.connect();
        leavesCount.setText(Integer
                .toString(s.getCount("leaf")));
```

```
        speciesCount.setText(Integer
                .toString(s.getCount("plant")));
        outputNodes.setText(Integer
                .toString(s.getCount("plant")));
        s.closeConnection();
}

// train network using back
// propagation
@SuppressWarnings("rawtypes")
public void trainNetwork()
        throws ClassNotFoundException,
        SQLException, IOException {
        ArrayList<Leaf> leaves =
                new ArrayList<Leaf>();
        SQL s = new SQL(url, user, pass);
        s.connect();
        leaves = s.queryLeaf();
        s.closeConnection();
        int i =
                Integer.parseInt(inputNodes
                        .getText());
        int h =
                Integer.parseInt(hiddenNodes
                        .getText());
        int o =
                Integer.parseInt(outputNodes
                        .getText());

        final DataSet trSet =
                new DataSet(i, o);
        errors.removeAll(errors);
        for (Leaf l : leaves) {
                int j = 0;
                double[] in = new double[i];

                for (double a : l.getMoments()) {
                        in[j] = a;
                        ++j;
                }
                for (double a : l.getRadii()) {
                        in[j] = a;
                        ++j;
                }
                for (double a : l.getVeins()) {
                        in[j] = a;
                        ++j;
                }

                clearOutput();
                out[l.getID() - 1] = 1;
                trSet.addRow(new DataSetRow(in,
                        out));
        }

        final MultiLayerPerceptron mlp =
                new MultiLayerPerceptron(
                        TransferFunctionType.SIGMOID,
                        i, h, o);
        LearningRule learningRule =
                mlp.getLearningRule();
        learningRule.addListener(this);
        learningRule.setTrainingSet(trSet);

        w = new SwingWorker() {
                @Override
                protected Object doInBackground()
                        throws Exception {
                        mlp.learn(trSet);
                        return null;
                }
        };
        w.execute();
        m = mlp;
}

// clear output
public void clearOutput() {
        out =
                new double[Integer
                        .parseInt(outputNodes.getText())];
        for (int i = 0; i < Integer
                .parseInt(outputNodes.getText()); i++) {
                out[i] = 0;
        }
}

// upload network file to
// server
public void upload()
        throws FileNotFoundException,
        SQLException,
        ClassNotFoundException {
        SQL s = new SQL(url, user, pass);
        s.connect();
        s.addToNetwork(new File(
                "file/Weights.txt"));
        s.closeConnection();
}
```

```
// delete from list
public void delete() {
        int i =
                imageList.getSelectedIndex();
        String leafNum = "";
        if (imageListModel.size() > 1) {
                if (i != 0)
                        imageList.setSelectedIndex(i);
                else
                        imageList
                                .setSelectedIndex(i - 1);
        }
        StringTokenizer tokens =
                new StringTokenizer(
                        imageList.getSelectedValue(),
                        " ");
        leafNum = tokens.nextToken();

        SQL s = new SQL(url, user, pass);
        try {
                s.connect();
                s.deletelImg(Integer
                        .parseInt(leafNum));
                s.closeConnection();
                displayTSet();
        } catch (ClassNotFoundException
                | SQLException
                | NumberFormatException
                | IOException e1) {
                JOptionPane.showMessageDialog(
                        frame, "An error occurred.");
        }
        if (imageListModel.size() > 0) {
                if (i != 0)
                        imageList
                                .setSelectedIndex(i - 1);
                else
                        imageList.setSelectedIndex(i);
        } else {
                imageLabel.setIcon(null);
                imageLabel
                        .setText("Image Preview");
        }

}

protected void
        btnLoadImageActionPerformed(
                ActionEvent e) {
        JOptionPane
                .showMessageDialog(
                        null,
                        "Please upload leaf images of type .jpg only."
                                + "\nThe background of the image should be white."
                                + "\nThe venation of the leaf image should be observable."
                                + "\nImages should be of size 800x600 only.",
                        "Instructions",
                        JOptionPane.WARNING_MESSAGE);
        String dir =
                "D:/Zee/School stuff/SP/Leaves/Test Leaves";
        JFileChooser fileChooser =
                new JFileChooser(dir);
        fileChooser
                .setDialogTitle("Open leaf image");
        FileNameExtensionFilter ff =
                new FileNameExtensionFilter(
                        "JPEG file", "jpg");
        // FileView fv = new
        // FileView()
        fileChooser.setFileFilter(ff);
        fileChooser
                .setAcceptAllFileFilterUsed(false);
        fileChooser.getFileView();
        int returnVal =
                fileChooser.showOpenDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
                File file =
                        fileChooser.getSelectedFile();
                try {
                        img =
                                file.getAbsolutePath()
                                        .replaceAll("[\\\\]", "/");
                        image = ImageIO.read(file);
                        loadImage(image);
                } catch (IOException ex) {
                        System.out
                                .println("Problem accessing file "
                                        + file.getAbsolutePath());
                }
        }

}

@Override
public void handleLearningEvent(
        LearningEvent event) {
        double lr =
                Double.parseDouble(learningRate
```

```
                        .getText());
                double me =
                        Double.parseDouble(maxError
                                .getText());

                BackPropagation bp =
                        (BackPropagation) event
                                .getSource();
                bp.setLearningRate(lr);
                bp.setMaxError(me);

                errors.add(100 * bp
                        .getTotalNetworkError());
                netError.setText(String.format(
                        "%.10f",
                        bp.getTotalNetworkError()));
                epochsCount.setText(Integer
                        .toString(bp
                                .getCurrentIteration()));

                this.epochs =
                        bp.getCurrentIteration();
                if (stopped) {
                        bp.stopLearning();
                        stopped = false;
                }
                if (bp.getTotalNetworkError() < me) {
                        btnTrain.setEnabled(true);
                        btnStopLearning.setEnabled(false);
                }
                createGraph(errors);
        }

        @Override
        public void valueChanged(
                ListSelectionEvent arg0) {
                if (arg0.getSource() == imageList) {
                        SQL s = new SQL(url, user, pass);
                        try {
                                s.connect();
                                int index =
                                        imageList.getSelectedIndex();
                                ImageIcon icon =
                                        images.get(index);
                                Image image =
                                        icon.getImage()
                                                .getScaledInstance(
                                                        imageLabel.getWidth(),
                                                        imageLabel.getHeight(),
                                                        Image.SCALE_SMOOTH);
                                icon.setImage(image);
                                imageLabel.setIcon(icon);
                        } catch (ClassNotFoundException
                                | SQLException e) {
                                e.printStackTrace();
                        }
                        revalidate();
                        repaint();
                }
        }

        public static void
                main(String[] args) {
                try {
                        @SuppressWarnings("resource")
                        BufferedReader br =
                                new BufferedReader(
                                        new FileReader("file/SQL.txt"));
                        String line = br.readLine();
                        StringTokenizer tokens =
                                new StringTokenizer(line, ";");
                        url = tokens.nextToken();
                        user = tokens.nextToken();
                        pass = tokens.nextToken();
                } catch (IOException io) {
                        JOptionPane
                                .showMessageDialog(
                                        null,
                                        "File containing connection details does not exist.");
                }
                EventQueue
                        .invokeLater(new Runnable() {
                                @Override
                                public void run() {
                                        try {
                                                UIManager.setLookAndFeel(UIManager
                                                        .getSystemLookAndFeelClassName());
                                                ExpertClassifyImage frame =
                                                        new ExpertClassifyImage(
                                                                "jdbc:mysql://" + url,
                                                                user, pass);
                                                frame.setVisible(true);
                                        } catch (Exception e) {
                                                e.printStackTrace();
                                        }
                                }
                        });
        }
}
```

9. IconListRenderer.java

```
import java.awt.Component;
import java.util.Map;

import javax.swing.DefaultListCellRenderer;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JList;

public class IconListRenderer extends
        DefaultListCellRenderer {

        private static final long serialVersionUID =
                -3776230699705222244L;
        private Map<String, ImageIcon> images =
                null;

        public IconListRenderer(
                Map<String, ImageIcon> images){
                this.images = images;
        }

        @SuppressWarnings("rawtypes")
        public Component
                getListCellRendererComponent(
                        JList list, String key,
                        int index, boolean isSelected,
                        boolean cellHasFocus) {
                JLabel tImg =
                        (JLabel) super
                                .getListCellRendererComponent(
                                        list, key, index, isSelected,
                                        cellHasFocus);

                System.out.println(key);
                ImageIcon icon = images.get(key);

                tImg.setIcon(icon);
                return tImg;
        }

}
```

10. ImageDifference.java

```
/**
 * *ImageDifference is an
 * algorithm to find the
 * difference between two
 * images *@author:Judy
 * Robertson, SELLIC OnLine T
 * *@author Timothy Sharman
 * *@see code.iface.imagediff
 */

public class ImageDifference extends
        Thread {
        // the width of the input
        // images in pixels
        private int i1_w;
        private int i2_w;
        // the width and height of
        // the output image
        private int d_w;
        private int d_h;
        private int[] dest_1d;

        // private boolean
        // is_colored;

        /**
         * *Constructs a new Image
         * Difference *@param
         * firstwidth The width of
         * the first image *@param
         * secondwidth The width of
         * the second image
         */

        public ImageDifference(
                int firstwidth , int secondwidth){
                i1_w = firstwidth;
                i2_w = secondwidth;
        }

        /**
         * *Applies the image
         * difference operator on
         * the specified image
         * arrays, with the
         * specified offset and
         * scale value *@param
         * src1_1d The first source
         * image as a pixel array
         * *@param src2_1d The
         * second source image as a
         * pixel array *@param width
```

```
 * width of the destination
 * image in pixels *@param
 * height height of the
 * destination image in
 * pixels *@param oset The
 * offset value *@param
 * scale The scale value *@param
 * reverse false if image2 -
 * image 1 *@return A pixel
 * array containing the
 * difference of the two
 * input images
 */

// Bob's image difference
// algorithm..
/**
 * a) assume the image is
 * grey level (hence
 * RR=GG=BB) b) use value
 * &0x000000ff to get the BB
 * value c) do this for both
 * input images d) apply the
 * operation (eg subtract)
 * e) add 128 and then scale
 * f) clip to lie from 0 to
 * 255. Call this value 0xCD
 * g) create int value
 * 0xffCDCDCD
 */

public int[] imagedifference(
        int[] src1_1d, int[] src2_1d,
        int width, int height, float oset,
        float scale, boolean reverse) {
        int place1 = -1;
        int place2 = -1;
        int src1rgb = 0;
        int src2rgb = 0;
        int diff = 0;
        // Get size of image and
        // make 1d_arrays
        d_w = width;
        d_h = height;
        dest_1d = new int[d_w * d_h];
        @SuppressWarnings("unused")
        boolean firstwider = false;
        boolean secondwider = false;
        int wrap;
        if (i1_w > d_w) {
                wrap = ((i1_w + 1) - d_w);
                firstwider = true;
        } else if (i2_w > d_w) {
                wrap = ((i2_w + 1) - d_w);
                secondwider = true;
        } else {
                wrap = 0;
        } // if you know there is
                // no wrap around,
                // you can save yourself
                // some
                // time
        if (wrap == 0) {
                for (int i = 0; i < dest_1d.length; i++) {
                        src2rgb =
                                src2_1d[i] & 0x000000ff;
                        src1rgb =
                                src1_1d[i] & 0x000000ff;
                        if (!reverse) {
                                diff =
                                        (int) (scale * (((float) src2rgb - (float) src1rgb) + oset));
                        } else {
                                diff =
                                        (int) (scale * (((float) src1rgb - (float) src2rgb) + oset));
                        } // clip to 0 ... 256
                        if (diff <= 0) {
                                diff = 0;
                        } else if (diff >= 255) {
                                diff = 255;
                        } // create an int
                                // value for
                                // dest_1d
                        dest_1d[i] =
                                0xff000000 | (diff
                                        + (diff << 16) + (diff << 8));
                }
                return dest_1d;
        } else {
                for (int i = 0; i < dest_1d.length; i++) {
                        // we might need to
                        // skip out some
                        // pixels which aren't
                        // in the
                        // overlap area
                        if ((i % d_w) == 0) {
                                if (i == 0) {
                                        place1 = 0;
                                        place2 = 0;
                                } else if (secondwider) {
```

```
                                        place2 = place2 + wrap;
                                        place1++;
                                } else {
                                        place1 = place1 + wrap;
                                        place2++;
                                }
                        } else {
                                place2++;
                                place1++;
                        }
                        src2rgb =
                                src2_1d[place2] & 0x000000ff;
                        src1rgb =
                                src1_1d[place1] & 0x000000ff;
                        if (!reverse) {
                                diff =
                                        (int) ((scale * ((float) src2rgb - (float) src1rgb)) + oset);
                        } else {
                                diff =
                                        (int) ((scale * ((float) src1rgb - (float) src2rgb)) + oset);
                        } // clip to 0 ... 256
                        if (diff < 0) {
                                diff = 0;
                        } else if (diff > 255) {
                                diff = 255;
                        } // create an int
                                // value for
                                // dest_1d
                        dest_1d[i] =
                                0xff000000 | (diff
                                        + (diff << 16) + (diff << 8));
                }
                return dest_1d;
        }
}

/**
 * *Subtracts the specified
 * constant value from the
 * specified input image,
 * also using offset and
 * scale values *@param
 * src1_1d The input pixel
 * array *@param width width
 * of the destination image
 * in pixels *@param height
 * height of the destination
 * image in pixels *@param
 * oset The offset value
 * *@param scale The scale
 * value *@param constant
 * The constant value to be
 * subtracted from every
 * pixel in the input array
 * *@return A pixel array
 * with the constant value
 * subtracted from every
 * pixel in the input array
 */

public int[] imagedifference(
        int[] src1_1d, int width,
        int height, float oset,
        float scale, int constant) {
        // Get size of image and
        // make 1d_arrays
        d_w = width;
        d_h = height;
        dest_1d = new int[d_w * d_h];
        int src1rgb;
        int diff = 0;
        // now do the threshold on
        // the input image

        for (int i = 0; i < src1_1d.length; i++) {
                src1rgb = src1_1d[i] & 0x000000ff;
                // constant = constant &
                // 0x000000ff;
                diff =
                        (int) ((scale * ((float) src1rgb - (float) constant)) + oset);
                if (diff > 255) {
                        diff = 255;
                }
                if (diff < 0) {
                        diff = 0;
                }
                dest_1d[i] =
                        0xff000000 | (diff
                                + (diff << 16) + (diff << 8));
        }
        return dest_1d;
}

/**
 * *Subtracts every pixel in
 * the input image from the
 * specified constant value,
 * also using offset and
 * scale values * *@param
```

```
             * constant The constant
             * value to be subtracted
             * from every pixel in the
             * input array *@param width
             * width of the destination
             * image in pixels *@param
             * height height of the
             * destination image in
             * pixels *@param oset The
             * offset value *@param
             * scale The scale value
             * *@param src1_1d The input
             * pixel array *@return A
             * pixel array with the
             * difference between the
             * constant value and every
             * pixel in the input array
             */

            public int[] imagedifference(
                    int constant, int width,
                    int height, float oset,
                    float scale, int[] src1_1d) {
                    // Get size of image and
                    // make 1d_arrays
                    d_w = width;
                    d_h = height;
                    dest_1d = new int[d_w * d_h];
                    int src1rgb;
                    int diff = 0;

                    // now do the threshold on
                    // the input image
                    for (int i = 0; i < src1_1d.length; i++) {
                            src1rgb = src1_1d[i] & 0x000000ff;
                            // constant = constant &
                            // 0x000000ff;
                            diff =
                                    (int) ((scale * ((float) constant - (float) src1rgb)) + oset);
                            if (diff > 255) {
                                    diff = 255;
                            }
                            if (diff < 0) {
                                    diff = 0;
                            }
                            dest_1d[i] =
                                    0xff000000 | (diff
                                            + (diff << 16) + (diff << 8));
                    }
                    return dest_1d;
            }

    }
```

11. ImageTools.java

```
    import java.awt.Image;
    import java.awt.image.PixelGrabber;

    import javax.swing.ImageIcon;

    /**
     * * This class implements
     * methods to apply to images.
     */
    public class ImageTools {

            /**
             * * Scales an image into
             * 256*256. * @param icon
             * the ImageIcon
             * representing the image. * @return
             * the ImageIcon
             * representing the scaled
             * image.
             */

            public ImageIcon scaleImage(
                    ImageIcon icon) {
                    Image image, image2;
                    ImageIcon icon2;
                    image = icon.getImage();
                    image2 =
                            image.getScaledInstance(256, 256,
                                    1);
                    icon2 = new ImageIcon(image2);
                    return icon2;
            }

            /**
             * * Returns the width of
             * the image
             */
            public int getWidth(ImageIcon icon) {
                    int width;
                    width = icon.getIconWidth();
                    return width;
            }
```

```
        /**
         * * Returns the height of
         * the image.
         */
        public int getHeight(ImageIcon icon) {
                int height;
                height = icon.getIconHeight();
                return height;
        }

        /**
         * * Converts an imageIcon
         * into a 1D array. * @param
         * the imageIcon
         * representing the image * @return
         * the 1D int array
         * representing the image.
         */
        public int[] examineInput(
                ImageIcon inputIcon) {
                int inputArray[];
                int width;
                int height;
                width = getWidth(inputIcon);
                height = getHeight(inputIcon);
                inputArray =
                        new int[height * width];
                PixelGrabber pg =
                        new PixelGrabber(
                                inputIcon.getImage(), 0, 0,
                                width, height, inputArray, 0,
                                width);
                try {
                        pg.grabPixels();
                } catch (InterruptedException e3) {
                        System.out.println("ERROR");
                }
                ;
                return inputArray;
        }
}
```

12. ImageUtils.java

```
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.awt.image.ColorConvertOp;
import java.awt.image.MemoryImageSource;
import java.io.IOException;

import javax.swing.ImageIcon;
import javax.swing.JPanel;

public class ImageUtils {

        private BufferedImage sourceImage;
        private BufferedImage grayImage;
        private BufferedImage gray;

        public BufferedImage getSourceImage() {
                return sourceImage;
        }

        public void setSourceImage(
                BufferedImage image) {
                sourceImage = image;

        }

        public void convertToGrayscale()
                throws IOException {
                // create a grayscale
                // image the same size
                gray =
                        new BufferedImage(
                                sourceImage.getWidth(),
                                sourceImage.getHeight(),
                                BufferedImage.TYPE_BYTE_GRAY);
                // convert the original
                // colored image to
                // grayscale
                ColorConvertOp op =
                        new ColorConvertOp(sourceImage
                                .getColorModel()
                                .getColorSpace(), gray
                                .getColorModel()
                                .getColorSpace(), null);
                op.filter(sourceImage, gray);
                setGrayImage();
        }

        private void setGrayImage() {
                grayImage = gray;

        }
```

```
        public BufferedImage getGrayImage() {
                return grayImage;
        }

        public BufferedImage getVeinImage(
                BufferedImage grey,
                BufferedImage opened) {
                JPanel imageCreator = new JPanel();
                ImageDifference imgDiff =
                        new ImageDifference(
                                grey.getWidth(),
                                opened.getWidth());
                ImageTools tools = new ImageTools();

                int width = grey.getWidth();
                int height = grey.getHeight();

                ImageIcon greyIcon =
                        new ImageIcon(grey);
                ImageIcon openIcon =
                        new ImageIcon(opened);

                int[] grey1D =
                        tools.examineInput(greyIcon);
                int[] open1D =
                        tools.examineInput(openIcon);

                int[] final1D =
                        imgDiff.imagedifference(grey1D,
                                open1D, width, height, 143f,
                                1f, false);

                Image img =
                        imageCreator
                                .createImage(new MemoryImageSource(
                                        greyIcon.getIconWidth(),
                                        greyIcon.getIconHeight(),
                                        final1D, 0, greyIcon
                                                .getIconWidth()));
                BufferedImage difference =
                        new BufferedImage(
                                img.getWidth(null),
                                img.getHeight(null),
                                BufferedImage.TYPE_INT_RGB);

                // Draw the image on to
                // the buffered image
                Graphics2D bGr =
                        difference.createGraphics();
                bGr.drawImage(img, 0, 0, null);
                bGr.dispose();

                BufferedImage veins =
                        new BufferedImage(
                                img.getWidth(null),
                                img.getHeight(null),
                                BufferedImage.TYPE_BYTE_BINARY);

                // Draw the image on to
                // the buffered image
                Graphics2D bGr1 =
                        veins.createGraphics();
                bGr1.drawImage(img, 0, 0, null);
                bGr1.dispose();

                return veins;
        }

}
```

13. Leaf.java

```
import java.sql.Blob;
import java.util.ArrayList;

public class Leaf {
        // parameters
        private int plantID;
        private int leafID;
        private ArrayList<Double> moments;
        private ArrayList<Double> radii;
        private ArrayList<Double> veins;
        private Blob blob;

        // getters
        public int getLID() {
                return leafID;
        }

        public int getID() {
                return plantID;
        }

        public ArrayList<Double> getMoments() {
                return moments;
        }

        public ArrayList<Double> getRadii() {
```

```
                return radii;
        }

        public ArrayList<Double> getVeins() {
                return veins;
        }

        public Blob getImg() {
                return blob;
        }

        // setters
        public void setLID(int leafID) {
                this.leafID = leafID;
        }

        public void setID(int plantID) {
                this.plantID = plantID;
        }

        public void setMoments(
                ArrayList<Double> moments) {
                this.moments = moments;
        }

        public void setRadii(
                ArrayList<Double> radii) {
                this.radii = radii;
        }

        public void setVeins(
                ArrayList<Double> veins) {
                this.veins = veins;
        }

        public void setImg(Blob blob) {
                this.blob = blob;
        }

}
```

14. LeaVeS.java

```
package com.leaves;

import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.Image;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.CopyOption;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.sql.SQLException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JProgressBar;
import javax.swing.JTextField;
import javax.swing.UIManager;
import javax.swing.border.EmptyBorder;
import javax.swing.border.MatteBorder;
import javax.swing.filechooser.FileNameExtensionFilter;

@SuppressWarnings({ "serial" })
public class LeaVeS extends JFrame {

        private JProgressBar prog =
                new JProgressBar();
        private JPanel contentPane;
        private JTextField txtUsername;

        private static String url =
                "jdbc:mysql://localhost:3306/leaves";
        private static String username =
                "root";
        private static String password =
                "leaves";
        private JPasswordField passwordField;

        /**
         * Launch the application.
         *
         * @throws IOException
         */
```

```
public static void
        main(String[] args)
                throws IOException {
        final Splash splash = new Splash();
        EventQueue
                .invokeLater(new Runnable() {
                        @Override
                        public void run() {
                                try {
                                        UIManager.setLookAndFeel(UIManager
                                                .getSystemLookAndFeelClassName());
                                        LeaVeS frame =
                                                new LeaVeS(url, username,
                                                        password);
                                        splash.setVisible(false);
                                        frame.setVisible(true);
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                });
}

/**
 * Create the frame.
 *
 * @throws IOException
 */
public LeaVeS(final String url ,
        final String username ,
        final String password)
        throws IOException{
        LeaVeS.url = url;
        LeaVeS.username = username;
        LeaVeS.password = password;

        setForeground(UIManager
                .getColor("Button.focus"));
        setFont(new Font("Cordia New",
                Font.BOLD, 23));
        setResizable(false);
        setBackground(new Color(0, 0, 0));
        setTitle("LeaVeS");
        File iconFile =
                new File(
                        "C:/Users/ACER/workspace/Leaves180/images/icon.png");
        setIconImage(ImageIO.read(iconFile));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 615, 445);
        setLocationRelativeTo(null);
        contentPane = new JPanel();
        contentPane
                .setBorder(new EmptyBorder(5, 5,
                        5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JPanel panel_3 = new JPanel();
        panel_3.setBounds(26, 11, 552, 246);
        contentPane.add(panel_3);
        panel_3.setLayout(null);

        JLabel lblLeafClassificationApplication =
                new JLabel(
                        "Plant Leaf Recognition by Venation and Shape");
        lblLeafClassificationApplication
                .setBounds(new Rectangle(77, 65,
                        420, 26));
        panel_3
                .add(lblLeafClassificationApplication);
        lblLeafClassificationApplication
                .setFont(new Font("Cordia New",
                        Font.BOLD, 30));
        lblLeafClassificationApplication
                .setForeground(UIManager
                        .getColor("Button.focus"));
        lblLeafClassificationApplication
                .setBackground(Color.PINK);

        JLabel lblUsingArtificialNeural =
                new JLabel(
                        "using Artificial Neural Networks");
        lblUsingArtificialNeural.setBounds(
                152, 100, 279, 26);
        panel_3
                .add(lblUsingArtificialNeural);
        lblUsingArtificialNeural
                .setFont(new Font("Cordia New",
                        Font.BOLD, 30));
        lblUsingArtificialNeural
                .setForeground(UIManager
                        .getColor("Button.focus"));
        lblUsingArtificialNeural
                .setBackground(Color.PINK);

        JLabel lblLeaves =
                new JLabel("LeaVeS");
        lblLeaves.setFont(new Font(
                "Cordia New", Font.BOLD, 36));
```

```
lblLeaves.setBounds(229, 133, 92,
        48);
panel_3.add(lblLeaves);

JLabel lblNewLabel = new JLabel("");
lblNewLabel.setBounds(0, 0, 550,
        246);
lblNewLabel
        .setIcon(new ImageIcon(
                "C:/Users/ACER/workspace/Leaves180/images/background.png"));
panel_3.add(lblNewLabel);

JLabel lblcLeavesCopyright =
        new JLabel(
                "(c) LeaVeS Copyright 2014 Azeil Louisse Codizar");
lblcLeavesCopyright.setBounds(10,
        392, 231, 14);
contentPane
        .add(lblcLeavesCopyright);
lblcLeavesCopyright
        .setForeground(Color.WHITE);
lblcLeavesCopyright
        .setFont(new Font(
                "Century Gothic", Font.BOLD, 10));

JPanel panel_1 = new JPanel();
panel_1.setBorder(new MatteBorder(
        2, 2, 2, 2, new Color(0, 0, 0)));
panel_1.setBackground(UIManager
        .getColor("Button.light"));
panel_1
        .setBounds(26, 268, 296, 113);
contentPane.add(panel_1);
panel_1.setLayout(null);

JLabel lblMorphologyExpert =
        new JLabel("Morphology Expert");
lblMorphologyExpert.setBounds(81,
        6, 126, 16);
panel_1.add(lblMorphologyExpert);
lblMorphologyExpert
        .setForeground(UIManager
                .getColor("Button.focus"));
lblMorphologyExpert
        .setFont(new Font("Cordia New",
                Font.BOLD, 22));

JLabel lblUsername =
        new JLabel("Username:");
lblUsername.setFont(new Font(
        "Cordia New", Font.BOLD, 18));
lblUsername.setForeground(UIManager
        .getColor("Button.focus"));
lblUsername.setBounds(20, 33, 64,
        14);
panel_1.add(lblUsername);

txtUsername = new JTextField();
txtUsername.setBounds(94, 33, 173,
        20);
panel_1.add(txtUsername);
txtUsername.setColumns(15);

JLabel lblPassword =
        new JLabel("Password:");
lblPassword.setForeground(UIManager
        .getColor("Button.focus"));
lblPassword.setFont(new Font(
        "Cordia New", Font.BOLD, 18));
lblPassword.setBounds(20, 60, 61,
        14);
panel_1.add(lblPassword);

passwordField =
        new JPasswordField();
passwordField.setBounds(94, 58,
        173, 20);
panel_1.add(passwordField);

JButton btnLogIn =
        new JButton("Log in");
btnLogIn.setBounds(119, 79, 61, 23);
panel_1.add(btnLogIn);
btnLogIn
        .addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(
                        ActionEvent e) {
                        try {
                                btnLogInActionPerformed(e);
                        } catch (IOException e1) {
                                e1.printStackTrace();
                        }
                }

        });

JPanel panel_2 = new JPanel();
panel_2.setBorder(new MatteBorder(
```

```
                2, 2, 2, 2, new Color(0, 0, 0)));
        panel_2.setBackground(UIManager
                .getColor("Button.light"));
        panel_2.setBounds(332, 268, 246,
                113);
        contentPane.add(panel_2);
        panel_2.setLayout(null);

        JButton btnUploadNewImage =
                new JButton("Upload new image");
        btnUploadNewImage.setBounds(57, 18,
                130, 33);
        panel_2.add(btnUploadNewImage);
        btnUploadNewImage
                .addActionListener(new ActionListener() {
                        @Override
                        public
                                void
                                actionPerformed(
                                        java.awt.event.ActionEvent evt) {
                                btnUploadNewImageActionPerformed(evt);
                        }
                });

        btnUploadNewImage
                .setForeground(Color.BLACK);
        btnUploadNewImage
                .setBackground(Color.PINK);

        JButton btnUpdateTrainingSet =
                new JButton("Update database");
        btnUpdateTrainingSet.setBounds(57,
                64, 130, 33);
        panel_2.add(btnUpdateTrainingSet);
        btnUpdateTrainingSet
                .addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(
                                ActionEvent e) {
                                btnUpdateTrainingSetActionPerformed(e);
                        }
                });
        btnUpdateTrainingSet
                .setForeground(Color.BLACK);
        btnUpdateTrainingSet
                .setBackground(Color.PINK);

        prog.setString("Downloading");
        prog.setStringPainted(true);
        prog.setValue(0);
}

protected void
        btnLogInActionPerformed(
                ActionEvent e) throws IOException {
        username = txtUsername.getText();
        password =
                String.copyValueOf(passwordField
                        .getPassword());

        if (url
                .equals("jdbc:mysql://localhost:3306/leaves")
                && username.equals("expert")
                && password
                        .equals("leavesexpert")) {
                this.dispose();
                new ExpertClassifyImage(url,
                        username, password)
                        .setVisible(true);
        }

}

public void
        btnUploadNewImageActionPerformed(
                java.awt.event.ActionEvent evt) {
        JOptionPane
                .showMessageDialog(
                        contentPane,
                        "Please upload leaf images of type .jpg only."
                                + "\nThe background of the image should be white."
                                + "\nThe venation of the leaf image should be observable."
                                + "\nImages should be of size 800x600 only.",
                        "Instructions",
                        JOptionPane.WARNING_MESSAGE);
        String dir =
                "D:/Zee/School stuff/SP/Leaves";
        JFileChooser fileChooser =
                new JFileChooser(dir);
        fileChooser
                .setDialogTitle("Open leaf image");
        FileNameExtensionFilter ff =
                new FileNameExtensionFilter(
                        "JPEG file", "jpg");
        fileChooser.setFileFilter(ff);
        fileChooser
                .setAcceptAllFileFilterUsed(false);
        fileChooser.getFileView();
        int returnVal =
```

```
                        fileChooser.showOpenDialog(this);
                if (returnVal == JFileChooser.APPROVE_OPTION) {
                        File file =
                                fileChooser.getSelectedFile();
                        try {
                                Image image =
                                        ImageIO.read(file);
                                this.dispose();
                                new ClassifyImage(image, url,
                                        username, password)
                                        .setVisible(true);
                        } catch (IOException ex) {
                                System.out
                                        .println("Problem accessing file "
                                                + file.getAbsolutePath());
                        }
                }
        }

        public
                void
                btnUpdateTrainingSetActionPerformed(
                        java.awt.event.ActionEvent e) {
                try {
                        JOptionPane.showMessageDialog(
                                contentPane,
                                "Please wait. Downloading ...");
                        downloadNetworkFile();
                } catch (ClassNotFoundException
                        | SQLException | IOException e1) {
                        e1.printStackTrace();
                }
        }

        // query from database,
        // download trained
        // network file
        public void downloadNetworkFile()
                throws ClassNotFoundException,
                SQLException, IOException {
                SQL s =
                        new SQL(url, username, password);
                s.connect();
                try {
                        File f = s.queryNetwork();
                        CopyOption[] options =
                                new CopyOption[] {
                                        StandardCopyOption.REPLACE_EXISTING,
                                        StandardCopyOption.COPY_ATTRIBUTES };
                        String fname = f.toString();
                        Files.copy(Paths.get(fname),
                                Paths.get("file/Weights.nnet"),
                                options);
                        JOptionPane.showMessageDialog(
                                null, "Update successful.");

                } catch (NullPointerException
                        | FileNotFoundException e) {
                        JOptionPane.showMessageDialog(
                                null, "Update unsuccessful.");
                }
                s.closeConnection();
        }
}
```

15. Moments.java

```
import java.util.logging.Logger;

/**
 * Image moments
 *
 * @author Arlington
 * @author Saulo
 *         (scsm@ecomp.poli
 *         .br)
 *         <p>
 *         {@link http
 *         ://en.wikipedia
 *         .org/
 *         wiki/Image_moment}
 */

public class Moments {

        private static Logger logger = Logger
                .getLogger(Moments.class.getName());

        public static double getRawMoment(
                int p, int q, double[][] matrix) {
                double m = 0;
                logger
                        .finest("Calculating raw moment for p="
                                + p + " and q=" + q);
                for (int i = 0, k = matrix.length; i < k; i++) {
                        for (int j = 0, l =
                                matrix[i].length; j < l; j++) {
                                m +=
```

```
                                        Math.pow(i, p)
                                                * Math.pow(j, q)
                                                * matrix[i][j];
                        }
                }
                return m;
        }

        public static double
                getCentralMoment(int p, int q,
                        double[][] img) {
                double mc = 0;
                double m00 =
                        Moments.getRawMoment(0, 0, img);
                double m10 =
                        Moments.getRawMoment(1, 0, img);
                double m01 =
                        Moments.getRawMoment(0, 1, img);
                double x0 = m10 / m00;
                double y0 = m01 / m00;
                for (int i = 0, k = img.length; i < k; i++) {
                        for (int j = 0, l = img[i].length; j < l; j++) {
                                mc +=
                                        Math.pow((i - x0), p)
                                                * Math.pow((j - y0), q)
                                                * img[i][j];
                        }
                }
                return mc;
        }

        public static double getCovarianceXY(
                int p, int q, double[][] matrix) {
                double mc00 =
                        Moments.getCentralMoment(0, 0,
                                matrix);
                double mc11 =
                        Moments.getCentralMoment(1, 1,
                                matrix);
                return mc11 / mc00;
        }

        /**
         * Returns the variance in
         * x-direction
         *
         * @param p
         * @param q
         * @param matrix
         *            containing pixel
         *            map for one
         *            layer
         * @return
         */
        public static double getVarianceX(
                int p, int q, double[][] matrix) {
                double mc00 =
                        Moments.getCentralMoment(0, 0,
                                matrix);
                double mc20 =
                        Moments.getCentralMoment(2, 0,
                                matrix);
                return mc20 / mc00;
        }

        /**
         * Returns the variance in
         * y-direction
         *
         * @param p
         * @param q
         * @param matrix
         *            containing pixel
         *            map for one
         *            layer
         * @return
         */
        public static double getVarianceY(
                int p, int q, double[][] matrix) {
                double mc00 =
                        Moments.getCentralMoment(0, 0,
                                matrix);
                double mc02 =
                        Moments.getCentralMoment(0, 2,
                                matrix);
                return mc02 / mc00;
        }

        /**
         * Normalized Central Moment
         *
         * @param p
         * @param q
         * @param matrix
         *            the pixel map
         * @return Normalized
         *          Central Moment
         *          n_pq
         */
```

```
public static double
        getNormalizedCentralMoment(int p,
                int q, double[][] matrix) {
        double gama = ((p + q) / 2) + 1;
        double mpq =
                Moments.getCentralMoment(p, q,
                        matrix);
        double m00gama =
                Math.pow(
                        Moments.getCentralMoment(0, 0,
                                matrix), gama);
        return mpq / m00gama;
}

/**
 * Hu invariant moments
 *
 * @param matrix
 *          the pixel map
 * @param n
 *          the Hu moment
 *          horder
 * @return n-order Hu moment
 */
public static double getHuMoment(
        double[][] matrix, int n) {
        double result = 0.0;

        double n20 =
                Moments
                        .getNormalizedCentralMoment(2,
                                0, matrix), n02 =
                Moments
                        .getNormalizedCentralMoment(0,
                                2, matrix), n30 =
                Moments
                        .getNormalizedCentralMoment(3,
                                0, matrix), n12 =
                Moments
                        .getNormalizedCentralMoment(1,
                                2, matrix), n21 =
                Moments
                        .getNormalizedCentralMoment(2,
                                1, matrix), n03 =
                Moments
                        .getNormalizedCentralMoment(0,
                                3, matrix), n11 =
                Moments
                        .getNormalizedCentralMoment(1,
                                1, matrix);

        switch (n) {
                case 1:
                        result = n20 + n02;
                        break;
                case 2:
                        result =
                                Math.pow((n20 - 02), 2)
                                        + Math.pow(2 * n11, 2);
                        break;
                case 3:
                        result =
                                Math
                                        .pow(n30 - (3 * (n12)), 2)
                                        + Math.pow((3 * n21 - n03),
                                                2);
                        break;
                case 4:
                        result =
                                Math.pow((n30 + n12), 2)
                                        + Math.pow((n12 + n03), 2);
                        break;
                case 5:
                        result =
                                (n30 - 3 * n12)
                                        * (n30 + n12)
                                        * (Math.pow((n30 + n12), 2) - 3 * Math
                                                .pow((n21 + n03), 2))
                                        + (3 * n21 - n03)
                                        * (n21 + n03)
                                        * (3 * Math.pow(
                                                (n30 + n12), 2) - Math
                                                .pow((n21 + n03), 2));
                        break;
                case 6:
                        result =
                                (n20 - n02)
                                        * (Math.pow((n30 + n12), 2) - Math
                                                .pow((n21 + n03), 2)) + 4
                                        * n11 * (n30 + n12)
                                        * (n21 + n03);
                        break;
                case 7:
                        result =
                                (3 * n21 - n03)
                                        * (n30 + n12)
                                        * (Math.pow((n30 + n12), 2) - 3 * Math
                                                .pow((n21 + n03), 2))
                                        + (n30 - 3 * n12)
```

```
                                                        * (n21 + n03)
                                                        * (3 * Math.pow(
                                                                (n30 + n12), 2) - Math
                                                                .pow((n21 + n03), 2));
                                        break;

                                default:
                                        throw new IllegalArgumentException(
                                                "Invalid number for Hu moment.");
                        }
                        return result;
                }
        }
```

16. MorphologicalOperation.java

```
import java.awt.image.BufferedImage;

/**
 * Interface for Morphological
 * operations. A morphological
 * operation should be able to
 * be applied to a
 * {@link BufferedImage} via
 * the
 * {@link MorphologicalOperation #execute(BufferedImage)}
 * .
 *
 * @author Tomas Toss 16 maj
 *         2011
 *
 */
public interface MorphologicalOperation {
        /**
         * The different shapes of a
         * structuring element
         */
        public enum STRUCTURING_ELEMENT_SHAPE {
                SQUARE, VERTICAL_LINE, HORIZONTAL_LINE, CIRCLE
        }

        public BufferedImage execute(
                BufferedImage img);

        public int getShapeSize();
}
```

17. Opening.java

```
import java.awt.image.BufferedImage;

/**
 * Opening operation for
 * grayscaled images. The
 * opening operand is the
 * {@link Dilation} of the
 * {@link Erosion}.
 *
 * The opening operation will
 * remove small "objects",
 * i.e. brighter parts, in the
 * image. This operation can
 * for instance be useful to
 * remove uninteresting
 * structures yielding better
 * accuracy when performing
 * edge detection.
 *
 * @author Tomas
 *
 */
public class Opening extends
        AbstractOperation {

        private BufferedImage sourceImage;

        public void setSourceImage(
                BufferedImage img) {
                sourceImage = img;
        }

        public BufferedImage getSourceImage() {
                return sourceImage;
        }

        private STRUCTURING_ELEMENT_SHAPE shape;

        public Opening(){
                shapeSize = 2;
                shape =
                        STRUCTURING_ELEMENT_SHAPE.CIRCLE;
        }

        // Codizar
        public Opening(int shapeSize){
                super.shapeSize = shapeSize;
                shape =
```

```
                        STRUCTURING_ELEMENT_SHAPE.CIRCLE;
        }

        public Opening(
                STRUCTURING_ELEMENT_SHAPE shape ,
                int shapeSize){
                this.shape = shape;
                super.shapeSize = shapeSize;
        }

        /**
         * @see AbstractOperation
         */
        @Override
        public BufferedImage execute(
                BufferedImage img) {
                if (img.getType() != BufferedImage.TYPE_BYTE_GRAY)
                        throw new IllegalArgumentException(
                                "The image must be of type TYPE_BYTE_GRAY");

                BufferedImage erodedImg, openedImg;
                Dilation dilation =
                        new Dilation(shape, shapeSize);
                Erosion erosion =
                        new Erosion(shape, shapeSize);

                erodedImg = erosion.execute(img);
                openedImg =
                        dilation.execute(erodedImg);

                return openedImg;
        }

}
```

18. OutputSpecies.java

```
public class OutputSpecies {
        private int ID;
        private double outputValue;

        public void setID(int ID) {
                this.ID = ID;
        }

        public void setOutput(
                double outputValue) {
                this.outputValue = outputValue;
        }

        public int getID() {
                return ID;
        }

        public double getOutput() {
                return outputValue;
        }

}
```

19. ProjectionHistogram.java

```
import java.util.Arrays;

/**
 * A class which can generate
 * horizontal and vertical
 * projections of an image.
 * The image must first be
 * converted to pixel array
 * and is assumed to be
 * binary.
 *
 * @author Kim Isle Cortez
 *
 */
public class ProjectionHistogram extends
        Thread {

        /**
         * Gets the horizontal
         * projection of the image
         * (image assumed binary)
         *
         * @param src
         *          - the 1D pixel
         *          array of the
         *          image
         * @param width
         *          - width of the
         *          image
         * @param height
         *          - height of the
         *          image
         * @return 1D pixel array
         *         representing
         *         horizontal the
```

```
     *          projection of the
     *          image
     */
    public int[] getHorizontalProjection(
            int[] src, int width, int height) {
            int[] hist1D = new int[src.length];
            int[][] hist2D =
                    new int[height][width];

            int row = 0, col = 0;
            for (int i = 0; i < src.length; i++) {
                    hist2D[row][col] = src[i];
                    col++;
                    if ((i + 1) % width == 0) {
                            col = 0;
                            row++;
                    }
            }

            for (int i = 0; i < hist2D.length; i++) {
                    Arrays.sort(hist2D[i]);
            }

            int index = 0;
            for (int i = 0; i < hist2D.length; i++) {
                    for (int j = 0; j < hist2D[i].length; j++) {
                            hist1D[index] = hist2D[i][j];
                            index++;
                    }
            }

            return hist1D;
    }

    /**
     * Gets the vertical
     * projection of the image
     * (image assumed binary)
     *
     * @param src
     *          - the 1D pixel
     *          array of the
     *          image
     * @param width
     *          - width of the
     *          image
     * @param height
     *          - height of the
     *          image
     * @return 1D pixel array
     *          representing
     *          vertical the
     *          projection of the
     *          image
     */
    public int[] getVerticalProjection(
            int[] src, int width, int height) {
            int[] hist1D = new int[src.length];
            int[][] hist2D =
                    new int[width][height];

            int row = 0, col = 0;
            for (int i = 0; i < src.length; i++) {
                    hist2D[row][col] = src[i];
                    row++;
                    if ((i + 1) % width == 0) {
                            row = 0;
                            col++;
                    }
            }

            for (int i = 0; i < hist2D.length; i++) {
                    Arrays.sort(hist2D[i]);
            }

            int index = 0;
            for (int i = hist2D[0].length - 1; i >= 0; i--) {
                    for (int j = 0; j < hist2D.length; j++) {
                            hist1D[index] = hist2D[j][i];
                            index++;
                    }
            }

            return hist1D;
    }

}
```

20. Species.java

```
import java.sql.Blob;

public class Species {
        // plant properties
        private int plantID;
        private Blob img;
        private String name;
```

```
        // leaf properties
        private String leafShape;
        private String leafPattern;
        private String leafArrangement;
        private String leafMargin;

        // leaf record
        private Leaf leaf;

        // getters
        public int getID() {
                return plantID;
        }

        public Blob getBlob() {
                return img;
        }

        public String getName() {
                return name;
        }

        public String getShape() {
                return leafShape;
        }

        public String getPattern() {
                return leafPattern;
        }

        public String getArrangement() {
                return leafArrangement;
        }

        public String getMargin() {
                return leafMargin;
        }

        public Leaf getLeaf() {
                return leaf;
        }

        // setters
        public void setID(int plantID) {
                this.plantID = plantID;
        }

        public void setImg(Blob img) {
                this.img = img;
        }

        public void setName(String name) {
                this.name = name;
        }

        public void
                setShape(String leafShape) {
                this.leafShape = leafShape;
        }

        public void setPattern(
                String leafPattern) {
                this.leafPattern = leafPattern;
        }

        public void setArrangement(
                String leafArrangement) {
                this.leafArrangement =
                        leafArrangement;
        }

        public void setMargin(
                String leafMargin) {
                this.leafMargin = leafMargin;
        }

        public void setLeaf(Leaf leaf) {
                this.leaf = leaf;
        }
}
```

21. Splash.java

```
package com.leaves;
import java.awt.Component;
import java.io.IOException;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

public class Splash extends JFrame {

        /**
         *
         */
```

```
        private static final long serialVersionUID =
                -2322807926670906301L;
        private JPanel contentPane;

        public Splash() throws IOException {
                setSize(475, 300);
                contentPane = new JPanel();
                contentPane.setBorder(null);
                setContentPane(contentPane);
                contentPane.setLayout(null);

                JLabel label = new JLabel("Developed by Azeil Louisse Codizar, 2014. All rights reserved.");
                label.setAlignmentX(Component.CENTER_ALIGNMENT);
                label.setHorizontalTextPosition(SwingConstants.CENTER);
                label.setHorizontalAlignment(SwingConstants.CENTER);
                label.setBounds(0, 275, 500, 14);
                contentPane.add(label);

                JLabel lblNewLabel = new JLabel("");

                lblNewLabel.setIcon(new ImageIcon(getClass().getResource("images/splash.png")));
                lblNewLabel.setBounds(0, 0, 475, 300);
                contentPane.add(lblNewLabel);
                setUndecorated(true);
                setLocationRelativeTo(null);
                setVisible(true);
        }
}
```

22. SQL.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.sql.Blob;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.StringTokenizer;

import javax.swing.JOptionPane;

public class SQL {
        // moments = 4, radii = 180,
        // veins = 5
        private final int mCount = 4;
        private final int rCount = 180;
        private final int vCount = 5;

        // connection parameters
        private String url =
                "localhost:3306/leaves";
        private String user = "root";
        private String pass = "leaves";
        private Connection conn;

        // constructor
        public SQL(String url , String user ,
                String pass){
                this.url = url;
                this.user = user;
                this.pass = pass;
        }

        // connect to database
        public void connect()
                throws ClassNotFoundException,
                SQLException {
                Class
                        .forName("com.mysql.jdbc.Driver");
                conn =
                        DriverManager.getConnection(url,
                                user, pass);
        }

        // create plant table
        public void createPlantTable()
                throws SQLException {
                Statement sment =
                        conn.createStatement();

                String pTable =
                        "CREATE TABLE IF NOT EXISTS plant("
                                + "plantID int NOT NULL AUTO_INCREMENT,"
                                + "img longblob NOT NULL,"
                                + "leafName varchar(100) NOT NULL,"
                                + "leafShape varchar(30) NOT NULL,"
                                + "leafPattern varchar(30) NOT NULL,"
                                + "leafArrangement varchar(30) NOT NULL,"
```

```
                        + "leafMargin varchar(30) NOT NULL,"
                        + "PRIMARY KEY (plantID))";
        sment.executeUpdate(pTable);
        sment.close();
}

// create leaf table
public void createLeafTable()
        throws SQLException {
        Statement sment =
                conn.createStatement();

        String lTable =
                "CREATE TABLE IF NOT EXISTS leaf("
                        + "leafID int NOT NULL AUTO_INCREMENT, plantID int NOT NULL,"
                        + "moment1 double NOT NULL, moment2 double NOT NULL,"
                        + "moment3 double NOT NULL, moment4 double NOT NULL,"
                        + "radius1 double NOT NULL, radius2 double NOT NULL,"
                        + "radius3 double NOT NULL, radius4 double NOT NULL,"
                        + "radius5 double NOT NULL, radius6 double NOT NULL,"
                        + "radius7 double NOT NULL, radius8 double NOT NULL,"
                        + "radius9 double NOT NULL, radius10 double NOT NULL,"
                        + "radius11 double NOT NULL, radius12 double NOT NULL,"
                        + "radius13 double NOT NULL, radius14 double NOT NULL,"
                        + "radius15 double NOT NULL, radius16 double NOT NULL,"
                        + "radius17 double NOT NULL, radius18 double NOT NULL,"
                        + "radius19 double NOT NULL, radius20 double NOT NULL,"
                        + "radius21 double NOT NULL, radius22 double NOT NULL,"
                        + "radius23 double NOT NULL, radius24 double NOT NULL,"
                        + "radius25 double NOT NULL, radius26 double NOT NULL,"
                        + "radius27 double NOT NULL, radius28 double NOT NULL,"
                        + "radius29 double NOT NULL, radius30 double NOT NULL,"
                        + "radius31 double NOT NULL, radius32 double NOT NULL,"
                        + "radius33 double NOT NULL, radius34 double NOT NULL,"
                        + "radius35 double NOT NULL, radius36 double NOT NULL,"
                        + "radius37 double NOT NULL, radius38 double NOT NULL,"
                        + "radius39 double NOT NULL, radius40 double NOT NULL,"
                        + "radius41 double NOT NULL, radius42 double NOT NULL,"
                        + "radius43 double NOT NULL, radius44 double NOT NULL,"
                        + "radius45 double NOT NULL, radius46 double NOT NULL,"
                        + "radius47 double NOT NULL, radius48 double NOT NULL,"
                        + "radius49 double NOT NULL, radius50 double NOT NULL,"
                        + "radius51 double NOT NULL, radius52 double NOT NULL,"
                        + "radius53 double NOT NULL, radius54 double NOT NULL,"
                        + "radius55 double NOT NULL, radius56 double NOT NULL,"
                        + "radius57 double NOT NULL, radius58 double NOT NULL,"
                        + "radius59 double NOT NULL, radius60 double NOT NULL,"
                        + "radius61 double NOT NULL, radius62 double NOT NULL,"
                        + "radius63 double NOT NULL, radius64 double NOT NULL,"
                        + "radius65 double NOT NULL, radius66 double NOT NULL,"
                        + "radius67 double NOT NULL, radius68 double NOT NULL,"
                        + "radius69 double NOT NULL, radius70 double NOT NULL,"
                        + "radius71 double NOT NULL, radius72 double NOT NULL,"
                        + "radius73 double NOT NULL, radius74 double NOT NULL,"
                        + "radius75 double NOT NULL, radius76 double NOT NULL,"
                        + "radius77 double NOT NULL, radius78 double NOT NULL,"
                        + "radius79 double NOT NULL, radius80 double NOT NULL,"
                        + "radius81 double NOT NULL, radius82 double NOT NULL,"
                        + "radius83 double NOT NULL, radius84 double NOT NULL,"
                        + "radius85 double NOT NULL, radius86 double NOT NULL,"
                        + "radius87 double NOT NULL, radius88 double NOT NULL,"
                        + "radius89 double NOT NULL, radius90 double NOT NULL,"
                        + "radius91 double NOT NULL, radius92 double NOT NULL,"
                        + "radius93 double NOT NULL, radius94 double NOT NULL,"
                        + "radius95 double NOT NULL, radius96 double NOT NULL,"
                        + "radius97 double NOT NULL, radius98 double NOT NULL,"
                        + "radius99 double NOT NULL, radius100 double NOT NULL,"
                        + "radius101 double NOT NULL, radius102 double NOT NULL,"
                        + "radius103 double NOT NULL, radius104 double NOT NULL,"
                        + "radius105 double NOT NULL, radius106 double NOT NULL,"
                        + "radius107 double NOT NULL, radius108 double NOT NULL,"
                        + "radius109 double NOT NULL, radius110 double NOT NULL,"
                        + "radius111 double NOT NULL, radius112 double NOT NULL,"
                        + "radius113 double NOT NULL, radius114 double NOT NULL,"
                        + "radius115 double NOT NULL, radius116 double NOT NULL,"
                        + "radius117 double NOT NULL, radius118 double NOT NULL,"
                        + "radius119 double NOT NULL, radius120 double NOT NULL,"
                        + "radius121 double NOT NULL, radius122 double NOT NULL,"
                        + "radius123 double NOT NULL, radius124 double NOT NULL,"
                        + "radius125 double NOT NULL, radius126 double NOT NULL,"
                        + "radius127 double NOT NULL, radius128 double NOT NULL,"
                        + "radius129 double NOT NULL, radius130 double NOT NULL,"
                        + "radius131 double NOT NULL, radius132 double NOT NULL,"
                        + "radius133 double NOT NULL, radius134 double NOT NULL,"
                        + "radius135 double NOT NULL, radius136 double NOT NULL,"
                        + "radius137 double NOT NULL, radius138 double NOT NULL,"
                        + "radius139 double NOT NULL, radius140 double NOT NULL,"
                        + "radius141 double NOT NULL, radius142 double NOT NULL,"
                        + "radius143 double NOT NULL, radius144 double NOT NULL,"
                        + "radius145 double NOT NULL, radius146 double NOT NULL,"
                        + "radius147 double NOT NULL, radius148 double NOT NULL,"
                        + "radius149 double NOT NULL, radius150 double NOT NULL,"
                        + "radius151 double NOT NULL, radius152 double NOT NULL,"
                        + "radius153 double NOT NULL, radius154 double NOT NULL,"
                        + "radius155 double NOT NULL, radius156 double NOT NULL,"
                        + "radius157 double NOT NULL, radius158 double NOT NULL,"
                        + "radius159 double NOT NULL, radius160 double NOT NULL,"
                        + "radius161 double NOT NULL, radius162 double NOT NULL,"
                        + "radius163 double NOT NULL, radius164 double NOT NULL,"
```

```
                        + "radius165 double NOT NULL, radius166 double NOT NULL,"
                        + "radius167 double NOT NULL, radius168 double NOT NULL,"
                        + "radius169 double NOT NULL, radius170 double NOT NULL,"
                        + "radius171 double NOT NULL, radius172 double NOT NULL,"
                        + "radius173 double NOT NULL, radius174 double NOT NULL,"
                        + "radius175 double NOT NULL, radius176 double NOT NULL,"
                        + "radius177 double NOT NULL, radius178 double NOT NULL,"
                        + "radius179 double NOT NULL, radius180 double NOT NULL,"
                        + "veins1 double NOT NULL, veins2 double NOT NULL,"
                        + "veins3 double NOT NULL, veins4 double NOT NULL,"
                        + "veins5 double NOT NULL, img longblob NOT NULL,"
                        + "PRIMARY KEY(leafID), FOREIGN KEY(plantID) REFERENCES plant(plantID))";

        sment.executeUpdate(lTable);
        sment.close();

}

// create table network
public void createNetworkTable()
        throws SQLException {
        Statement sment =
                conn.createStatement();
        ;

        String nTable =
                "CREATE TABLE IF NOT EXISTS network ("
                        + "networkID int NOT NULL AUTO_INCREMENT,"
                        + "weights longblob NOT NULL,"
                        + "PRIMARY KEY (networkID))";

        sment.executeUpdate(nTable);
        sment.close();
}

// add a record to database
// (plant table)
public void addToPlant(Species s,
        File file) throws SQLException,
        FileNotFoundException {
        PreparedStatement ps =
                conn
                        .prepareStatement("INSERT INTO plant VALUES(null,?,?,?,?,?,?)");
        FileInputStream is =
                new FileInputStream(file);
        ps.setBinaryStream(1, is,
                (int) file.length());

        ps.setString(2, s.getName());
        ps.setString(3, s.getShape());
        ps.setString(4, s.getPattern());
        ps.setString(5, s.getArrangement());
        ps.setString(6, s.getMargin());
        ps.executeUpdate();
        ps.close();
}

// add a record to database
// (leaf table)
public boolean addToLeaf(Leaf s,
        File file) throws SQLException,
        FileNotFoundException {
        PreparedStatement ps =
                conn
                        .prepareStatement("INSERT INTO leaf VALUES"
                                + "(null,?,?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?,"
                                + "?,?,?,?,?,?,?,?,?,?)");

        if (s.getID() <= 0) {
                return false;
        }
        ps.setInt(1, s.getID());
        for (int i = 0; i < mCount; i++) {
                ps.setDouble(i + 2, s
                        .getMoments().get(i));
        }

        for (int i = 0; i < rCount; i++) {
                ps.setDouble(i + 6, s.getRadii()
                        .get(i));
        }
```

```
        for (int i = 0; i < vCount; i++) {
                ps.setDouble(i + 186, s
                        .getVeins().get(i));
        }

        FileInputStream is =
                new FileInputStream(file);
        ps.setBinaryStream(191, is,
                (int) file.length());

        ps.executeUpdate();
        ps.close();
        return true;
}

// add a record to database
// (network table)
public void addToNetwork(File file)
        throws FileNotFoundException,
        SQLException {
        PreparedStatement ps =
                conn
                        .prepareStatement("INSERT INTO network VALUES(null,?)");
        FileInputStream is =
                new FileInputStream(file);
        ps.setBinaryStream(1, is,
                (int) file.length());
        ps.executeUpdate();
        ps.close();
}

// add a record to database
// (plant table)
public void addToLeafImg(Blob b)
        throws SQLException {
        PreparedStatement ps =
                conn
                        .prepareStatement("INSERT INTO leafimg VALUES(null,?)");
        ps.setBlob(1, b);
        ps.executeUpdate();
        ps.close();
}

// query all from database
// (leaf table)
public ArrayList<Leaf> queryLeaf()
        throws SQLException {
        String sql = "SELECT * from leaf";
        Statement sment =
                conn.createStatement();
        ResultSet rs =
                sment.executeQuery(sql);
        ArrayList<Leaf> leaves =
                new ArrayList<Leaf>();

        while (rs.next()) {
                Leaf l = new Leaf();
                ArrayList<Double> m =
                        new ArrayList<Double>();
                ArrayList<Double> r =
                        new ArrayList<Double>();
                ArrayList<Double> v =
                        new ArrayList<Double>();

                for (int i = 0; i < mCount; i++) {
                        Double mTemp =
                                rs.getDouble("moment"
                                        + (i + 1));
                        m.add(mTemp);
                }

                for (int i = 0; i < rCount; i++) {
                        Double rTemp =
                                rs.getDouble("radius"
                                        + (i + 1));
                        r.add(rTemp);
                }
                for (int i = 0; i < vCount; i++) {
                        Double vTemp =
                                rs.getDouble("veins"
                                        + (i + 1));
                        v.add(vTemp);
                }

                l.setLID(rs.getInt(1));
                l.setID(rs.getInt(2));
                l.setMoments(m);
                l.setRadii(r);
                l.setVeins(v);
                l.setImg(rs.getBlob(192));
                leaves.add(l);
        }

        rs.close();
        sment.close();

        return leaves;
}
```

```
// query all from database
// (plant table)
public ArrayList<Species>
        queryPlant() throws SQLException {
        String sql = "SELECT * FROM plant";
        Statement sment =
                conn.createStatement();
        ResultSet rs =
                sment.executeQuery(sql);
        ArrayList<Species> species =
                new ArrayList<Species>();

        while (rs.next()) {
                Species s = new Species();
                s.setID(rs.getInt(1));
                s.setImg(rs.getBlob(2));
                s.setName(rs.getString(3));
                s.setShape(rs.getString(4));
                s.setPattern(rs.getString(5));
                s.setArrangement(rs.getString(6));
                s.setMargin(rs.getString(7));
                species.add(s);
        }
        return species;
}

// query a record from
// database by ID (plant
// table)
public Species queryByID(int plID)
        throws SQLException {
        String sql =
                "SELECT * FROM plant WHERE plantID = "
                        + plID;
        Statement sment =
                conn.createStatement();
        ResultSet rs =
                sment.executeQuery(sql);
        Species s = new Species();
        while (rs.next()) {
                s.setID(rs.getInt(1));
                s.setImg(rs.getBlob(2));
                s.setName(rs.getString(3));
                s.setShape(rs.getString(4));
                s.setPattern(rs.getString(5));
                s.setArrangement(rs.getString(6));
                s.setMargin(rs.getString(7));
        }

        return s;
}

// query by name and return
// species
public Species querySpec(String name)
        throws SQLException {
        String sql =
                "SELECT * FROM plant WHERE leafName = '"
                        + name + "'";
        Statement sment =
                conn.createStatement();
        ResultSet rs =
                sment.executeQuery(sql);
        Species s = new Species();
        while (rs.next()) {
                s.setID(rs.getInt(1));
                s.setImg(rs.getBlob(2));
                s.setName(rs.getString(3));
                s.setShape(rs.getString(4));
                s.setPattern(rs.getString(5));
                s.setArrangement(rs.getString(6));
                s.setMargin(rs.getString(7));
        }

        return s;
}

// query by plant ID and
// return plant name
// (plant table)
public String queryNameByID(int plID)
        throws SQLException {
        String sql =
                "SELECT * FROM plant WHERE plantID = "
                        + plID;
        Statement sment =
                conn.createStatement();
        ResultSet rs =
                sment.executeQuery(sql);
        String name = "";
        while (rs.next()) {
                name = rs.getString(3);
        }

        return name;
}

// query a record from
// database by name (plant
```

```
// table)
public int queryByName(String nme)
        throws SQLException {
        String sql =
                "SELECT * FROM plant WHERE leafName = '"
                        + nme + "'";
        Statement sment =
                conn.createStatement();
        ResultSet rs =
                sment.executeQuery(sql);
        int pID = 0;
        while (rs.next()) {
                pID = rs.getInt(1);
        }

        return pID;
}

// query a record from
// database (network) OR
// retrieve trained network
public File queryNetwork()
        throws SQLException, IOException {
        String sql =
                "SELECT COUNT(*) FROM network";
        Statement sment =
                conn.createStatement();
        ResultSet rs =
                sment.executeQuery(sql);
        int count = 0;

        while (rs.next()) {
                count = rs.getInt(1);
        }

        sql =
                "SELECT * FROM network WHERE networkID = "
                        + count;
        sment = conn.createStatement();
        rs = sment.executeQuery(sql);
        Blob b = null;

        while (rs.next()) {
                b = rs.getBlob(2);
        }

        File f = new File("weights.txt");
        OutputStream out =
                new FileOutputStream(f);
        byte[] buff =
                b.getBytes(1, (int) b.length());
        out.write(buff);
        out.close();

        return f;
}

// delete leaf image
public void deletelImg(int leafNum)
        throws SQLException, IOException {
        String sql =
                "DELETE FROM leaf WHERE leafID = "
                        + leafNum;
        Statement sment =
                conn.createStatement();
        sment.executeUpdate(sql);
}

// get number of records,
// table = plant/leaf
public int getCount(String table)
        throws SQLException {
        Statement sment =
                conn.createStatement();
        String sql =
                "SELECT COUNT(*) from " + table;
        ResultSet rs =
                sment.executeQuery(sql);
        int count = 0;

        while (rs.next()) {
                count = rs.getInt(1);
        }

        return count;
}

// close connection to
// database
public void closeConnection()
        throws SQLException {
        conn.close();
}

// main
public static void
        main(String[] args)
                throws ClassNotFoundException,
                SQLException {
```

```
                String url = "", user = "", pass =
                        "";
                try {
                        @SuppressWarnings("resource")
                        BufferedReader br =
                                new BufferedReader(
                                        new FileReader("file/SQL.txt"));
                        String line = br.readLine();
                        StringTokenizer tokens =
                                new StringTokenizer(line, ";");
                        url = tokens.nextToken();
                        user = tokens.nextToken();
                        pass = tokens.nextToken();
                } catch (IOException io) {
                        JOptionPane
                                .showMessageDialog(
                                        null,
                                        "File containing connection details does not exist.");
                }

                if (!url.equals("")
                        || !user.equals("")
                        || !pass.equals("")) {
                        SQL s =
                                new SQL("jdbc:mysql://" + url,
                                        user, pass);
                        s.connect();
                        s.createPlantTable();
                        s.createLeafTable();
                        s.createNetworkTable();
                        s.closeConnection();
                        System.out
                                .println("Operation completed.");
                } else {
                        System.out
                                .println("Cannot perform operation");
                }
        }

}
```

XII. Acknowledgement

I have been dreaming of this night for so long - the night when my brain child for almost a year would be born the next day. The most appropriate euphemism would be I am undertaking the last part of labor tonight; and tomorrow, I give birth. I could write more eloquently the following paragraphs as I have constructed them in my mind over and over again as I imagined the coming of this day, but I fear that whatever I have put together in my mind would translate into words unrelated and sentences hanging. So please, do bear with me as I try and show my appreciation and gratitude to those who rightly deserve it.

Truly, when you have a problem on your hands, no matter how big it is, you have to have faith that your God is bigger. And to Him, I owe this success. Our Almighty Lord has blessed me beyond belief this past year and one of the most tangible testament is this Special Problem. I may have had a rough journey but He surely delivered me and I reached my destination. "do not be anxious about anything, but in everything by prayer and supplication with thanksgiving let your requests be made known to God." (Phil. 4:6) And now I'm here.

To the university that nurtured me and moulded me into the person that I am today, thank you UP. I entered college as a kid who didn't care about the people who didn't mean much to her, and admittedly, was selfish. But as UP was known to be the University of the People, I slowly evolved into someone who strongly believed in advocacies that most people don't even understand. I fought for things and people mostly deemed useless. I can now confidently say that I am critical and know **HOW** to think rather than just fed **what** to think. I am now proud to say, *Isang sem na lang, UP!*.

To Sir Geoff Solano and Sir Jasper Obico who were the proponents of this study, I would like to thank them for entrusting me with a task as big as this and believing that I could do a good job at it. Thank you for all the help you gave and the time you imparted for me to be able to overcome this obstacle. To my other professors, Sir Bryann Chua, Doc Magboo, Ma'am Carpio, and Ma'am Sheila, thank you for teaching me the things that were necessary for me to finish this SP. And to Ma'am Perl, the closest to my heart, thank you for helping me and entertaining my queries about network training and parameters. I would never have had the courage to finish this SP without you. ;) To Ate Jeselle Sosa who was very patient with me, thank you for doing and giving all you can to help me in this endeavor.

To my family who have been nothing short of supportive(most of the time), I have nothing less than utmost gratitude to extend. To my mom, who has never failed to supply the things I needed and the things I wanted, who supported me all the way, and who was willing to overlook my mistakes and failures, thank you for never giving up on me. To my grandparents and generous aunts and uncles, thank you for the fun and relaxing times when I thought I had the time to rest.

To the person who gave back direction to my life, who stayed up with me through all the nights I slaved on writing codes and words accompanying it, who believed in me when I didn't even believe in myself, who cheered me up when I was at my lowest of lows, who reminded me of the things I so easily forgot, and to the one who I have so many things to be thankful for to, but I can't put it all in writing in fear that I forget most, Kim Isle Gonzales Cortez, thank you. *Salamat b, sa lahat-lahat. Ayoko na magdrama kasi baka kiligin ka at sila. ;)*

To my blockmates who both served as my idols and competitors because and for high grades, thank you for inspiring me and making me work harder both at the same time. To Cole, Maris, Liezl, Jezra, Sherwin, Aly, Sarah, Gera, Troy, Rizza, Mau, Ate Mich, Ate Hainah, and Ate Rhea, I will never forget you, our study group moments and the times we had fun, became sad, and shared stories - relevant and irrelevant. I hope I made a big enough mark in your lives to ensure even just a small place in your hearts because I surely reserved some for you in mine.

To Wowo, Jaye, Jayrell, Pat, Geebebe, and Jayvee, thank you for being the truest friends I have ever stumbled upon in my 4 and a half years in college. And even though I'm always the one bullied, I know you'll all miss me when I'm gone. ;) Thank you for being the people closest to me and for putting me in my rightful place every time I was confused. For all the depressing moments you turned fun, and the serious scenarios that became a hilarious anecdote afterwards, I am very grateful.

To my closest friends from high school, Tetel, Jem, Meng, Bhea, Joano, Marielle, and Ronne, thank you for understanding my predicament and not forgetting about me every time I couldn't make it to meet you guys. Thank you for the support and believing in my abilities although you don't know what I was up against. =)) And I would like to say sorry for the numerous times I hadn't really been a friend to one or most of you. Thanks for staying. ;)

Lastly, this is for all the people who didn't believe in me. This is the paragraph I was so passionate about when I was writing it in my head because this is really the moment I was waiting for. To the people who said I couldn't make it, who told me that I was just another lucky private school kid that passed the most prestigious entrance exam to the most recognized university in the Philippines, who predicted I would just give up, who bet against me and my success, who looked down upon me thinking I will stay down, I won't say I told you so. But instead, I want to thank you all. You fueled in me a drive to continue and show you that I'm not just a little girl who needed to hold on to everyone who offered. You taught me to fight for what I believe in - may it be right or wrong. Not every time I was right, but I was glad I fought for it.

I made a lot of mistakes in the past, and I regret most, if not all, of it. But as Chef Graham said in one episode of Masterchef, "It's not how you start, but how you end.". And here I am, about to end a journey I didn't think had an ending for me. It's a good thing I don't know how to give up; because if I did, I would be lost in a sea of people right now, mourning over what I could have done instead of standing out for the things I did.