University of the Philippines Manila

College of Arts and Sciences
Department of Physical Sciences and Mathematics

Plant Leaf Recognition using Neural Networks (LeafRApp)

A Special Problem in Partial Fulfillment
Of the Requirements for the Degree of
Bachelor of Science in Computer Science

Submitted by:

Jeselle Petrize M. Sosa
2009-27472

April 2013

ACCEPTANCE SHEET

The Special Problem entitled "Plant Leaf Recognition using Neural Networks (LeafRApp)" prepared and submitted by Jeselle Petrize M. Sosa 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, Ph.D.
Adviser

EXAMINERS:

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

8. Bernie B. Terrado, M.S. (candidate) _____ _____

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

_____ _____

Avegail D. Carpio, M.S.
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'g
Dean
College of Arts and Sciences

Abstract

Plant Leaf Recognition using Neural Networks (LeafRApp) is a desktop application which recognizes a plant from an input image file using the plant leaf's shape. A hybrid of two modelling techniques is used to extract features from the leaf. The Moment-Invariant method is used to extract the first four moments of the image while Centroid-Radii method is used to extract 36 radii with respect to the image's centroid.

An optional reduction feature is then used to reduce the list of matches of the unidentified plant species inputted by the user.

KEYWORDS: plant leaf recognition, Moment-invariant, Centroid-radii, Neural network

Table of Contents

I. Introduction

A. Background of the Study

Plant science is highly essential to life on Earth. It contributes to studies concerning different issues such as different species and the climate. Scientists and researchers conduct field work to do further analysis of these species. Doing such would require expertise and appropriate training on the said field.

Analysis of plants would depend on their different characteristics. From these characteristics, a knowledgeable individual would be able to identify from which classification it belongs to. There are several ways to identify a plant. The traditional methods commonly used are expert determination, recognition, comparison and use of keys and similar devices. These methods have different advantages of their own [1].

When it comes to reliability or accuracy, expert determination is said to be the best option. But despite this fact, an expert would be required time and delays for identification. Recognition is also deemed reliable, next to expert determination. In some cases, this method is not applicable. Comparison is also reliable but time-consuming or virtually impossible. The fourth option is claimed to be the most widely used method since it does not require time, materials or experience unlike in the other methods [1].

There are several parameters that can be used to identify a plant. Plant morphology, the study of the different parts of a plant our naked eye could see, or the physical and external parts [2]. One can examine its flowers or fruit. But the quickest and easiest way is by analyzing its leaves. Plant's leaves have different properties namely color, texture, structure or shape. The most commonly used in algorithms is the shape, most probably because the inputs are images of the leaf and the property that can be most easily extracted is its shape.

B. **Statement of the Problem**

Recognition of plants is a work that requires valuable time. In manual recognition, professionals consume lots of time in accomplishing this task prior to their research in which a plant's identity is necessary and a prerequisite for further studies. Say for example, a biology student is given 2 hours to identify a plant manually [3].

Other methods are proven to be helpful but the need for results in the soonest possible time seems to be requiring attention. The lack of necessary materials for examination of a plant or even its leaves is also an issue. Considering the field works that professionals are to do, they need to go to places where laboratory equipment won't be accessible, or to a biologist or plant morphologist who can easily identify a plant.

Besides time and portability issues, our naked eye's capability is very limited. There are some things technology can be better used with. Our naked eye can see things subjectively and cannot perform highly accurate calculations like what computers can do. It is possible for us to manually examine a plant's leaf and its properties but computers can be helpful in filling our limitations with its capabilities. Thus we see the need for help from technology in aiding these problems at hand.

The time it takes to identify a plant is dependent on an individual's level of expertise [3]. Thus, one who doesn't have enough knowledge would take hours or so in order to accomplish this task.

C. **Objectives of the Study**

General Objectives

To develop an application that accepts an image of a plant leaf and identifies the plant using its shape.

Specific Objectives

To develop a desktop application with the following features:

a. Expert:

- Update reference data set of plant species

  Experts such as biologists or plant morphologists with enough knowledge and authority is be responsible for keeping updates in the reference data set.

- Train, export and upload network

  Experts such as biologists or plant morphologists are responsible for training the network, saving the trained network to a file, and uploading it.

b. Unregistered user:

- Identify plant species from the plant leaf image

  From the user's collection are images of a plant's leaf. The image is normalized, i.e. the image is made invariant to translation, rotation and scaling. The edge is then detected and the radii from the centroid to the boundary points of the image are obtained. Using the outputs of the preprocessing operations, the neural network is trained and used for classification. A list of matches is displayed.

- Add textual features to narrow down the list of similar matches

  After the recognition process, a user is given an option whether to reduce the list by adding textual features of a plant leaf such as venation, arrangement along the stem, etc. The textual features are then compared with the records from the reference data set and a reduced list of matches is displayed.

- View/Add/Delete image files from/to the user's collection

The collection is the set of image files added by the user. User is allowed to view his collection and add files to it. Delete feature is also allowed to save space for erroneously inputted data.

- View reference data set.

  The reference data set is the training set. User can only view the reference data set. No modifications are allowed for the unregistered user.

D. **Significance of the Study**

The development of the application cuts the time spent on identifying plants and would allow researchers to work on their studies timely enough. The application can be used by anyone who is interested in the study of plants.

The identification feature is most useful to people without enough knowledge about plants. The time it takes for an individual to correctly identify a plant varies depending on his level of expertise [3]. The users would be able to identify a plant, or at least the family where it belongs to, without the need to conduct experiments and bring tools which could make it difficult for them to travel for field work. They can also make use of the application in different locations. The application does not require internet connection for the recognition process.

Even in laboratory setting, one needs not to conduct experiments or examinations in recognizing plants. Rather, they can use the application. With an image upload and the reference plant list, he could easily identify the plant or its family.

Several algorithms, methodologies and applications have already been developed using data from different countries. And several of them were able to successfully do so. But the

methodology to be used in this application has been proven to work exceptionally well relatively compared with others.

The application would be of significant help in preserving the local plants researches may acquire. With the use of the said application, there can also be organization of the plant list. The existing identified leaves are used to populate the list and to be reference for plant classification.

Since the application is used by non-experts in plants, it provides users with information regarding plants and their properties.

E. **Scope and Limitations**

a. The application allows its users to add images to its collection. Viewing and deleting are also allowed.
b. A user can only view his collection. Machines are independent and cannot communicate with each other.
c. A user can view the reference data set but he cannot perform changes on it.
d. Identification is dependent on the reference data available. A list of matches is the output with the percentage of match displayed.
e. For the plant recognition, only the shape is extracted from the leaf image file**.**
f. The reference database is updated using the Update option under the Network Menu.

F. **Assumptions**

a. The image background to be uploaded should be white for more accurate extraction of shape.
b. The photo is taken during the leaf's maturity stage.

c. The leaf's shape does not change.

d. The leaf is not distorted (folded, crumpled, crushed) or damaged significantly.

e. A group of experts such as biologists or plant morphologists knowledgeable in plant classification or morphology is responsible for updating the reference database.

f. For testing purposes, JPEG images with 480x360 dimensions would be used.

II. Review of Related Literature

The demand for computerized plant classifiers has been significantly increasing. Several technologies have been used to implement plant classification using leaf images. Some even

involved flowers. These have been widely used to efficiently manage and identify different plant species. Image processing played a big role in these methods.

Content-based image retrieval system was proposed in [4]. This field is said to be an active research area for several decades and has been given attention due to the increasing volume of digital images. The proposed system consists of steps. An image is the input for the digitizer. The image features are extracted. As for the images in the list, each is uniformly divided into 8-coarse partitions. The centroid of each partition can be selected as its dominant color. Texture and shape features would then be extracted. A vector for the combined features would be created next. The distance between the feature vector of the input image and the images from the list would be calculated and sorted. Lastly, the most similar images with minimum distance would be retrieved.

In [5], various image processing methods have been discussed and differentiated. Some leaf analysis methods were also described in the said paper. Under this category are leaf shape, venation extraction and analysis, leaf margin analysis, leaf texture analysis, other lamina-based methods and flowers and other plant organs. Approaches to leaf analysis include Elliptic Fourier Descriptors (EFDs) or Elliptic Fourier analysis (EFA) where leaf shape is analyzed in the frequency domain rather than spatial domain, Contour Signatures which represent the shape as a vector independent of orientation and location, Landmark and Linear Measurements which can be used to characterize an organ's shape, Shape features which are referred to as quantitative descriptors typically intuitive, easy to calculate, and applicable to a wide variety of different shapes, and Polygon Fitting and Fractal Dimensions, used to represent how completely a shape fills the dimensional space to which it belongs.

The common goal for an automated plant classifier is to have a picture fed to a portable computer, have it classified and allow the system to display closest matches as the output. In [6], a captured digital image is pre-processed and segmented using preferential segmentation. Then features are extracted from the image. A matching algorithm is applied to enable the system to display a list of possible matches of the leaf. This isn't unique. There actually exists an application which can do such but the training data is very limited to a particular location. A hybrid of pre-processing techniques was introduced in [7] using a combination of contrast stretching and adaptive thresholding that simultaneously adjusts the intensity level of leaf images using boundaries is developed. Pre-processing was also used in [8] for recognizing a plant through its leaf or image.

Neural networks have been used in several proposed systems as well. Back propagation was used for leaf classification in [9]. The software model would be used to suggest remedial measures for pest or disease management in agricultural crops. Probabilistic Neural Network, together with image and data processing techniques was employed in [10] for leaf recognition. Features are extracted and orthogonalized into 5 principal variables, which is then used as input for the probabilistic neural network. It is said to be 90 %(or so) accurate. A hybrid of shape modelling techniques was used in [11] based on the Moments-Invariant model and on the Centroid-Radii model. Multilayer Perceptron was used as classifiers for discrimination. Accuracies range from 90-100% which can be highly comparable to other research outputs.

To avoid the time-consuming point-wise matching used in other algorithms, [12] proposes the use of Multiscale distance matrix to obtain shape geometry while achieving invariance to translation, rotation, scaling and bilateral symmetry.

Plant classification has proven its worth not only for plant management but also for other purposes. Software developed for plant classification has also been developed for studying the diseases. The naked-eye's capabilities are very limited, subjective and can be expensive as well. The need for experts with enough experience is an issue. Segmentation was used in [13] using K-means clustering technique after a color transformation for the RGB leaf image was created. The texture features of the segmented infected objects are then calculated then passed through a pre-trained neural network.

III. Theoretical Framework

A. Machine Learning

Machine learning is defined to be the changes in systems that perform Artificial Intelligence (AI)-related tasks [14]. The role of machine learning is to build information processing systems with high performance [15].

B. Pattern Recognition

A pattern is said to be a set of measurements used to describe a physical object [14]. Recognition is defined to be the association of a classification and a label [16]. Put together, the ability to understand and read different types of data and make action based on the pattern category is said to be pattern recognition [17]. Some examples include face, handwriting recognition and fingerprint analysis. The components of pattern recognition include preprocessing, selection and extraction of feature, design of classifier and optimization according to [18].

1. Preprocessing

Preprocessing includes elimination of noise and normalization. It also defines a pattern's compact representation.

2. Selection and Extraction of Feature

Features are characteristics. In pattern recognition, there are two types of features, one with physical meaning and the other one without. The former include geometrical, structural or statistical features. The latter are features also called as mapping features.

Physical features need not to deal with features which are unrelated. Mapping features are said to make classification easier since between classes, clear boundaries are acquired.

3. Design of Classifier

In order to recognize an object after being preprocessed and having a feature selected and extracted from it, a classifier is then used. There are different approaches. Some involve concept of similarity while some use the probabilistic approach.

4. Optimization

Optimization is part of every component. It is used to assure quality of input. Optimization techniques are also applied in acquiring features. It is then used for lowering classification error rate.

C. **Plant Recognition**

Plant recognition is the process of identifying a plant using its features through different methodologies. A plant has different parts and features that can be used for recognition. Leaves, flowers fruits, roots, etc. are some of those. Existing systems make use of image for preprocessing and classification. Visual characteristics of the plant are extracted and analyzed.

D. **Leaf**

Leaves are said to be the powerhouse of plants. Food production in plant occurs in leaves. Their structure allows conversion of sunlight energy to chemical energy which can be used by plants as food [19].

Leaves come in different sizes and shapes. They differ in various categories as well. Some properties of leaves include the petiole, blade, edge, veins, arrangement along the stem, etc. The following are the types and properties per category of leaves according to [20-23].

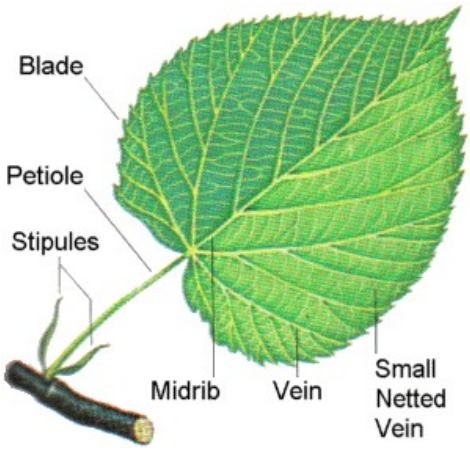


Figure 1. Leaf Structure

1. Persistence

Coniferous doesn't lose its leaves and is evergreen. Deciduous loses its leaves during fall.

2. Simple vs. Compound

Simple leaves have no leaflets while compound leaves have little leaflets.

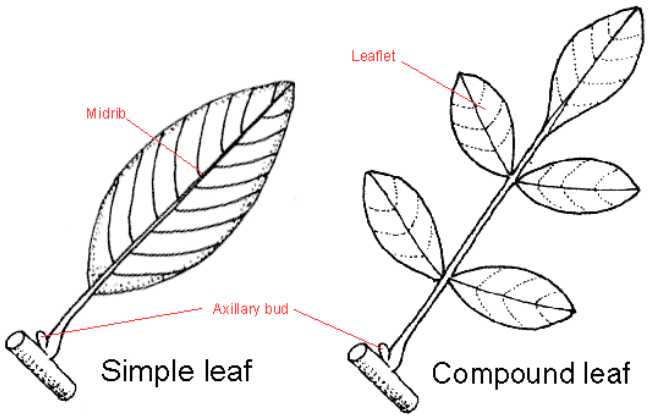
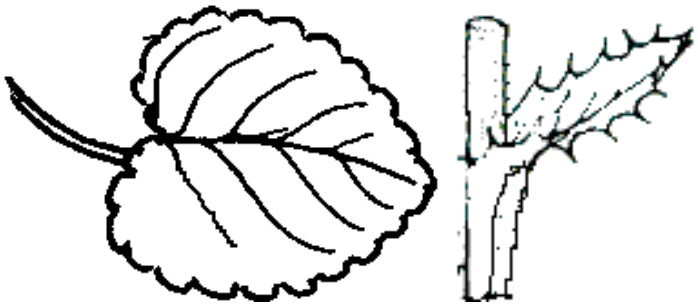


Figure 2. Simple vs. Compound Leaf

3. Petiole

Leaves that have petiole are petiolated while leaves that don't have any are sessile.

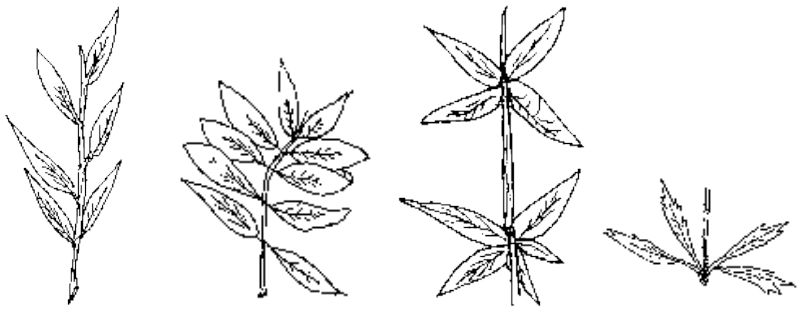


Petiolated Sessile

Figure 3. Leaf Petiole

4. Arrangement along the stem

Alternate leaves alternate their position in the stem while opposite leaves position themselves directly across each other. Whorled leaves are positioned at the same level around the stem. Rosulate leaves form a rosette, like a ring around the stem.



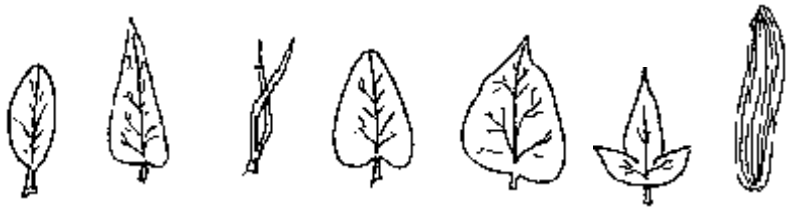
Alternate Opposite Whorled Rosulate

Figure 4. Leaf Arrangement along the Stem

5. Shape of the Blade

A leaf's blade is the expanded part of the leaf. Elliptic form an ellipse and are 2 or 3 times wide. Lanceolate leaves are spear-shaped. Acicular leaves are needle-shaped. They are several times longer than wide. Ovate leaves have shape like that of an egg.

Cordate leaves are heart-shaped. Hastate leaves are halberd-shaped. Linear leaves are strip-shaped.



Elliptic Lanceolate Acicular Ovate Cordate hastate linear

Figure 5. Leaf Blade Shape

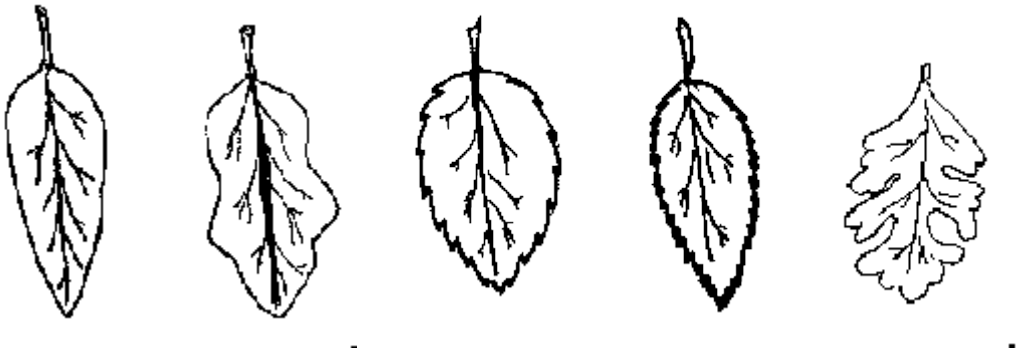
6. Venation

Venation defines how veins are arranged in the blade of a leaf. Venation could be reticulated, parallel or dichotomous, etc. Veins follow a net-like pattern in reticulated venation. Veins are interconnected like strands of a net. It is also the most common type of venation. This type occurs in nearly all dicotyledonous Angiosperms. Veins are parallel or nearly parallel in parallel venation. Veins are smaller in size connecting the large veins. This type occurs in nearly all monocotyledonous Angiosperms. The third type is dichotomous venation, where veins branch out like branches of a tree. This type of venation is very rare. It occurs in some ferns and in *Gingko biloba*[24].

Figure 6. Leaf Venation

7. Leaf Edge

A leaf's edge is called as its margin. Entire leaves have a smooth margin. Sinnuate leaves have little curves with smooth edges like waves. Dentate leaves have little teeth at the margin. Serrate have little bent teeth like those of a saw. Lobed leaves have divisions that do not arrive to the center of the half blade.



Entire Sinuate Dentate Serrate

Lobed

Figure 7. Leaf Edge

E. **Moment-Invariant**

Moment-invariant is a modelling technique used for image processing. Moments have the ability to provide characteristics on an object that can uniquely denote its shape. The moment invariants are used in response to the degradations or distortions as the image is acquired. There are 7 moment features according to [25] which can be used to describe shapes. The use of this technique makes the image invariant to rotation translation and scaling. From [25], given an order (p,q) and function $f(x,y)$, the regular moments are defined by:

$$m_{pq} = \iint x^p y^q f(x,y)\, dxdy$$

1. Translation Invariance

Moments are computed and normalized with respect to the center of gravity resulting to the center of the mass of the distribution positioned at the center of gravity. The central gravity coordinates are given by:

$$x = m10m00 \qquad \bar{y} = m01m00$$

central moments can be defined by:

$$\mu pq = xy[x\text{-}xpy\text{-}\bar{y}qf(x,y)]$$

2. Rotation invariance

A set of 7 moment invariants can be computed from the second and third order values of the normalized central moments. The first four invariant moments invariant to rotation are the following:

$\phi_{1=m20+m02}$

$\phi_{2=(m20-m02)2+(2m11)2}$

$\phi_{3=(m30-3m12)2+(3m21-m03)2}$

$\phi_{4=(m30-m12)2+(m21-m03)2}$

3. Scaling invariance

Size invariants are derived from algebraic invariants. However, simple size normalization can result to these. The moments are made invariant to scaling by dividing a power of μ_{00} .The normalized central moments are given by:

$$m_{pq} = \mu_{pq}\mu_{00}\omega \quad \text{where} \quad \omega = 1+p+q2$$

F. **Centroid-Radii**

Centroid-Radii model is a modelling technique used for estimating shapes of image objects. Here are some terms defined in [11].

1. Shape

   A shape is an area of black on a white background.

2. Centroid

   The Centroid of an image is the point (Cx,Cy) where Cx is the average of the x-coordinates of the black pixels and Cy is the average of the y-coordinates all black pixels.

3. Boundary point

   A boundary point is a black pixel with a white neighbor pixel. A series of boundary points together is a boundary.

4. Radius

   A radius is a straight line that joins the Centroid and a boundary point. Using the Euclidean distance, the shape's descriptor is represented by the lengths of a shape's radii at regular intervals. The regular interval is represented by Θ (in degrees). The number of intervals is $k = 360/\Theta$. Normalization of all radii lengths are done by dividing the radii by the longest radius length from the set of radii. The shape descriptor can be represented as a vector of all the radii.

G. **Canny Edge Detection**

An edge detection algorithm developed by John F. Canny in 1986. It has become one of the standard edge detection methods still used in research [26]. Edge detection reduces the amount of image data while maintaining the structural properties. According to [26], the canny edge detection algorithm consists of 5 separate steps: smoothing, finding gradients, non-maximum suppression, double thresholding and edge tracking by hysteresis.

1. Smoothing

   Smoothing involves blurring of an image to remove noise. *Gaussian filter* is applied on the image to smooth it.

2. Finding gradients

   The edges are marked depending on which locations in the image have large magnitudes. Edges where the grayscale intensity of the image varies the most are found by determining the image gradients. The *Sobel-operator* is used to determine the gradients of the image after being smoothed.

3. Non-maximum suppression

   The only ones to be marked as edges are the local maxima. This sharpens the edges. The local maxima in the gradient image are preserved, everything else is deleted.

4. Double thresholding

   Thresholding is used to determine potential edges. It is used to distinguish between true edges and edges marked because of noise or color variations. Using this, only the edges stronger than a certain value would be retained. Pixels stronger than the high threshold are strong while pixels weaker than the low threshold are weak.

5. Edge tracking by hysteresis

All edges that are not connected to a strong edge are suppressed and final edges are determined. Strong edges can immediately be included in the set of final edges. Weak ones can only be included if they're connected to strong ones. BLOB (Binary Large Object) can be used for edge tracking.

Here is an example of the application of the Canny Edge Detection in an image:

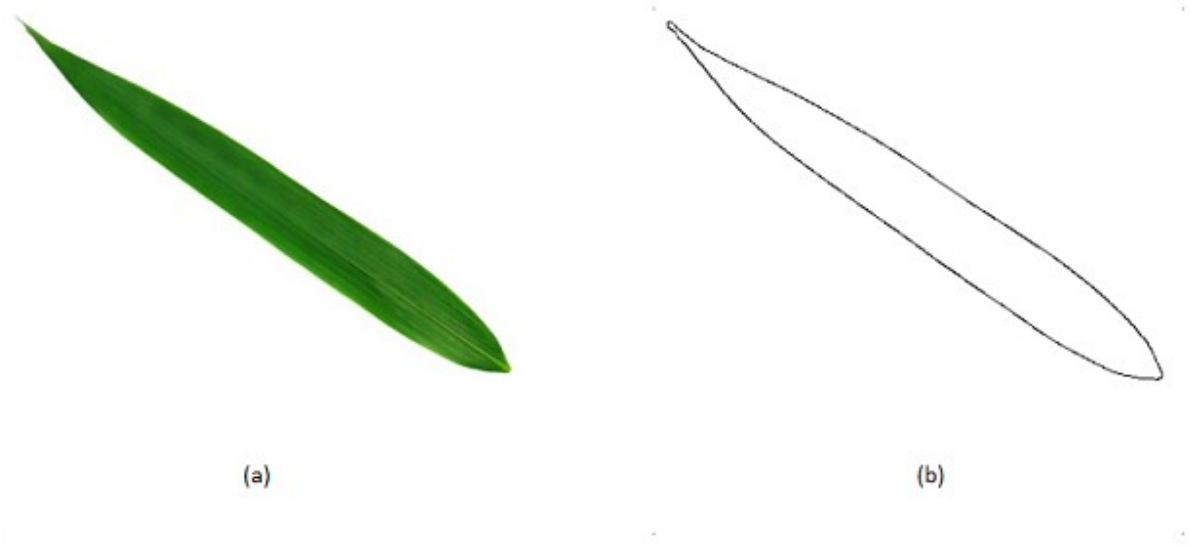


Figure 8. (a) Before and (b) After Canny Edge Detection

H. **Neural Network**

Neural networks are a programming paradigm inspired by the biological structure of the human brain [27]. They are most commonly used in artificial intelligence problems like pattern recognition. The human brain has billions of cells called *neurones*. They are also defined as simple processing units. They can be linked together in a *network*. *Input* is received from the lower neurones in a processing chain while *output* is transmitted to higher neurones in a processing chain.

Neurotransmitters are responsible for forming input to the next neurone and constitute the message sent among them. The messages sent can be of three forms: *excitation*, *inhibition* and potentiation. Matrices of numbers that represent the connection between neurones are called *weights*.

Excitation is when excitatory neurotransmitters increase the probability of the activation of the next neurone. Inhibition is when inhibitory neurones decrease the probability of the activation of the next neurone. Potentiation is when the sensitivity of the next neurone is adjusted to excitation or inhibition. This is also referred to as the learning mechanism [27].

I. Multilayer Perceptron

Multilayer Perceptron is an example of a feedforward artificial neural network solving different types of problems including pattern recognition [28]. It has one or more layers between the input and output layers. Feedforward refers to the unidirectional flow of data from input to output. Backpropagation learning algorithm is used in training this type of network. The basic Multilayer Perceptron is implemented in this manner, as discussed in [27]:

1. Initialization of network. All weights are set to random numbers from -1 to 1.
2. Presentation of the first training pattern and output.
3. Comparison of the network output with the target output.
4. Backward propagation

   Output layers of weights are corrected using the following formula

who= who+(ɳδooh), where who = weight connecting hidden unit(h) with utput unit(o), ɳ = learning rate, oh = output at hidden unit(h), δo=oo1-ooto-oo, where oo= output at node o of the output layer and t-o is the target output for that node.

5. Error calculation. The difference between the target and the output vector are calculated. The following function can be used:

   E=n=1pto-oo2p

6. Repetition from step 2 for each pattern in the training set for the completion of an epoch.
7. Shuffling of the training set. This disregards the influence of the order of the data.
8. Repetition from step 2 for a set of a number of epochs or until error stops changing.

IV. Design and Implementation

A. Use-Case Diagram

There are two types of users. The target end-user is the general public or individuals who seem to show interest in plant species. Researchers such as environmentalists or wildlife conservationists without enough knowledge about plant species can also be potential users of the

application. The other would be experts such as morphologists who are responsible for updating the reference data set.

Each unregistered user, after having the application downloaded from the web-based system and installed in their respective machines, have the following abilities to: view plant species, add an image to collection, edit collection, delete records, recognize plant and request for data updates.

The unregistered user can add data describing a plant species and its properties. He can delete images for erroneously inputted data. He can also recognize a plant using an uploaded image and some textual features.

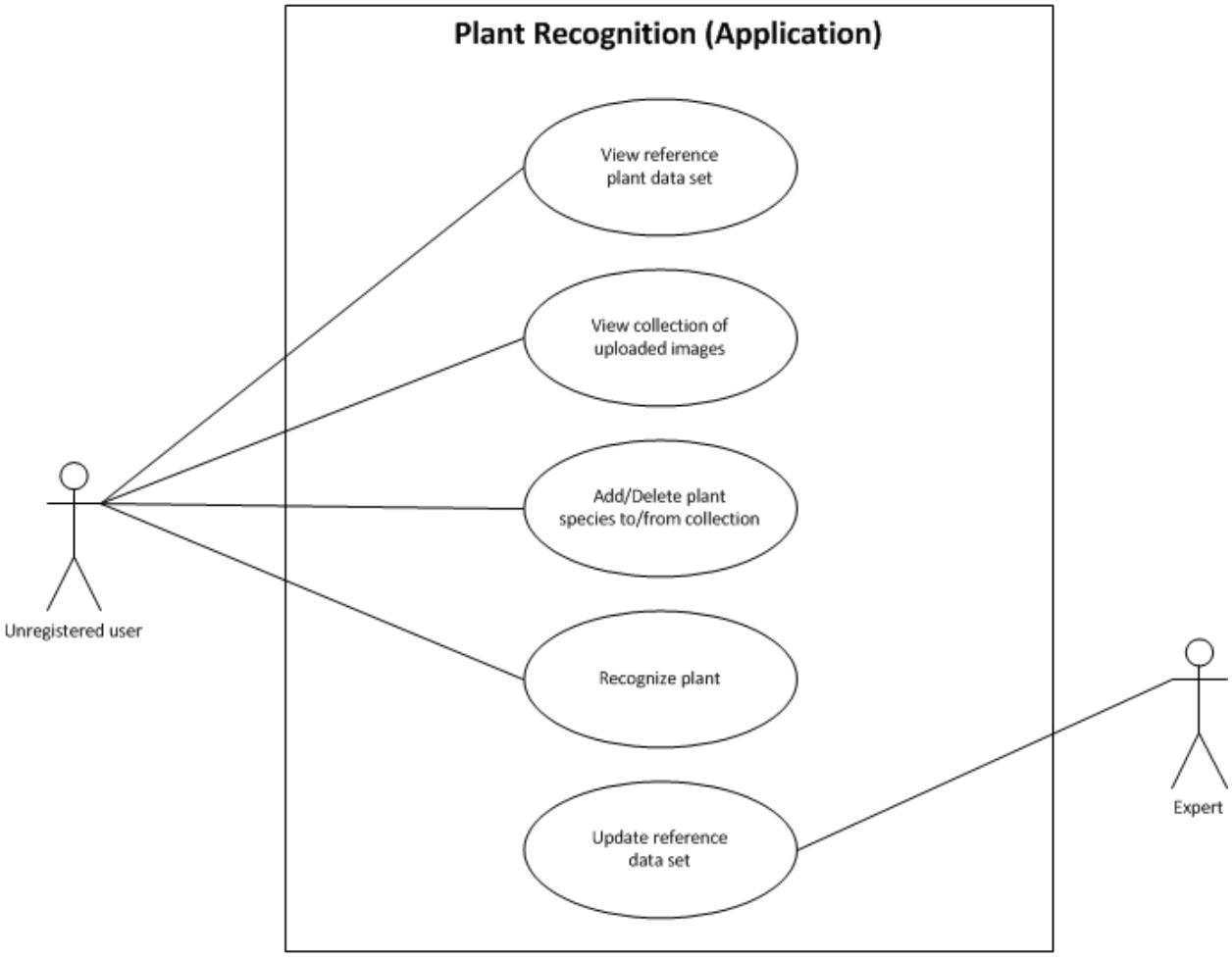


Figure 9. LeafRApp Use-Case Diagram

B. **Class Diagram**

The main function of the application is recognition. Recognition involves edge detection and use of multilayer Perceptron as a neural network classifier. Recognition makes use of plant species record for identification. The plant species records also have properties, both textual and visual. The plant species class makes use of another class which is Image.

Image represents an image's properties. Several operations can also be performed in the image for processing and normalization before classification to remove noise. An image's vector for its pixels and for its radii can also be obtained using this class.

Edge Detection class implements Canny Edge Detection Algorithm. The MLP is the neural network classifier used. Multilayer Perceptron class has the network. This classifier is used for the recognition using inputs coming preprocessing operations.

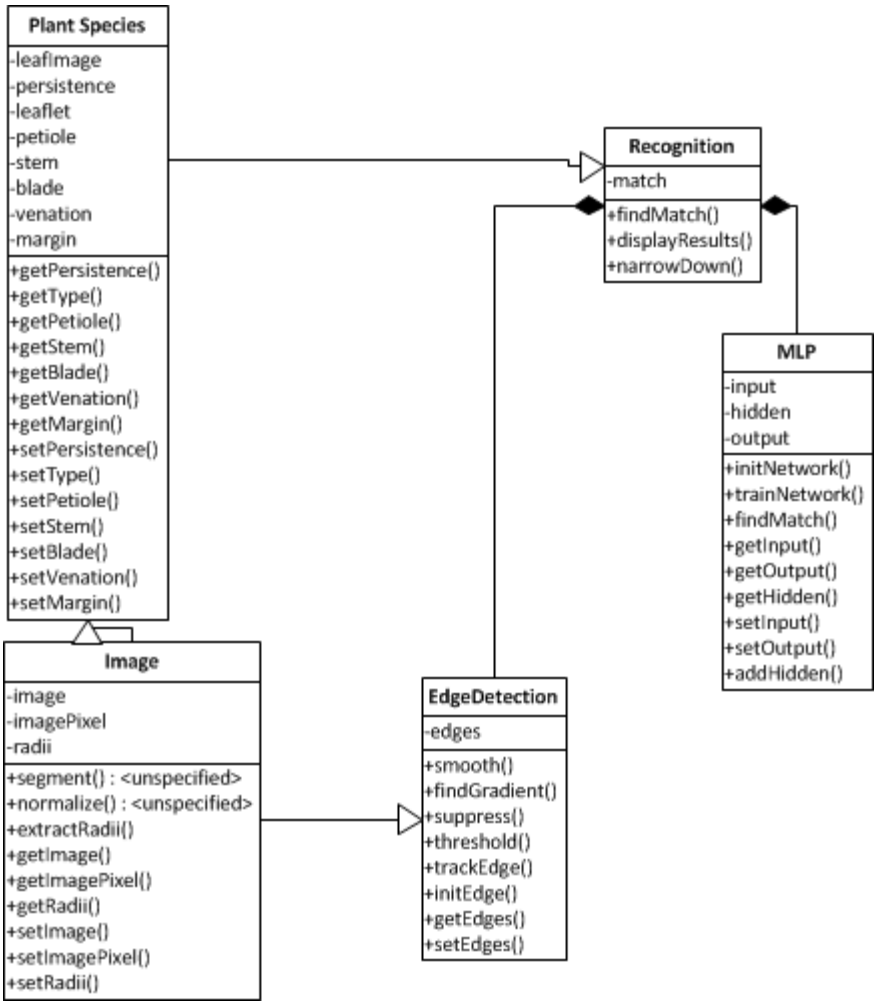


Figure 10. LeafRApp Class Diagram

C. **Flowchart**

The user inputs an image. The image is normalized and made invariant to translation, rotation and scaling using the Moment-Invariant Method. The edge is detected using Canny's Edge Detection Algorithm. Centroid-Radii method is then used to compute for the Centroid and to obtain the set of radii.

The neural network is then trained. The input comes from the preprocessing operations performed on the image. The plant can now be processed for identification using the Multilayer Perceptron. If the plant is recognized, then a list of possible matches would be

displayed and the user is asked whether or not he wants to reduce the list using the textual features of the plant. Else if the plant is not recognized, a “Plant not found” message is displayed.

Figure 11. LeafRApp Flowchart Diagram

D. **Technical Architecture**

The plant recognition application is a stand-alone application that can be used on a computer. The application is installed into the computer needs internet connection for updates in the training set. It makes use of local memory for storage of information.

Minimum System Requirements

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

V. Results

The Plant Leaf Recognition desktop application (LeafRApp) is an application that accepts leaf image files as input and recognizes plants using the leaf's shape. LeafRApp has two types of users, the expert and the unregistered user. Figure 12 shows the welcome screen of LeafRApp for the Expert access level. There are three tabs for the Expert, the Species tab is shown in Figure 13.

Figure 12. LeafRApp Welcome screen (Expert)
Figure 13. LeafRApp Species Tab (Expert)

Figure 14. LeafRapp Canny Edge Menu (Expert)

Figure 15. LeafRApp Low threshold (Expert)

Figure 14 shows the Canny Edge Menu. Under this menu are three menu items, low threshold, high threshold and detect edge. Dialog boxes for the low and high threshold are shown in Figures 15 and 16 respectively.

The image file chooser is shown in Figure 17. There are two image fields found in the Species Tab. The image field on the left shows the thumbnail of the original image while the image field on the right shows the edge detected from the image. The expert can choose the image file and input the textual features on the textboxes found below the first image field on the

left. The chosen image file is initially displayed on the two image fields. The detected edge is then displayed on the image filed found on the right panel.

Figure 16. LeafRApp High Threshold (Expert)

Figure 18 shows an example of an image file chosen by the expert. The textual features representing the leaf properties inputted by the expert are shown on the bottom left part of the panel.

Figure 17. LeafRApp Image File Chooser (Expert)

Figure 18. LeafRApp Add Species (Expert)

After an image has been chosen, expert has to choose whether he adds it as a new species or as an existing species. Figure 19 shows a confirmation dialog box for adding a new plant species while figure 20 shows a confirmation dialog box for adding an existing plant species. Once a new plant species is mistakenly added as an existing species, i.e., the expert clicks the Existing button instead of New button, a message is displayed telling him that the plant species does not exist yet.

Figure 19. Add as New Plant Species (Expert)

Prior to adding another image file as a new plant species or an existing plant species, expert has the option to detect the edge first to be able to determine the correct low and high threshold parameters for the Canny Edge Detection. Figure 21 shows the edge detection process output of the sample leaf image file chosen by the expert.

Figure 20. LeafRApp Add as Existing Plant Species (Expert)

Figure 21. LeafRApp Canny Edge Detection (Expert)

Another menu is the Network Menu as shown in Figure 22. Under which are two menu items, the Save and Upload options. The Neural Network Tab is shown in Figure 23. Found on

the left side of the panel are summaries of the leaf image files and the network parameters which could be modified by the expert. Network training would be dependent on these parameters. The train button allows the expert to train the network while the Stop Learning button allows the expert to stop the network from training. The clear button resets the network parameters.

Figure 22. LeafRApp Network Menu (Expert)

An example of network training is shown in Figure 24. The epochs and the total network error are updated and found on the top left part of the panel. The train button is disabled while the stop learning button is disabled to allow the expert to have control of the network training and its configuration. Figure 25 shows that the network has stopped training after 4701 epochs, achieving a total network error of less than 0.01. Only the fields for the hidden neurons, learning rate and max error could be edited. The fixed number of input neurons is 40, representing the 40 input values from the two modelling techniques, the Moment-Invariant and Centroid-Radii methods. The number of output neurons is dependent on the number of identified plant species on the database.

Figure 23. LeafRApp Network Tab (Expert)

Figure 24. LeafRApp Network Training (Expert)

Figure 25. LeafRApp Network Training Convergence (Expert)

After having the network trained, the expert could generate an error graph showing the total network error per epoch. Figure 26 shows the error graph generated after training the network.

Figure 26. LeafRApp Trained Network Error Graph (Expert)

Figure 27. LeafRApp Network Saving (Expert)

In order to make use of the trained network, the expert can save it to a file. Figure 27 shows a confirmation dialog box asking the expert if he wants to save the network. Figure 28 confirms the success of saving the trained network to a file.

The next step after saving the trained network is uploading it to the database. The expert can update the network using this feature. Unregistered users can then download the trained network file. A confirmation dialog box for the upload is shown in Figure 29. The success of file upload is shown I Figure 30. A message dialog box is shown if the application failed to upload the trained network file to the database.

Figure 28. LeafRApp Trained Network Saved (Expert)

Figure 29. LeafRApp Trained Network Upload (Expert)

Figure 30. LeafRApp Trained Network Uploaded (Expert)

The third tab for the expert user type is the Unknown Species Tab. This tab is for verification of the probable new plant species found by unregistered users. Found on the left side in Figure 31 are an empty list and three buttons on the bottom left. The update list button updates the list of the unknow species sent by the unregistered users. The download button allows the expert to download the image file of the unknown species. The delete button deletes the file from the database.

Shown in Figure 32 is an updated list of unknown species. When an unknown file is selected from the list, the image is displayed on the right side of the panel.

Lastly, Figure 33 shows the Exit menu. Under which is the close option which closes the window and exits the program.

Figure 31. LeafRApp Unknown Species Tab (Expert)

Figure 32. LeafRApp Unknown Species List Updated (Expert)

Figure 33. LeafRApp Exit Menu (Expert)

The second type of user is the unregistered user. An unregistered user is allowed to view the reference dataset from the database. But he is now allowed to make modifications to it. He is also allowed to recognize a plant using an input image file. Reduction can also follow after recognition. He is also allowed to have a plant leaf image uploaded to the database to be verified as a new plant species by the expert.

The welcome screen for the unregistered user type is shown in Figure 34.

Figure 34. LeafRApp Welcome screen (Unregistered User)

Figure 35. LeafRApp Training Set Tab (Unregistered User)

The Training Set tab is shown in Figure 35. A list of the plant species in the database is displayed on the left side of the panel. Once a user selects from the list, the image is displayed on the center. The properties of the leaf is displayed on the right side of the panel. No modifications can be performed by any unregistered user on the training set. Only viewing can be done on this tab.

Figure 36 shows the image file chooser for the unregistered user. As the user chooses an image file, using the Load Image button, it is displayed on two parts of the panel, one on the left and the other on the center. The left side of the panel shows a thumbnail of the original image file before preprocessing. While the other shows the edge detected after the low and high threshold have been configured. The defaut values for the low and high threshold are 0.1 and 15.0 respectively.

Figure 36. LeafRApp Image File Chooser (Unregistered User)

Figure 37 shows the Leaf Menu which has two sub items, the Canny Edge and Verify optios. Canny has three menu items, the low threshold, high threshold and detect edge options. The low and high threshold parameters can be modified as shown in Figures 38 and 39. The Canny Edge menu is basically the same with that for the Expert type of user. The verify sub item is just added.

Another feature for the unregistered user is deleting of a plant species from his collection. He is allowed to remove an image from the testing set folder using the Delete Species button. User is asked for confirmation of deletion of the selected image from the list on the lower left side of the panel.

Figure 37. LeafRApp Leaf Menu (Unregistered User)

Figure 38. LeafRApp Low threshold (Unregistered User)

Figure 39. LeafRApp High threshold (Unregistered User)

Another menu is the Network menu. Under it is the Update option as show in Figure 40. The update option allows the user to download the update trained network file. It also allows the users to update the list of the training set which can be viewed on the training set tab.

Figure 41 shows a dialog box telling the user that the update was successful. A message dialog box is displayed if there be any problems encountered in updating the reference data set on the training tab or in downloading the trained network file which is used for the recognition process, which is the main feature of the application.

Figure 42 shows the Results menu. Under this menu is the acceptance percentage which determines the cutoff for the match list displayed on the table found on the top right side of the panel. The two columns of the results table represent the common name and the percentage of

match for each species. These species are ranked from highest to lowest. The acceptance percentage can be modified as shown in Figure 43.

Figure 40. LeafRApp Network Menu (Unregistered User)

Figure 41. LeafRApp Network File Update Success (Unregistered User)

Figure 42. LeafRApp Results Menu (Unregistered User)

Figure 43. LeafRApp Acceptance percentage (Unregistered User)

The most essential feature of the application is the recognition process. Figure 44 shows the recognition of an input leaf image file by the user. The results are shown on the top right side of the panel. Only the images which have been added to the collection can be recognized. A chosen image file won't be allowed for recognition unless added to the collection which makes use of the user's local memory.

Figure 44. LeafRApp Recognition (Unregistered User)

After the recognition process, an additional feature is the Reduction. Textual features can be inputted by the user on the text fields found on the bottom right side of the panel. The reduce button is only enabled after a leaf has been identified. Otherwise, it remains disabled. The clear button clears all the text fields. Figure 45 shows the reduced list of matches after reduction. Leaf type was entered by user and only the species with this leaf type are displayed on the list of matches.

The verify feature is shown in Figure 46. A message dialog box signals the user that image file is pending for verification.

Figure 45. LeafRApp Match List Reduction (Unregistered User)

Figure 46. LeafRApp Verify (Unregistered User)

The Exit menu is shown in Figure 47. Under which is the Close option which closes the window and exits the program.

Figure 47. LeafRApp Exit Menu (Unregistered User)

VI. Discussion

The Plant Leaf Recognition desktop application (LeafRApp) accepts an input image file and recognizes the plant species based on the leaf's shape. Moment-Invariant method is used to extract the first four moments of the image. Canny Edge Detection is then used to extract the outline or shape of the leaf. Centroid Radii makes use of the leaf's shape to extract 36 radii computed with respect to the centroid of the image outline.

There are two types of users, the expert and the unregistered user. The expert is responsible for adding records to the reference data set or the training set by inputting the leaf image and its textual properties. There are two ways in adding a record, first by adding a record as a new plant species and the other by adding a record as an existing plant species, i.e., the plant has already been previously added by the expert but added for training purposes.

Assuming he is knowledgeable in plant morphology or botany and neural networks, the expert is also responsible for training the network. Using the records from the database, the

network is trained, saved and exported to a file. The trained network can be uploaded by the expert to the database. An error graph generator is added showing the total network error per epoch after the network has been trained. This error graph can be reset. The database summary is shown in the network tab. The network parameters can also be modified by the expert.

The other type of user is the unregistered user. The unregistered users and their collections are independent per machine, making their testing sets vary. An unregistered user is allowed to view the training set (a sample image and its properties) but make no modifications on it. He can also perform the major functionality of the application, which is recognition. He is to input a plant leaf image, add it to his collection and have it recognized. The same preprocessing operations are performed to the image file when a new record is added to the database. Moments are extracted, edge is detected and radii are computed. He can also modify the parameters of Canny Edge to remove unnecessary lines found within the leaf's shape or outline which could possibly decrease or affect the accuracy of the recognition process itself.

Another feature is the reduction of the list of matches. Originally, a table is shown with the plant species name and the percentage of match. A panel is added below this table for optional inputting of additional textual features in case the user intends to reduce the list of matches.

Compared to manual recognition of plant species, one needs relatively significant level of expertise in the field of botany or plant morphology to perform this operation. It takes time for a non-expert individual to be able to identify a plant species. In addition, his identification could be highly inaccurate. LeafRApp makes recognition easier for non-experts individuals

or researchers who are interested in studying or determining plant species using just an image file as input. In a matter of a few seconds, the application can display a list of possible matches where the plant species could come from, could belong to or even be identical to.

Preprocessing methods which have been proven to be relatively excellent and competitive to other techniques are used to convert the images to numerical data. There is no need for manual input of numerical data.

On the other hand, in general, the recognition process is still dependent on the data. Very few leaf images for a particular species could make the network unreliable. Its data-dependency would affect the accuracy of the network. And the parameters of the network are not determined easily. It takes time to train a network using the appropriate parameters. In addition, recognition requires connection to the database so the application can output the properties or the name of the plant species where the unknown could come from.

Other similar applications have been implemented but using a different algorithm or methodology. The hybrid of the two modelling techniques, Moment-Invariants and Centroid-Radii shows relatively excellent output compared with other existing applications. The choice of the neural network, which is MultiLayer Perceptron is also helpful since it has Backpropagation for mimnimization of errors. Other open source implementations make use of other neural networks or methodologies.

Some species are just too similar to distinguish. And only the shape alone is the basis for recognition. The addition of the reduction feature decreases the probability for confusion and increases the accuracy of the output. Leaf properties can be added textually to narrow down the list of matches and to be nearer the closest possible match of the plant species being identified.

VII. Conclusion

The Plant Leaf Recognition Application (LeafRApp) is a desktop application that accepts an input image file and displays a list of possible matches, with percentage of match, for the plant species depending on the training set. It makes use of the leaf's shape extracted from the image file using Canny Edge Detection and converted to numerical data using two

modelling techniques namely Moment-Invariants and Centroid Radii. These numerical data is then used as input for training and testing the Neural Network.

There are two types of users, expert and unregistered user. The expert is responsible for uploading training data, including the image files and textual features of the plant species. The parameters, the low and high threshold, for the edge detection of the image file can also be modified. He is also responsible for training, saving, exporting and uploading the network file to the database. In addition, he is to verify the unknown species sent by users which could probably be new plant species. An additional feature is the error graph generator, showing the total network error for the epochs after the network has been trained.

The unregistered user has the privilege to view the plant species from the training set. He is allowed to input image files for recognition. An optional reduction feature is added in order to reduce the list of matches displayed after recognition. His input image files can also be added to the database to be verified by the expert. He can also modify the parameters for the edge detection process. Only the files which have been added to the collection can be recognized. Textual features can also be inputted after the recognition process to reduce the list of matches.

VIII. **Recommendation**

The Plant Leaf Recognition desktop application (LeafRApp) can further be enhanced in different ways. The present implementation of the said application makes use of the leaf's

shape alone. Other leaf properties such as venation could be extracted and converted to numerical data to be added to the inputs of the network as basis for training and recognition. Canny Edge Detection can be used to extract venation by modifying the parameters of the detection process. Doing so could possibly improve the accuracy of the recognition process without the need for several input image file per plant species.

The implementation of this said application makes use of a hybrid of two modelling techniques. Other techniques could be explored to possibly provide higher accuracy.

Another feature that can be added would be a summary of the plant species which could possibly be related. The relationship of these plant species can be represented using a tree. Similarities in different categories between these plant species could be the basis for creating a relationship tree.

For researchers who aim to study more and discover new plant species, a geographical feature would be helpful, showing where each plant species could be found. Mapping the location of these plant species could help researchers to identify possible relationship or even identify the family of an unknown or newly identified species based on the location where it was found.

An informative part could also be added to be able to preserve the records of plant species in a particular location.

IX. Bibliography

[1] Plant Identification," [Online]. Available: http://www.ibiblio.org/unc-biology/herbarium/courses/CHPT25.html. [Accessed 2012].

[2] "American Society of Plant Taxonomists," 2006. [Online]. Available: http://www.aspt.net/careers/careers8.php. [Accessed 2012].
[3] P. J. Obico, Interviewee, [Interview]. 15 October 2012.
[4] D. S. S. and B. A. P., "CONTENT BASED IMAGE RETRIEVAL SYSTEM," 2012.
[5] J. S. Cope, D. Corney, J. Y. Clark, P. Remagnino and P. Wilkin, "Plant Species Identification using Digital Morphometrics: A Review".
[6] N. Valliamma and D. S. Geethalakshmi, "Automatic Recognition System Using Preferential Image Segmentation For Leaf And Flower Images," 2011.
[7] N. Valliammal and D. N. Geethalakshmi, "A Hybrid Method for Enhancement of Plant Leaf Recognition," 2011.
[8] C. Pornpanomchai, C. Kuakiatngam, P. Supapattranon and N. Siriwisesokul, "Leaf and Flower Recognition System (e-Botanist)," 2011.
[9] M. S. P. Babu and B. Rao, "LEAVES RECOGNITION USING BACK PROPAGATION NEURAL NETWORK-ADVICE FOR PEST & DISEASE CONTROL ON CROPS".
[10] S. G. Wu, F. S. Bao, E. Y. Xu, Y.-X. Wang, Y.-F. Chang and Q.-L. Xiang, "A Leaf Recognition Algorithm for Plant Classification using Probabilistic Neural Network," 2007.
[11] J. Chaki and R. Parekh, "Plant Leaf Recognition using Shape based Features and Neural Network classifiers," 2011.
[12] R. Hu, W. Jia, H. Ling and D. Huang, "Multiscale Distance Matrix for Fast Plant Leaf".
[13] D. A. Bashish, M. Braik and S. Bani-Ahmad, "Detection and Classification of Leaf Diseases using K-means-based Segmentation and Neural-networks-based Classification," 2011. "
[14] N. J. Nilsson, Introduction to Machine Learning, 1998.
[15] "Introduction to Pattern Recognition," [Online]. Available: http://www.cedar.buffalo.edu/~srihari/CSE555/Chap1.Part1.pdf. [Accessed 2012].
[16] "What is Pattern Recognition," vol. 25, 2003.

[17] "Pattern Recognition," [Online]. Available: http://isa.umh.es/asignaturas/cscs/PR/1%20-%20Pattern%20recognition.pdf. [Accessed 2012].
[18] L. Zheng and X. He, "Classification Techniques in Pattern Recognition," 2007.
[19] "Leaves and Leaf Anatomy," 2003-2012. [Online]. Available: http://www.enchantedlearning.com/subjects/plants/leaf/. [Accessed 2012].
[20] "Botanical-online," 1999-2011. [Online]. Available: http://www.botanical-online.com/hojastiposangles.htm. [Accessed 2012].
[21] "Woody Plant Identification System," 2012. [Online]. Available: http://oregonstate.edu/dept/ldplants/plant_ident/plant/search/type/1. [Accessed 2012].
[22] "Plant Identification: Examining Leaves," 1999-2012. [Online]. Available: http://oregonstate.edu/dept/ldplants/Plant%20ID-Leaves.htm. [Accessed 2012].
[23] "The Parts of a Leaf," 17 August 2009. [Online]. Available: http://www.robinsonlibrary.com/science/botany/anatomy/leafparts.htm. [Accessed October 2012].
[24] "Leaf-Venation," 2012. [Online]. Available: http://science.jrank.org/pages/3869/Leaf-Venation.html. [Accessed 2012].
[25] L. Keyes and A. Winstanley, "USING MOMENT INVARIANTS FOR CLASSIFYING SHAPES ON LARGE SCALE MAPS".

[26] "Canny Edge Detection," 2009.
[27] L. Noriega, "Multilayer Perceptron Tutorial," 2005.
[28] "Multilayer Perceptron," [Online]. Available: http://neuroph.sourceforge.net/tutorials/MultiLayerPerceptron.html. [Accessed 2012].
[29] "Skeleton Leaves," [Online]. Available: http://levasseurr082.angelfire.com/venation.html. [Accessed October 2012].
[30] "How to use the key to identify a twig of Elder," [Online]. Available: http://www-saps.plantsci.cam.ac.uk/trees/elderintro.htm. [Accessed October 2012].
[31] "Leaf Shapes Database," [Online]. Available: http://www.imageprocessingplace.com/downloads_V3/root_downloads/image_databases/leaf%20shape%20database/leaf_shapes_downloads.htm. [Accessed October 2012].

X. Appendix

XI. **CannyEdgeDetector.java**

```
XII.   package leafRec;
XIII.  import java.awt.image.BufferedImage;
XIV.   import java.util.ArrayList;
XV.    import java.util.Arrays;
XVI.
XVII.
```

```
/**
 * <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;
```

```
LXV.    private int[][] pixelArr; // SOSA
LXVI.   private int[] data;
LXVII.  private int[] magnitude;
LXVIII.         private BufferedImage sourceImage;
LXIX.   private BufferedImage edgesImage;
LXX.
LXXI.   private float gaussianKernelRadius;
LXXII.  private float lowThreshold;
LXXIII.         private float highThreshold;
LXXIV.          private int gaussianKernelWidth;
LXXV.   private boolean contrastNormalized;
LXXVI.
LXXVII.         private float[] xConv;
LXXVIII.        private float[] yConv;
LXXIX.          private float[] xGradient;
LXXX.           private float[] yGradient;
LXXXI.
LXXXII.         // constructors
LXXXIII.
LXXXIV.         /**
LXXXV.           * Constructs a new detector with default parameters.
LXXXVI.          */
LXXXVII.
LXXXVIII.               public CannyEdgeDetector() {
LXXXIX.                 lowThreshold = 2.5f;
XC.             highThreshold = 7.5f;
XCI.            gaussianKernelRadius = 2f;
XCII.           gaussianKernelWidth = 16;
XCIII.          contrastNormalized = false;
XCIV.   }
XCV.
XCVI.   // accessors
XCVII.
XCVIII.         /**
XCIX.    * The image that provides the luminance data used by this detector to
C.      * generate edges.
CI.     *
CII.     * @return the source image, or null
CIII.    */
CIV.
CV.     public BufferedImage getSourceImage() {
CVI.            return sourceImage;
CVII.   }
CVIII.
CIX.    /**
CX.      * Specifies the image that will provide the luminance data in which edges
CXI.     * 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;
}
```

```
CCLIII.
CCLIV.          /**
CCLV.    * Sets whether the contrast is normalized
CCLVI.          * @param contrastNormalized true if the contrast should be normalized,
CCLVII.         * false otherwise
CCLVIII.        */
CCLIX.
CCLX.    public void setContrastNormalized(boolean contrastNormalized) {
CCLXI.                  this.contrastNormalized = contrastNormalized;
CCLXII.         }
CCLXIII.
CCLXIV.         // methods
CCLXV.          //modification
CCLXVI.         public void preprocess(){
CCLXVII.                width = sourceImage.getWidth();
CCLXVIII.                       height = sourceImage.getHeight();
CCLXIX.                 picsize = width * height;
CCLXX.                  initArrays();
CCLXXI.                 readLuminance();
CCLXXII.
CCLXXIII.               }
CCLXXIV.                public void process() {
CCLXXV.                 width = sourceImage.getWidth();
CCLXXVI.                        height = sourceImage.getHeight();
CCLXXVII.                       picsize = width * height;
CCLXXVIII.                      initArrays();
CCLXXIX.                        readLuminance();
CCLXXX.                 if (contrastNormalized) normalizeContrast();
CCLXXXI.                        computeGradients(gaussianKernelRadius, gaussianKernelWidth);
CCLXXXII.                       int low = Math.round(lowThreshold * MAGNITUDE_SCALE);
CCLXXXIII.                      int high = Math.round( highThreshold * MAGNITUDE_SCALE);
CCLXXXIV.                       performHysteresis(low, high);
CCLXXXV.                        thresholdEdges();
CCLXXXVI.                       writeEdges(data);
CCLXXXVII.              }
CCLXXXVIII.
CCLXXXIX.               // private utility methods
CCXC.    private void initArrays() {
CCXCI.                  if (data == null || picsize != data.length) {
CCXCII.                         data = new int[picsize];
CCXCIII.                        magnitude = new int[picsize];
CCXCIV.
CCXCV.                          xConv = new float[picsize];
CCXCVI.                         yConv = new float[picsize];
CCXCVII.                        xGradient = new float[picsize];
CCXCVIII.                               yGradient = new float[picsize];
CCXCIX.                 }
```

```
CCC.        }
CCCI.
CCCII.      private void computeGradients(float kernelRadius, int kernelWidth) {
CCCIII.
CCCIV.                      //generate the gaussian convolution masks
CCCV.               float kernel[] = new float[kernelWidth];
CCCVI.                      float diffKernel[] = new float[kernelWidth];
CCCVII.                     int kwidth;
CCCVIII.                    for (kwidth = 0; kwidth < kernelWidth; kwidth++) {
CCCIX.                              float g1 = gaussian(kwidth, kernelRadius);
CCCX.                       if (g1 <= GAUSSIAN_CUT_OFF && kwidth >= 2) break;
CCCXI.                              float g2 = gaussian(kwidth - 0.5f, kernelRadius);
CCCXII.                             float g3 = gaussian(kwidth + 0.5f, kernelRadius);
CCCXIII.                            kernel[kwidth] = (g1 + g2 + g3) / 3f / (2f * (float) Math.PI * kernelRadius *
        kernelRadius);
CCCXIV.                             diffKernel[kwidth] = g3 - g2;
CCCXV.                      }
CCCXVI.
CCCXVII.                    int initX = kwidth - 1;
CCCXVIII.                           int maxX = width - (kwidth - 1);
CCCXIX.                     int initY = width * (kwidth - 1);
CCCXX.                      int maxY = width * (height - (kwidth - 1));
CCCXXI.
CCCXXII.                    //perform convolution in x and y directions
CCCXXIII.                           for (int x = initX; x < maxX; x++) {
CCCXXIV.                                    for (int y = initY; y < maxY; y += width) {
CCCXXV.                                     int index = x + y;
CCCXXVI.                                            float sumX = data[index] * kernel[0];
CCCXXVII.                                           float sumY = sumX;
CCCXXVIII.                                          int xOffset = 1;
CCCXXIX.                                            int yOffset = width;
CCCXXX.                                             for(; xOffset < kwidth ;) {
CCCXXXI.                                                    sumY += kernel[xOffset] * (data[index - yOffset] +
        data[index + yOffset]);
CCCXXXII.                                                   sumX += kernel[xOffset] * (data[index - xOffset] +
        data[index + xOffset]);
CCCXXXIII.                                                  yOffset += width;
CCCXXXIV.                                                   xOffset++;
CCCXXXV.                                            }
CCCXXXVI.
CCCXXXVII.                                          yConv[index] = sumY;
CCCXXXVIII.                                         xConv[index] = sumX;
CCCXXXIX.                                   }
CCCXL.
CCCXLI.                     }
CCCXLII.
CCCXLIII.                           for (int x = initX; x < maxX; x++) {
CCCXLIV.                                    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
```

```
CCCXCI.                                 float nMag = hypot(xGradient[indexN], yGradient[indexN]);
CCCXCII.                                float sMag = hypot(xGradient[indexS], yGradient[indexS]);
CCCXCIII.                                       float wMag = hypot(xGradient[indexW], yGradient[indexW]);
CCCXCIV.                                        float eMag = hypot(xGradient[indexE], yGradient[indexE]);
CCCXCV.                                 float neMag = hypot(xGradient[indexNE], yGradient[indexNE]);
CCCXCVI.                                        float seMag = hypot(xGradient[indexSE], yGradient[indexSE]);
CCCXCVII.                                       float swMag = hypot(xGradient[indexSW], yGradient[indexSW]);
CCCXCVIII.                                      float nwMag = hypot(xGradient[indexNW], yGradient[indexNW]);
CCCXCIX.                                        float tmp;
CD.
CDI.                                    if (xGrad * yGrad <= (float) 0 /*(1)*/
CDII.                                           ? Math.abs(xGrad) >= Math.abs(yGrad) /*(2)*/
CDIII.                                                  ? (tmp = Math.abs(xGrad * gradMag)) >= Math.abs(yGrad * neMag
        - (xGrad + yGrad) * eMag) /*(3)*/
CDIV.                                                           && tmp > Math.abs(yGrad * swMag - (xGrad + yGrad)
        * wMag) /*(4)*/
CDV.                                                    : (tmp = Math.abs(yGrad * gradMag)) >= Math.abs(xGrad * neMag
        - (yGrad + xGrad) * nMag) /*(3)*/
CDVI.                                                           && tmp > Math.abs(xGrad * swMag - (yGrad + xGrad)
        * sMag) /*(4)*/
CDVII.                                                  : Math.abs(xGrad) >= Math.abs(yGrad) /*(2)*/
CDVIII.                                                         ? (tmp = Math.abs(xGrad * gradMag)) >=
        Math.abs(yGrad * seMag + (xGrad - yGrad) * eMag) /*(3)*/
CDIX.                                                           && tmp > Math.abs(yGrad * nwMag + (xGrad - yGrad)
        * wMag) /*(4)*/
CDX.                                                    : (tmp = Math.abs(yGrad * gradMag)) >= Math.abs(xGrad * seMag
        + (yGrad - xGrad) * sMag) /*(3)*/
CDXI.                                                           && tmp > Math.abs(xGrad * nwMag + (yGrad - xGrad)
        * nMag) /*(4)*/
CDXII.                                                  ) {
CDXIII.                                                 magnitude[index] = gradMag >= MAGNITUDE_LIMIT ?
        MAGNITUDE_MAX : (int) (MAGNITUDE_SCALE * gradMag);
CDXIV.                                  } else {
CDXV.                                   magnitude[index] = 0;
CDXVI.                                  }
CDXVII.                                 }
CDXVIII.                        }
CDXIX.          }
CDXX.
CDXXI.          private float hypot(float x, float y) {
CDXXII.                 return (float) Math.hypot(x, y);
CDXXIII.        }
CDXXIV.
CDXXV.          private float gaussian(float x, float sigma) {
CDXXVI.                 return (float) Math.exp(-(x * x) / (2f * sigma * sigma));
CDXXVII.                }
CDXXVIII.
CDXXIX.         private void performHysteresis(int low, int high) {
CDXXX.                  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;
        }
```

```
CDLXXVII.
CDLXXVIII.                  //modification
CDLXXIX.                    public int getCentroidY(){
CDLXXX.                             return centroidY;
CDLXXXI.                    }
CDLXXXII.
CDLXXXIII.                  //modification
CDLXXXIV.                   public int[][] get2DPixels(){
CDLXXXV.                            return pixelArr;
CDLXXXVI.                   }
CDLXXXVII.
CDLXXXVIII.                 //modification
CDLXXXIX.                   public int[] getData(){
CDXC.                       return data;
CDXCI.          }
CDXCII.
CDXCIII.        //modification
CDXCIV.         public ArrayList<Integer> getOutline(){
CDXCV.                      return outline;
CDXCVI.         }
CDXCVII.
CDXCVIII.                   //modification-inverted colors
CDXCIX.         private void thresholdEdges() {
D.      int sumX=0;
DI.     int sumY=0;
DII.            int blackPixelsCount=0;
DIII.           pixelArr = new int[width][height];
DIV.            for (int i = 0; i < picsize; i++) {
DV.                         data[i] = data[i] > 0 ? 0xff000000 : -1 ;
DVI.                        pixelArr[i%width][i/width] = data[i];
DVII.                       if(data[i] == 0xff000000){
DVIII.                              sumX += i%width;
DIX.                                sumY += i/width;
DX.                                 outline.add(i);
DXI.                                ++blackPixelsCount;
DXII.                       }
DXIII.          }
DXIV.           data[114*115] = 55;
DXV.            centroidX = sumX/blackPixelsCount;
DXVI.           centroidY = sumY/blackPixelsCount;
DXVII.          }
DXVIII.
DXIX.   private int luminance(float r, float g, float b) {
DXX.            return Math.round(0.299f * r + 0.587f * g + 0.114f * b);
DXXI.   }
DXXII.
DXXIII.         public void readLuminance() {
```

```
DXXIV.                          int type = sourceImage.getType();
DXXV.                           if (type == BufferedImage.TYPE_INT_RGB || type == BufferedImage.TYPE_INT_ARGB) {
DXXVI.                                  int[] pixels = (int[]) sourceImage.getData().getDataElements(0, 0, width, height, null);
DXXVII.                                 for (int i = 0; i < picsize; i++) {
DXXVIII.                                        int p = pixels[i];
DXXIX.                                          int r = (p & 0xff0000) >> 16;
DXXX.                                           int g = (p & 0xff00) >> 8;
DXXXI.                                          int b = p & 0xff;
DXXXII.                                         data[i] = luminance(r, g, b);
DXXXIII.                                }
DXXXIV.                         } else if (type == BufferedImage.TYPE_BYTE_GRAY) {
DXXXV.                                  byte[] pixels = (byte[]) sourceImage.getData().getDataElements(0, 0, width, height, null);
DXXXVI.                                 for (int i = 0; i < picsize; i++) {
DXXXVII.                                                data[i] = (pixels[i] & 0xff);
DXXXVIII.                               }
DXXXIX.                         } else if (type == BufferedImage.TYPE_USHORT_GRAY) {
DXL.                            short[] pixels = (short[]) sourceImage.getData().getDataElements(0, 0, width, height, null);
DXLI.                           for (int i = 0; i < picsize; i++) {
DXLII.                                  data[i] = (pixels[i] & 0xffff) / 256;
DXLIII.                                 }
DXLIV.                          } else if (type == BufferedImage.TYPE_3BYTE_BGR) {
DXLV.       byte[] pixels = (byte[]) sourceImage.getData().getDataElements(0, 0, width, height, null);
DXLVI.              int offset = 0;
DXLVII.             for (int i = 0; i < picsize; i++) {
DXLVIII.               int b = pixels[offset++] & 0xff;
DXLIX.                 int g = pixels[offset++] & 0xff;
DL.                 int r = pixels[offset++] & 0xff;
DLI.                data[i] = luminance(r, g, b);
DLII.            }
DLIII.      } else {
DLIV.                           throw new IllegalArgumentException("Unsupported image type: " + type);
DLV.                    }
DLVI.       }
DLVII.
DLVIII.                 private void normalizeContrast() {
DLIX.                   int[] histogram = new int[256];
DLX.                    for (int i = 0; i < data.length; i++) {
DLXI.                           histogram[data[i]]++;
DLXII.                  }
DLXIII.                         int[] remap = new int[256];
DLXIV.                          int sum = 0;
DLXV.                   int j = 0;
DLXVI.                          for (int i = 0; i < histogram.length; i++) {
DLXVII.                                 sum += histogram[i];
DLXVIII.                                int target = sum*255/picsize;
DLXIX.                                  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);
        }
        }

        package leafRec;
import java.util.ArrayList;

public class CentroidRadii {
                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){
```

CentroidRadii.java

```
DCXVI.                  x -= centroidX;
DCXVII.                 y -= centroidY;
DCXVIII.                return Math.sqrt(x*x+y*y);
DCXIX.          }
DCXX.
DCXXI.          //get set of radii
DCXXII.         public ArrayList<Double> getRadiiSet(){
DCXXIII.                return radii;
DCXXIV.         }
DCXXV.
DCXXVI.         //normalize radii
DCXXVII.                public void normalizeArray(){
DCXXVIII.                       double longest = 0;
DCXXIX.                 for(int i = 0; i < 4; i++){
DCXXX.                          for(int j = 0; j < 9; j++){
DCXXXI.                                 radii.add(degRadii[i][j]);
DCXXXII.                                        longest = degRadii[i][j] > longest ? degRadii[i][j] : longest;
DCXXXIII.                               }
DCXXXIV.                        }
DCXXXV.                 longest = longest > 0 ? longest : 1;
DCXXXVI.                        for(int i = 0; i < radii.size(); i++){
DCXXXVII.                               radii.set(i, radii.get(i)/longest);
DCXXXVIII.                      }
DCXXXIX.                }
DCXL.
DCXLI.          //get quadrant of point with respect to origin(centroidX, centroidY)
DCXLII.         public int getQuadrant(int x, int y){
DCXLIII.                if(x<centroidX){
DCXLIV.                         if(y < centroidY){
DCXLV.                                  return 2;
DCXLVI.                         }else if(y >= centroidY){
DCXLVII.                                return 3;
DCXLVIII.                               }
DCXLIX.                 }else if(x>centroidX){
DCL.                    if(y <= centroidY){
DCLI.                           return 1;
DCLII.                  }else if(y > centroidY){
DCLIII.                                 return 4;
DCLIV.                          }
DCLV.           }else{ // 90 degrees, 270 degrees
DCLVI.                          if(y < centroidY){
DCLVII.                                 return 2;
DCLVIII.                        }else if(y > centroidY){
DCLIX.                                  return 4;
DCLX.                   }
DCLXI.                  }
```

```
DCLXII.                 return 0;
DCLXIII.        }
DCLXIV.
DCLXV.          //initialize 4 x 9 array
DCLXVI.         public void initDeg(){
DCLXVII.                degRadii = new double[4][9];
DCLXVIII.                       degrees = new double[4][9];
DCLXIX.                 for(int i = 0; i < 4; i++){
DCLXX.                          for(int j = 0; j < 9; j++){
DCLXXI.                                 degRadii[i][j] = 1.0;
DCLXXII.                                degrees[i][j] = 10.0*(j+1);
DCLXXIII.                               }
DCLXXIV.                        }
DCLXXV.                 degrees[1][0] = 100.0;
DCLXXVI.                        degrees[3][0] = 100.0;
DCLXXVII.
DCLXXVIII.              }
DCLXXIX.
DCLXXX.                 //update degrees
DCLXXXI.                public void updateDeg(double deg, int x, int y){
DCLXXXII.                       int tempDeg = 0;
DCLXXXIII.                      tempDeg = (int) (Math.round(deg/10)*10);
DCLXXXIV.                       int xIndex = getQuadrant(x,y) <= 0? 0 : getQuadrant(x,y)-1;
DCLXXXV.                        int yIndex = tempDeg == 90 ? 0 : tempDeg/10;
DCLXXXVI.                       if(tempDeg == 90 && xIndex <2) {
DCLXXXVII.                              xIndex = 1;
DCLXXXVIII.                     }else if(tempDeg == 90 && xIndex >1) {
DCLXXXIX.                               xIndex = 3;
DCXC.                   }else if(tempDeg == 0 && (xIndex == 0 || x == 3)) {
DCXCI.                          xIndex = 0;
DCXCII.                 }else if(tempDeg == 0 && (xIndex == 1 || x == 2)) {
DCXCIII.                        xIndex = 2;
DCXCIV.                 }
DCXCV.                  double d = degrees[xIndex][yIndex];
DCXCVI.                 if(Math.abs(tempDeg-deg) <= Math.abs(d - tempDeg)){
DCXCVII.                                degrees[xIndex][yIndex] = deg;
DCXCVIII.                               degRadii[xIndex][yIndex] = getDistance(x,y);
DCXCIX.                 }
DCC.    }
DCCI.
DCCII.  //obtain 36 points
DCCIII.         public void determineRadii(){
DCCIV.                  initDeg();
DCCV.           for(int i = 0; i < outline.size(); i++){
DCCVI.                          int x = outline.get(i)%width;
DCCVII.                         int y = outline.get(i)/width;
```

```
                                Double slope =  Math.abs((y-centroidY)/(x-centroidX));
                                Double deg = Math.toDegrees(Math.atan((slope)));
                                Double d = (double) (Math.round(deg/10)*10);
                                if(Math.round(d) % 10 == 0 || Math.ceil(d) %10 == 0 || Math.floor(d) %10 == 0){
                                        updateDeg(deg,x,y);
                                }
                        }
                        normalizeArray();
                }
        }

ErrorGraph.java
package leafRec;
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
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);
   }
```

```
DCCLV.
DCCLVI.             //draw numerical labels
DCCLVII.            int i = 0;
DCCLVIII.                   for(int x = 70; x < width; x+=50, i++){
DCCLIX.                             g2D.drawString((i*50)+"",x,height-35);
DCCLX.              }
DCCLXI.
DCCLXII.            //plot points(errors)
DCCLXIII.                   g2D.setColor(Color.red);
DCCLXIV.                    for(i = 0; i < width-70 && i < err.size(); i++){
DCCLXV.                 int x = i+70;
DCCLXVI.                            double y = 250-err.get(i)*2;
DCCLXVII.                                   g2D.drawLine(x,(int)y,x,(int)y);
DCCLXVIII.                  }
DCCLXIX.
DCCLXX.         }
DCCLXXI.        }
DCCLXXII.
DCCLXXIII.      ExpertGUI.java
DCCLXXIV.       package leafRec;
DCCLXXV.
DCCLXXVI.       import java.awt.BorderLayout;
DCCLXXVII.      import java.awt.Color;
DCCLXXVIII.     import java.awt.Dimension;
DCCLXXIX.       import java.awt.GridLayout;
DCCLXXX.        import java.awt.Image;
DCCLXXXI.       import java.awt.Toolkit;
DCCLXXXII.      import java.awt.event.ActionEvent;
DCCLXXXIII.     import java.awt.event.ActionListener;
DCCLXXXIV.      import java.awt.image.BufferedImage;
DCCLXXXV.       import java.io.BufferedReader;
DCCLXXXVI.      import java.io.ByteArrayInputStream;
DCCLXXXVII.     import java.io.File;
DCCLXXXVIII.    import java.io.FileNotFoundException;
DCCLXXXIX.      import java.io.FileOutputStream;
DCCXC.  import java.io.FileReader;
DCCXCI. import java.io.IOException;
DCCXCII.import java.nio.file.Files;
DCCXCIII.       import java.nio.file.Path;
DCCXCIV.        import java.nio.file.Paths;
DCCXCV. import java.sql.Blob;
DCCXCVI.        import java.sql.SQLException;
DCCXCVII.       import java.util.ArrayList;
DCCXCVIII.      import java.util.StringTokenizer;
DCCXCIX.
DCCC.import javax.imageio.ImageIO;
DCCCI.  import javax.sql.rowset.serial.SerialBlob;
```

```
DCCCII.  import javax.swing.*;
DCCCIII. import javax.swing.border.Border;
DCCCIV.  import javax.swing.border.CompoundBorder;
DCCCV.   import javax.swing.border.EmptyBorder;
DCCCVI.  import javax.swing.border.LineBorder;
DCCCVII. import javax.swing.event.ListSelectionEvent;
DCCCVIII.        import javax.swing.event.ListSelectionListener;
DCCCIX.
DCCCX.   import org.neuroph.core.events.LearningEvent;
DCCCXI.  import org.neuroph.core.events.LearningEventListener;
DCCCXII. import org.neuroph.core.learning.DataSet;
DCCCXIII.        import org.neuroph.core.learning.DataSetRow;
DCCCXIV. import org.neuroph.core.learning.LearningRule;
DCCCXV.  import org.neuroph.nnet.MultiLayerPerceptron;
DCCCXVI. import org.neuroph.nnet.learning.BackPropagation;
DCCCXVII.        import org.neuroph.util.TransferFunctionType;
DCCCXVIII.
DCCCXIX. public class ExpertGUI implements ActionListener, LearningEventListener{
DCCCXX.          //final vars
DCCCXXI.         private final int WIDTH = 800;
DCCCXXII.        private final int HEIGHT = 450;
DCCCXXIII.
DCCCXXIV.        //epochs
DCCCXXV.         private int epochs = 0;
DCCCXXVI.
DCCCXXVII.       //canny
DCCCXXVIII.      private float low = 0.1f;
DCCCXXIX.        private float high = 10f;
DCCCXXX.
DCCCXXXI.        //expected output
DCCCXXXII.       private double[] out;
DCCCXXXIII.
DCCCXXXIV.       //error
DCCCXXXV.        private ArrayList<Double> errors = new ArrayList<Double>();
DCCCXXXVI.
DCCCXXXVII.      //login details
DCCCXXXVIII.     private static String url;
DCCCXXXIX.       private static String user;
DCCCXL.  private static String pass;
DCCCXLI.
DCCCXLII.        //image
DCCCXLIII.       private String img;
DCCCXLIV.
DCCCXLV.         //flag for stopping network from learning
DCCCXLVI.        private Boolean isStop =false;
DCCCXLVII.
DCCCXLVIII.      //main frame
```

```
DCCCXLIX.               private JFrame frame = new JFrame("Plant Leaf Recognition - Expert");
DCCCL.          private JTabbedPane tabbedPane = new JTabbedPane();
DCCCLI.
DCCCLII.        //menu
DCCCLIII.               JMenuBar menu = new JMenuBar();
DCCCLIV.                JMenu canny = new JMenu("Canny");
DCCCLV.         JMenu net = new JMenu("Network");
DCCCLVI.                JMenu exit = new JMenu("Exit");
DCCCLVII.               JMenuItem lowT = new JMenuItem("Low Threshold");
DCCCLVIII.              JMenuItem highT = new JMenuItem("High Threshold");
DCCCLIX.                JMenuItem detect = new JMenuItem("Detect Edge");
DCCCLX.         JMenuItem saveNet = new JMenuItem("Save");
DCCCLXI.                JMenuItem upNet = new JMenuItem("Upload");
DCCCLXII.               JMenuItem close = new JMenuItem("Close");
DCCCLXIII.
DCCCLXIV.               //panels OR tabs
DCCCLXV.                private JPanel sPanel = new JPanel(new BorderLayout());
DCCCLXVI.               private JPanel nPanel = new JPanel(new BorderLayout());
DCCCLXVII.              private JPanel uPanel = new JPanel(new BorderLayout());
DCCCLXVIII.
DCCCLXIX.               //other panels
DCCCLXX.                private JPanel sLeftPanel = new JPanel(new BorderLayout());
DCCCLXXI.               private JPanel sRightPanel = new JPanel(new BorderLayout());
DCCCLXXII.              private JPanel sLeftImgPanel = new JPanel(new BorderLayout());
DCCCLXXIII.             private JPanel sLeftPropPanel = new JPanel(new BorderLayout());
DCCCLXXIV.              private JPanel sLeftFormPanel = new JPanel(new GridLayout(6,2));
DCCCLXXV.               private JPanel sRightBottomPanel = new JPanel(new GridLayout(1,2));
DCCCLXXVI.
DCCCLXXVII.             private JPanel errorPanel = new ErrorGraph(500,300,errors);
DCCCLXXVIII.            private JPanel errorBottomPanel = new JPanel(new GridLayout(1,2));
DCCCLXXIX.              private JPanel nLeftPanel = new JPanel(new BorderLayout());
DCCCLXXX.               private JPanel nRightPanel = new JPanel(new BorderLayout());
DCCCLXXXI.              private JPanel nSummaryPanel = new JPanel(new GridLayout(4,2));
DCCCLXXXII.             private JPanel nPropPanel = new JPanel(new GridLayout(5,2));
DCCCLXXXIII.            private JPanel nnPanel = new JPanel(new BorderLayout());
DCCCLXXXIV.             private JPanel ncPanel = new JPanel(new BorderLayout());
DCCCLXXXV.              private JPanel nncPanel = new JPanel(new BorderLayout());
DCCCLXXXVI.             private JPanel nsPanel = new JPanel(new GridLayout(3,1));
DCCCLXXXVII.
DCCCLXXXVIII.                   private JPanel leftPanel = new JPanel(new BorderLayout());
DCCCLXXXIX.             private JPanel bPanel = new JPanel(new GridLayout(3,1));
DCCCXC.
DCCCXCI.                //Scroll Pane
DCCCXCII.               private JScrollPane sPane = new JScrollPane();
DCCCXCIII.              private JScrollPane uPane = new JScrollPane();
DCCCXCIV.
```

```
DCCCXCV.                    //labels
DCCCXCVI.                   private JLabel summary = new JLabel("Summary");
DCCCXCVII.                  private JLabel netProp = new JLabel("Network");
DCCCXCVIII.                 private JLabel imgCount = new JLabel("0");
DCCCXCIX.                   private JLabel specCount = new JLabel("0");
CM.         private JLabel epochCount = new JLabel("0");
CMI.        private JLabel netError = new JLabel("0.0");
CMII.       private JLabel uImg = new JLabel("Image Preview");
CMIII.
CMIV.       //list model
CMV.        private DefaultListModel<String> uListModel = new DefaultListModel<String>();
CMVI.
CMVII.              //list
CMVIII.             private JList<String> uList = new JList<String>(uListModel);
CMIX.
CMX.        //text field
CMXI.       private JTextField sName = new JTextField("");
CMXII.              private JTextField sType = new JTextField("");
CMXIII.             private JTextField sArr = new JTextField("");
CMXIV.              private JTextField sBlade = new JTextField("");
CMXV.               private JTextField sMargin = new JTextField("");
CMXVI.              private JTextField sVena= new JTextField("");
CMXVII.
CMXVIII.            private JTextField input = new JTextField("40");
CMXIX.              private JTextField hidden = new JTextField("20");
CMXX.               private JTextField output = new JTextField("1");
CMXXI.              private JTextField learningRate = new JTextField("0.5");
CMXXII.             private JTextField maxError = new JTextField("0.01");
CMXXIII.
CMXXIV.             //buttons
CMXXV.              private JButton sLeftImg = new JButton("Image Preview");
CMXXVI.             private JButton sRightImg = new JButton("Image Preview");
CMXXVII.                    private JButton sLoad = new JButton("Load Image");
CMXXVIII.                   private JButton sNew = new JButton("New");
CMXXIX.             private JButton sExisting = new JButton("Existing");
CMXXX.              private JButton sClear = new JButton("Clear");
CMXXXI.
CMXXXII.                    private JButton train = new JButton("Train");
CMXXXIII.                   private JButton stopTraining = new JButton("Stop Learning");
CMXXXIV.                    private JButton nClear = new JButton("Clear");
CMXXXV.                     private JButton errGraph = new JButton("Generate error graph");
CMXXXVI.                    private JButton errClear = new JButton("Reset graph");
CMXXXVII.
CMXXXVIII.                  private JButton update = new JButton("Update list");
CMXXXIX.                    private JButton download = new JButton("Download file");
CMXL.               private JButton delete = new JButton("Delete");
```

```
CMXLI.
CMXLII.          //Thread
CMXLIII.         @SuppressWarnings("rawtypes")
CMXLIV.          private SwingWorker w;
CMXLV.
CMXLVI.          //network
CMXLVII.                 private MultiLayerPerceptron m = null;
CMXLVIII.
CMXLIX.          //constructor
CML.     @SuppressWarnings("static-access")
CMLI.    public ExpertGUI(String url, String user, String pass){
CMLII.                   this.url = url;
CMLIII.                  this.user = user;
CMLIV.                   this.pass = pass;
CMLV.    }
CMLVI.
CMLVII.          //create frame
CMLVIII.         public void createFrame() throws ClassNotFoundException, SQLException, IOException{
CMLIX.                   tabbedPane.add("Species", sPanel);
CMLX.                    tabbedPane.add("Neural Network", nPanel);
CMLXI.                   tabbedPane.add("Unknown", uPanel);
CMLXII.
CMLXIII.                 menu.add(canny);
CMLXIV.                  menu.add(net);
CMLXV.                   menu.add(exit);
CMLXVI.
CMLXVII.                         canny.add(lowT);
CMLXVIII.                        canny.add(highT);
CMLXIX.                  canny.add(detect);
CMLXX.                   net.add(saveNet);
CMLXXI.                  net.add(upNet);
CMLXXII.                         exit.add(close);
CMLXXIII.
CMLXXIV.                         lowT.addActionListener(this);
CMLXXV.                          highT.addActionListener(this);
CMLXXVI.                         detect.addActionListener(this);
CMLXXVII.                        saveNet.addActionListener(this);
CMLXXVIII.                       upNet.addActionListener(this);
CMLXXIX.                         close.addActionListener(this);
CMLXXX.
CMLXXXI.                         initAS();
CMLXXXII.                        initNN();
CMLXXXIII.                       initU();
CMLXXXIV.
CMLXXXV.                         createASTab();
CMLXXXVI.                        createNNTab();
```

```
                createUTab();
                Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
            JWindow window = new JWindow();
        ImageFade fade = new ImageFade("img/expert.gif",0);
        window.getContentPane().add(fade, SwingConstants.CENTER);
        window.setBounds(screenSize.width/2 - (510/2), screenSize.height/2 - (176/2),510,176);
        window.setVisible(true);
        try {
            Thread.sleep(5000);
                } catch (InterruptedException e) {
                    System.out.println("An error occurred.");
        }
window.setVisible(false);
                frame.add(tabbedPane);
                frame.setJMenuBar(menu);
                frame.pack();
                frame.setVisible(true);
                frame.setSize(WIDTH,HEIGHT);
                frame.setResizable(false);
                frame.setLocation(screenSize.width/2 - (WIDTH/2), screenSize.height/2 - (HEIGHT/2));
                frame.setIconImage(new ImageIcon(getClass().getResource("img/leafIcon.png")).getImage());
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                window.dispose();
    }

    //initialize vars in tab for adding species
    public void initAS(){
                Border line = new LineBorder(Color.BLACK);
                Border margin = new EmptyBorder(5, 15, 5, 15);
                    Border compound = new CompoundBorder(line, margin);

                sLoad.addActionListener(this);
                sNew.addActionListener(this);
                sExisting.addActionListener(this);
                    sClear.addActionListener(this);

                    sNew.setEnabled(false);
                    sExisting.setEnabled(false);
                    detect.setEnabled(false);
                    sLeftImg.setContentAreaFilled(false);
                    sLeftImg.setOpaque(false);
                    sLeftImg.setFocusable(false);
                    sLeftImg.setBorderPainted(true);
                    sLeftImg.setForeground(Color.BLACK);
                    sLeftImg.setBackground(Color.WHITE);
                    sLeftImg.setBorder(compound);
```

```
MXXXIV.
MXXXV.                  sRightImg.setContentAreaFilled(false);
MXXXVI.                 sRightImg.setOpaque(false);
MXXXVII.                        sRightImg.setFocusable(false);
MXXXVIII.                       sRightImg.setBorderPainted(true);
MXXXIX.                 sRightImg.setForeground(Color.BLACK);
MXL.            sRightImg.setBackground(Color.WHITE);
MXLI.           sRightImg.setBorder(compound);
MXLII.
MXLIII.                 sLeftPanel.setBackground(Color.WHITE);
MXLIV.                  sLeftImgPanel.setBackground(Color.WHITE);
MXLV.           sLeftFormPanel.setBackground(Color.WHITE);
MXLVI.                  sRightPanel.setBackground(Color.WHITE);
MXLVII.                 sRightBottomPanel.setBackground(Color.WHITE);
MXLVIII.        }
MXLIX.
ML.     //create tab for adding species
MLI.    public void createASTab(){
MLII.           sLeftPanel.add(sLeftImgPanel, BorderLayout.CENTER);
MLIII.          sLeftPanel.add(sLeftPropPanel, BorderLayout.SOUTH);
MLIV.
MLV.            sLeftPropPanel.add(sLeftFormPanel, BorderLayout.CENTER);
MLVI.           sLeftPropPanel.add(sClear, BorderLayout.SOUTH);
MLVII.
MLVIII.                 sLeftImgPanel.add(sLeftImg, BorderLayout.CENTER);
MLIX.           sLeftImgPanel.add(sLoad, BorderLayout.SOUTH);
MLX.
MLXI.           sRightBottomPanel.add(sNew);
MLXII.                  sRightBottomPanel.add(sExisting);
MLXIII.
MLXIV.                  sRightPanel.add(sRightImg, BorderLayout.CENTER);
MLXV.                   sRightPanel.add(sRightBottomPanel, BorderLayout.SOUTH);
MLXVI.
MLXVII.                 sLeftFormPanel.add(new JLabel("Leaf Name: "));
MLXVIII.                sLeftFormPanel.add(sName);
MLXIX.                  sLeftFormPanel.add(new JLabel("Leaf Type: "));
MLXX.                   sLeftFormPanel.add(sType);
MLXXI.                  sLeftFormPanel.add(new JLabel("Leaf Arrangement: "));
MLXXII.                 sLeftFormPanel.add(sArr);
MLXXIII.                sLeftFormPanel.add(new JLabel("Leaf Blade: "));
MLXXIV.                 sLeftFormPanel.add(sBlade);
MLXXV.                  sLeftFormPanel.add(new JLabel("Leaf Margin: "));
MLXXVI.                 sLeftFormPanel.add(sMargin);
MLXXVII.                        sLeftFormPanel.add(new JLabel("Leaf Venation: "));
MLXXVIII.                       sLeftFormPanel.add(sVena);
MLXXIX.
```

```
MLXXX.                  sPanel.add(sLeftPanel,BorderLayout.WEST);
MLXXXI.                 sPanel.add(sRightPanel,BorderLayout.CENTER);
MLXXXII.                }
MLXXXIII.
MLXXXIV.                //initialize vars in tab for neural network
MLXXXV.                 public void initNN(){
MLXXXVI.                        Border line = new LineBorder(Color.BLACK);
MLXXXVII.                       Border margin = new EmptyBorder(5, 15, 5, 15);
MLXXXVIII.                      Border compound = new CompoundBorder(line, margin);
MLXXXIX.                        input.setEditable(false);
MXC.            output.setEditable(false);
MXCI.           stopTraining.setEnabled(false);
MXCII.                  stopTraining.addActionListener(this);
MXCIII.                 train.addActionListener(this);
MXCIV.                  nClear.addActionListener(this);
MXCV.                   errClear.addActionListener(this);
MXCVI.                  errGraph.addActionListener(this);
MXCVII.
MXCVIII.                summary.setBorder(compound);
MXCIX.                  netProp.setBorder(compound);
MC.             errorPanel.setBorder(compound);
MCI.
MCII.           nPanel.setBackground(Color.WHITE);
MCIII.          nLeftPanel.setBackground(Color.WHITE);
MCIV.           nRightPanel.setBackground(Color.WHITE);
MCV.            nnPanel.setBackground(Color.WHITE);
MCVI.           ncPanel.setBackground(Color.WHITE);
MCVII.                  nsPanel.setBackground(Color.WHITE);
MCVIII.                 nncPanel.setBackground(Color.WHITE);
MCIX.           nSummaryPanel.setBackground(Color.WHITE);
MCX.            nPropPanel.setBackground(Color.WHITE);
MCXI.           errorPanel.setBackground(Color.WHITE);
MCXII.                  sPane.setBackground(Color.WHITE);
MCXIII.                 errorBottomPanel.setBackground(Color.WHITE);
MCXIV.
MCXV.           }
MCXVI.
MCXVII.         //create tab for neural network
MCXVIII.        public void createNNTab() throws ClassNotFoundException, SQLException{
MCXIX.                  nPanel.add(nLeftPanel, BorderLayout.WEST);
MCXX.                   nPanel.add(nRightPanel, BorderLayout.CENTER);
MCXXI.
MCXXII.                 nnPanel.add(summary, BorderLayout.NORTH);
MCXXIII.                nnPanel.add(nSummaryPanel, BorderLayout.CENTER);
MCXXIV.                 nSummaryPanel.add(new JLabel("Leaf Images: "));
MCXXV.                  nSummaryPanel.add(imgCount);
```

```
                nSummaryPanel.add(new JLabel("Leaf Species: "));
                        nSummaryPanel.add(specCount);
                        nSummaryPanel.add(new JLabel("Epochs: "));
                nSummaryPanel.add(epochCount);
                nSummaryPanel.add(new JLabel("Total Network Error: "));
                nSummaryPanel.add(netError);

                        ncPanel.add(netProp, BorderLayout.NORTH);
                        ncPanel.add(nPropPanel);
                        nPropPanel.add(new JLabel("Input: "));
                        nPropPanel.add(input);
                        nPropPanel.add(new JLabel("Hidden: "));
                        nPropPanel.add(hidden);
                        nPropPanel.add(new JLabel("Output: "));
                nPropPanel.add(output);
                nPropPanel.add(new JLabel("Learning Rate: "));
                nPropPanel.add(learningRate);
                nPropPanel.add(new JLabel("Max Error: "));
                nPropPanel.add(maxError);

                nsPanel.add(train);
                        nsPanel.add(stopTraining);
                        nsPanel.add(nClear);

        nncPanel.add(nnPanel, BorderLayout.NORTH);
        nncPanel.add(ncPanel, BorderLayout.SOUTH);

                nLeftPanel.add(nncPanel, BorderLayout.CENTER);
                nLeftPanel.add(nsPanel, BorderLayout.SOUTH);
        errorPanel.add(new JLabel("Error Graph"));
                errorPanel.setPreferredSize(new Dimension(500,300));
                sPane.setViewportView(errorPanel);

                errorBottomPanel.add(errGraph);
                errorBottomPanel.add(errClear);


                nRightPanel.add(sPane, BorderLayout.CENTER);
                nRightPanel.add(errorBottomPanel, BorderLayout.SOUTH);
                updateSummary();
        }

                //initialize unknown tab
        public void initU() throws ClassNotFoundException, SQLException, IOException{
                Border line = new LineBorder(Color.BLACK);
                Border margin = new EmptyBorder(5, 15, 5, 15);
                        Border compound = new CompoundBorder(line, margin);
```

```
MCLXXIII.
MCLXXIV.                    uPanel.setBackground(Color.WHITE);
MCLXXV.                     leftPanel.setBackground(Color.WHITE);
MCLXXVI.                    uImg.setBackground(Color.WHITE);
MCLXXVII.
MCLXXVIII.                  leftPanel.setBorder(compound);
MCLXXIX.                    uImg.setBorder(compound);
MCLXXX.
MCLXXXI.                    update.addActionListener(this);
MCLXXXII.                   download.addActionListener(this);
MCLXXXIII.                  delete.addActionListener(this);
MCLXXXIV.
MCLXXXV.                    displayUSet();
MCLXXXVI.
MCLXXXVII.                  uList.setFixedCellHeight(30);
MCLXXXVIII.         uList.setFixedCellWidth(235);
MCLXXXIX.           uList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
MCXC.       uList.addListSelectionListener(new ListSelectionListener() {
MCXCI.              public void valueChanged(ListSelectionEvent event) {
MCXCII.                     if(uList.getSelectedIndex() > -1){
MCXCIII.                    SQL s = new SQL(url,user,pass);
MCXCIV.                     try {
MCXCV.                                                  s.connect();
MCXCVI.                                                 Blob b =
        s.queryImage(Integer.parseInt(uList.getSelectedValue().substring(8)));
MCXCVII.                                                        byte[] imgData = b.getBytes(1,(int)
        b.length());
MCXCVIII.                                           BufferedImage img = ImageIO.read(new
        ByteArrayInputStream(imgData));
MCXCIX.                                     ImageIcon icon = new ImageIcon(img);
MCC.                                          Image image = icon.getImage().getScaledInstance(350, 350,
        Image.SCALE_SMOOTH);
MCCI.                                         icon.setImage(image);
MCCII.                                            uImg.setIcon(icon);
MCCIII.                                     uImg.setText("");
MCCIV.                                          } catch (ClassNotFoundException | SQLException | IOException e)
        {
MCCV.                                                   JOptionPane.showMessageDialog(frame,"An error
        occurred!");
MCCVI.                                          }
MCCVII.
MCCVIII.                    }
MCCIX.          }});
MCCX.       uPane.setViewportView(uList);
MCCXI.      uPane.setBorder(null);
MCCXII.             }
MCCXIII.
MCCXIV.             //display training set collection
```

```
public void displayUSet() throws ClassNotFoundException, SQLException, IOException{
        uListModel.clear();
                ArrayList<LeafImage> leaves = new ArrayList<LeafImage>();
                SQL s = new SQL(url, user, pass);
        s.connect();
        leaves = s.queryImages();
        for(LeafImage l: leaves){
                        uListModel.addElement("Unknown "+l.getCaseNo());
                }
        }

        //create unknown tab
        public void createUTab(){
                uPanel.add(leftPanel, BorderLayout.WEST);
                uPanel.add(uImg, BorderLayout.CENTER);

                leftPanel.add(uPane,BorderLayout.CENTER);
                leftPanel.add(bPanel,BorderLayout.SOUTH);

                bPanel.add(update);
                bPanel.add(download);
                bPanel.add(delete);
        }

        //listener
public void actionPerformed(ActionEvent e) {
        if(e.getSource() == sLoad){
                        JFileChooser chooser = new JFileChooser();
                        try {
                                File f = new File(new File("Choose file").getCanonicalPath());
                    chooser.setSelectedFile(f);
                          int result = chooser.showOpenDialog(frame);
                          if(result == JFileChooser.APPROVE_OPTION){
                                        File curFile = chooser.getSelectedFile();
                            img = "";
                          img = curFile.toString();
                          int index = img.indexOf(".");
                          if(img.substring(index+1).equals("jpg")){
                                ImageIcon icon = new ImageIcon(img);
                            Image image = icon.getImage().getScaledInstance(150, 150,
        Image.SCALE_SMOOTH);
                            icon.setImage(image);
                            sLeftImg.setIcon(icon);
                          sLeftImg.setText(""); //set the label to null ready for the image to be
        displayed

                          ImageIcon icon2 = new ImageIcon(img);
```

```
MCCLX.                                                  Image image2 = icon.getImage().getScaledInstance(350, 350,
        Image.SCALE_SMOOTH);
MCCLXI.                                                 icon2.setImage(image2);
MCCLXII.                                                    sRightImg.setIcon(icon2);
MCCLXIII.                                               sRightImg.setText(""); //set the label to null ready for the
        image to be displayed
MCCLXIV.                                                sNew.setEnabled(true);
MCCLXV.                                             sExisting.setEnabled(true);
MCCLXVI.                                                detect.setEnabled(true);
MCCLXVII.                                               }else{
MCCLXVIII.                                              JOptionPane.showMessageDialog(frame,"Please select
        file of type jpg");
MCCLXIX.                                                }
MCCLXX.                                             }
MCCLXXI.                                            } catch (IOException e1) {
MCCLXXII.                                               JOptionPane.showMessageDialog(frame,"Cannot perform
        operation.");
MCCLXXIII.                                          }
MCCLXXIV.
MCCLXXV.                                    }else if(e.getSource() == sNew){
MCCLXXVI.                                           //add record to plant table and leaf table
MCCLXXVII.                                          int response = JOptionPane.showConfirmDialog(frame, "Add new plant
        species?", "Confirm",
MCCLXXVIII.                                             JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);
MCCLXXIX.                                           if(response == JOptionPane.YES_NO_OPTION){
MCCLXXX.
MCCLXXXI.                                               try {
MCCLXXXII.                                                  addToP();
MCCLXXXIII.                                                 addToL();
MCCLXXXIV.                                                  updateSummary();
MCCLXXXV.                                               } catch (ClassNotFoundException | SQLException | IOException
        e3) {
MCCLXXXVI.                                                  JOptionPane.showMessageDialog(frame,"An error
        occurred!");
MCCLXXXVII.                                                 e3.printStackTrace();
MCCLXXXVIII.                                            }
MCCLXXXIX.                                          }
MCCXC.                              }else if(e.getSource() == sExisting){
MCCXCI.                                     //add record to leaf table
MCCXCII.                                            if(!sName.getText().equals("") | !sName.getText().equals(null)){
MCCXCIII.                                               int response = JOptionPane.showConfirmDialog(frame, "Add
        existing plant species?", "Confirm",
MCCXCIV.                                                    JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);
MCCXCV.                                                 if(response == JOptionPane.YES_NO_OPTION){
MCCXCVI.                                                    try {
MCCXCVII.                                                       addToL();
MCCXCVIII.                                                      updateSummary();
MCCXCIX.                                                    } catch (ClassNotFoundException | SQLException e1) {
```

```
MCCC.                                                   JOptionPane.showMessageDialog(frame,"An error
        occurred!");
MCCCI.                                          }
MCCCII.                                 }
MCCCIII.                        }else{
MCCCIV.                                 JOptionPane.showMessageDialog(frame,"Please input species name");
MCCCV.                          }
MCCCVI.                 }else if(e.getSource() == sClear){
MCCCVII.                                sName.setText("");
MCCCVIII.                               sType.setText("");
MCCCIX.                         sArr.setText("");
MCCCX.                          sBlade.setText("");
MCCCXI.                         sMargin.setText("");
MCCCXII.                                sVena.setText("");
MCCCXIII.                       }else if(e.getSource() == train){
MCCCXIV.                                if(Integer.parseInt(imgCount.getText()) > 0){
MCCCXV.                                         stopTraining.setEnabled(true);
MCCCXVI.                                        train.setEnabled(false);
MCCCXVII.                                       try {
MCCCXVIII.                                              trainNetwork();
MCCCXIX.                                        } catch (ClassNotFoundException | SQLException e1) {
MCCCXX.                                                 JOptionPane.showMessageDialog(frame,"An error
        occurred!");
MCCCXXI.                                        } catch (IOException e1) {
MCCCXXII.                                               JOptionPane.showMessageDialog(frame,"An error
        occurred!");
MCCCXXIII.                                      }
MCCCXXIV.                               }else{
MCCCXXV.                                        JOptionPane.showMessageDialog(frame, "Cannot train network.");
MCCCXXVI.                               }
MCCCXXVII.                      }else if(e.getSource() == saveNet){
MCCCXXVIII.                             if(m!=null){
MCCCXXIX.                                       int response = JOptionPane.showConfirmDialog(frame, "Save
        network?", "Confirm",
MCCCXXX.                                            JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);
MCCCXXXI.                                       if(response == JOptionPane.YES_NO_OPTION){
MCCCXXXII.                                              m.save("file/Weights.txt");
MCCCXXXIII.                                             JOptionPane.showMessageDialog(frame,"Trained
        network saved!");
MCCCXXXIV.                                      }
MCCCXXXV.                               }else{
MCCCXXXVI.                                      JOptionPane.showMessageDialog(frame, "Network has not yet
        been trained.");
MCCCXXXVII.                             }
MCCCXXXVIII.                    }else if(e.getSource() == upNet){
MCCCXXXIX.                              int response = JOptionPane.showConfirmDialog(frame, "Upload network?",
        "Confirm",
```

```
MCCCXL.                                         JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);
MCCCXLI.                                    if(response == JOptionPane.YES_NO_OPTION){
MCCCXLII.                                       try {
MCCCXLIII.                                          upload();
MCCCXLIV.                                           JOptionPane.showMessageDialog(frame,"Trained
        network uploaded!");
MCCCXLV.                                        } catch (FileNotFoundException | ClassNotFoundException |
        SQLException e1) {
MCCCXLVI.                                           JOptionPane.showMessageDialog(frame,"Error. File
        cannot be uploaded.");
MCCCXLVII.                                          e1.printStackTrace();
MCCCXLVIII.                                     }
MCCCXLIX.                                   }
MCCCL.                          }else if(e.getSource() == nClear){
MCCCLI.                             hidden.setText("30");
MCCCLII.                                    learningRate.setText("0.5");
MCCCLIII.                                   maxError.setText("0.01");
MCCCLIV.                            }else if(e.getSource() == errClear){
MCCCLV.                             //epochs = 0;
MCCCLVI.                                    //epochCount.setText("0");
MCCCLVII.                                   createGraph(new ArrayList<Double>());
MCCCLVIII.                          }else if(e.getSource() == errGraph){
MCCCLIX.                                    createGraph(errors);
MCCCLX.                             }else if(e.getSource() == stopTraining){
MCCCLXI.                                    isStop = true;
MCCCLXII.                                   train.setEnabled(true);
MCCCLXIII.                                  stopTraining.setEnabled(false);
MCCCLXIV.                           }else if(e.getSource() == close){
MCCCLXV.                                    int response = JOptionPane.showConfirmDialog(frame, "Exit?", "Confirm",
MCCCLXVI.                                       JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);
MCCCLXVII.                                  if(response == JOptionPane.YES_NO_OPTION){
MCCCLXVIII.                                     System.exit(0);
MCCCLXIX.                                   }
MCCCLXX.                            }else if(e.getSource() == lowT){
MCCCLXXI.                                   boolean flag = false;
MCCCLXXII.                                  float l = low;
MCCCLXXIII.                                 try{
MCCCLXXIV.                                      l = Float.parseFloat(JOptionPane.showInputDialog(frame,"Enter
        low threshold: ",low));
MCCCLXXV.                                   }catch(NumberFormatException n){
MCCCLXXVI.                                      JOptionPane.showMessageDialog(frame, "Invalid input!");
MCCCLXXVII.                                     flag = true;
MCCCLXXVIII.                                }catch(NullPointerException n){
MCCCLXXIX.                                      flag = true;
MCCCLXXX.                                   }
MCCCLXXXI.                                  if(!flag) low = l;
```

```
MCCCLXXXII.                 }else if(e.getSource() == highT){
MCCCLXXXIII.                        boolean flag = false;
MCCCLXXXIV.                         float h = high;
MCCCLXXXV.                          try{
MCCCLXXXVI.                                 h = Float.parseFloat(JOptionPane.showInputDialog(frame,"Enter
        high threshold: ",high));
MCCCLXXXVII.                        }catch(NumberFormatException n){
MCCCLXXXVIII.                               JOptionPane.showMessageDialog(frame, "Invalid
        input!");
MCCCLXXXIX.                         flag = true;
MCCCXC.                             }catch(NullPointerException n){
MCCCXCI.                                    flag = true;
MCCCXCII.                           }
MCCCXCIII.                          if(!flag) high = h;
MCCCXCIV.                   }else if(e.getSource() == detect){
MCCCXCV.                            CannyEdgeDetector c = extractFeatures();
MCCCXCVI.                           c.process();
MCCCXCVII.                          BufferedImage edges = c.getEdgesImage();
MCCCXCVIII.                         Image image = edges.getScaledInstance(350, 350, Image.SCALE_SMOOTH);
MCCCXCIX.                     sRightImg.setIcon(new ImageIcon(image));
MCD.            sRightImg.setText("");
MCDI.           }else if(e.getSource() == update){
MCDII.                      try {
MCDIII.                             displayUSet();
MCDIV.                      } catch (ClassNotFoundException | SQLException | IOException e1) {
MCDV.                               JOptionPane.showMessageDialog(frame,"An error occurred.");
MCDVI.                      }
MCDVII.                 }else if(e.getSource() == download){
MCDVIII.                    if(uList.getSelectedValue() != null){
MCDIX.                              boolean err = false;
MCDX.                               int caseNo = Integer.parseInt(uList.getSelectedValue().substring(8));
MCDXI.                              SQL s = new SQL(url, user, pass);
MCDXII.                             try {
MCDXIII.                                    s.connect();
MCDXIV.                                     Blob b = s.queryImage(caseNo);
MCDXV.                                      byte[] imgData = b.getBytes(1,(int) b.length());
MCDXVI.                     FileOutputStream fout = new FileOutputStream(uList.getSelectedValue()+".jpg");
MCDXVII.                        fout.write(imgData);
MCDXVIII.                       fout.flush();
MCDXIX.                 fout.close();
MCDXX.                              } catch (ClassNotFoundException | SQLException | IOException e1) {
MCDXXI.                                     JOptionPane.showMessageDialog(frame,"An error occurred.");
MCDXXII.                                        err = true;
MCDXXIII.                                   }
MCDXXIV.                                    if(!err) JOptionPane.showMessageDialog(frame,"Download
        successful!");
MCDXXV.                             }else{
```

```
MCDXXVI.                                        JOptionPane.showMessageDialog(frame,"Please select an image.");
MCDXXVII.                               }
MCDXXVIII.
MCDXXIX.                        }else if(e.getSource() == delete){
MCDXXX.                                 if(uList.getSelectedValue() != null){
MCDXXXI.                                        int response = JOptionPane.showConfirmDialog(frame, "Do you
        want to continue?", "Confirm",
MCDXXXII.                                           JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);
MCDXXXIII.                                      if (response == JOptionPane.YES_OPTION) {
MCDXXXIV.                                               delete();
MCDXXXV.                                        }
MCDXXXVI.                               }else{
MCDXXXVII.                                      JOptionPane.showMessageDialog(frame,"Please select an image.");
MCDXXXVIII.                             }
MCDXXXIX.                       }
MCDXL.          }
MCDXLI.
MCDXLII.                //delete from list
MCDXLIII.               public void delete(){
MCDXLIV.                        int i = uList.getSelectedIndex();
MCDXLV.                 if(uListModel.size() >1){
MCDXLVI.                                if(i!=0)
MCDXLVII.                                       uList.setSelectedIndex(i);
MCDXLVIII.                              else
MCDXLIX.                                        uList.setSelectedIndex(i-1);
MCDL.                   }
MCDLI.                  SQL s = new SQL(url, user, pass);
MCDLII.                 try {
MCDLIII.                        s.connect();
MCDLIV.                         s.deleteImg(Integer.parseInt(uList.getSelectedValue().substring(8)));
MCDLV.                          s.closeConnection();
MCDLVI.                         displayUSet();
MCDLVII.                        } catch (ClassNotFoundException | SQLException | NumberFormatException |
        IOException e1) {
MCDLVIII.                               JOptionPane.showMessageDialog(frame,"An error occurred.");
MCDLIX.                 }
MCDLX.                  if(uListModel.size() >0){
MCDLXI.                         if(i!=0) uList.setSelectedIndex(i-1);
MCDLXII.                                else uList.setSelectedIndex(i);
MCDLXIII.                       }else{
MCDLXIV.                                uImg.setIcon(null);
MCDLXV.                                 uImg.setText("Image Preview");
MCDLXVI.                        }
MCDLXVII.               }
MCDLXVIII.              //add record to plant
MCDLXIX.                public void addToP() throws ClassNotFoundException, SQLException, IOException{
```

```
                                Path path = Paths.get(img);
                                byte[] b = Files.readAllBytes(path);
                                Blob blob = null;
                                blob = new SerialBlob(b);

                                Species sp = new Species();
                                sp.setName(sName.getText());
                                sp.setType(sType.getText());
                                sp.setArrangement(sArr.getText());
                                sp.setBlade(sBlade.getText());
                                sp.setMargin(sMargin.getText());
                                sp.setVenation(sVena.getText());
                                sp.setImg(blob);

                                SQL s = new SQL(url, user, pass);
                                s.connect();
                                s.addToPlant(sp);
                                s.closeConnection();
                        }

        //add record to leaf
        public void addToL() throws ClassNotFoundException, SQLException{
                                SQL s = new SQL(url, user, pass);
                                s.connect();
                                CannyEdgeDetector c = extractFeatures();
                                Leaf leaf = new Leaf();
                                leaf.setID(s.queryByName(sName.getText()));
                                leaf.setMoments(getMoments(c));
                                leaf.setRadii(getRadii(c));

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

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

                //network listener
      public void handleLearningEvent(LearningEvent event) {
                        double lr = Double.parseDouble(learningRate.getText());
                        double me = Double.parseDouble(maxError.getText());
```

```
MDXVI.
MDXVII.     BackPropagation bp = (BackPropagation)event.getSource();
MDXVIII.                    bp.setLearningRate(lr);
MDXIX.                      bp.setMaxError(me);
MDXX.
MDXXI.                      errors.add(100*bp.getTotalNetworkError());
MDXXII.                     netError.setText(String.format("%.10f", bp.getTotalNetworkError()));
MDXXIII.                    epochCount.setText(Integer.toString(bp.getCurrentIteration()));
MDXXIV.
MDXXV.                      this.epochs = bp.getCurrentIteration();
MDXXVI.                     if(isStop){ bp.stopLearning(); isStop = false; }
MDXXVII.                            if(bp.getTotalNetworkError() < me){ train.setEnabled(true);
        stopTraining.setEnabled(false); }
MDXXVIII.           }
MDXXIX.
MDXXX.          //train network using back propagation
MDXXXI.         @SuppressWarnings("rawtypes")
MDXXXII.                    public void trainNetwork() throws ClassNotFoundException, SQLException, IOException{
MDXXXIII.                           ArrayList<Leaf> leaves = new ArrayList<Leaf>();
MDXXXIV.                            SQL s = new SQL(url, user, pass);
MDXXXV.                             s.connect();
MDXXXVI.                            leaves = s.queryLeaf();
MDXXXVII.                           s.closeConnection();
MDXXXVIII.                          int i = Integer.parseInt(input.getText());
MDXXXIX.                            int h = Integer.parseInt(hidden.getText());
MDXL.                       int o = Integer.parseInt(output.getText());
MDXLI.
MDXLII.                     final DataSet trSet = new DataSet(i,o);
MDXLIII.                    errors.removeAll(errors);
MDXLIV.                     for(Leaf l : leaves){
MDXLV.                              int j = 0;
MDXLVI.                             double[] in = new double[i];
MDXLVII.
MDXLVIII.                                   for(double a : l.getMoments()){
MDXLIX.                                     in[j] = a;
MDL.                                ++j;
MDLI.                       }
MDLII.                              for(double a : l.getRadii()){
MDLIII.                                     in[j] = a;
MDLIV.                                      ++j;
MDLV.                       }
MDLVI.
MDLVII.                             clearOutput();
MDLVIII.                            out[l.getID()-1] = 1;
MDLIX.                              trSet.addRow(new DataSetRow(in, out));
MDLX.                       }
MDLXI.
```

```
MDLXII.                 final MultiLayerPerceptron mlp = new
        MultiLayerPerceptron(TransferFunctionType.SIGMOID,i,h,o);
MDLXIII.        LearningRule learningRule = mlp.getLearningRule();
MDLXIV.         learningRule.addListener(this);
MDLXV.          learningRule.setTrainingSet(trSet);
MDLXVI.
MDLXVII.                w = new SwingWorker(){
MDLXVIII.                               @Override
MDLXIX.                 protected Object doInBackground() throws Exception {
MDLXX.                     mlp.learn(trSet);
MDLXXI.                 return null;
MDLXXII.                                }
MDLXXIII.               };
MDLXXIV.                w.execute();
MDLXXV.                 m = mlp;
MDLXXVI.
MDLXXVII.       }
MDLXXVIII.
MDLXXIX.        //upload network file to server
MDLXXX.         public void upload() throws FileNotFoundException, SQLException, ClassNotFoundException{
MDLXXXI.                SQL s = new SQL(url, user, pass);
MDLXXXII.               s.connect();
MDLXXXIII.              s.addToNetwork(new File("file/Weights.txt"));
MDLXXXIV.               s.closeConnection();
MDLXXXV.        }
MDLXXXVI.
MDLXXXVII.      //generate error graph
MDLXXXVIII.     public void createGraph(ArrayList<Double> err){
MDLXXXIX.
MDXC.           Border line = new LineBorder(Color.BLACK);
MDXCI.          Border margin = new EmptyBorder(5, 15, 5, 15);
MDXCII.         Border compound = new CompoundBorder(line, margin);
MDXCIII.
MDXCIV.         if(epochs < 400){
MDXCV.                  errorPanel = new ErrorGraph(500,300,err);
MDXCVI.                 errorPanel.setPreferredSize(new Dimension(500,300));
MDXCVII.                }else{
MDXCVIII.                       errorPanel = new ErrorGraph(epochs+150,300,err);
MDXCIX.                 errorPanel.setPreferredSize(new Dimension(epochs+150,300));
MDC.    }
MDCI.   errorPanel.add(new JLabel("Error Graph"));
MDCII.          errorPanel.setBorder(compound);
MDCIII.
MDCIV.          sPane.repaint();
MDCV.           sPane.revalidate();
MDCVI.          sPane.setViewportView(errorPanel);
MDCVII.         sPane.getViewport().setScrollMode(JViewport.SIMPLE_SCROLL_MODE);
```

```
MDCVIII.                epochCount.setText(Integer.toString(epochs));
MDCIX.          }
MDCX.
MDCXI.          //get summary
MDCXII.         public void updateSummary() throws ClassNotFoundException, SQLException{
MDCXIII.                SQL s = new SQL(url, user, pass);
MDCXIV.                 s.connect();
MDCXV.                  imgCount.setText(Integer.toString(s.getCount("leaf")));
MDCXVI.                 specCount.setText(Integer.toString(s.getCount("plant")));
MDCXVII.                        output.setText(Integer.toString(s.getCount("plant")));
MDCXVIII.                       s.closeConnection();
MDCXIX.         }
MDCXX.
MDCXXI.         //extract image moments
MDCXXII.                @SuppressWarnings("static-access")
MDCXXIII.               public ArrayList<Double> getMoments(CannyEdgeDetector c){
MDCXXIV.                        c.preprocess();
MDCXXV.                         int w = c.getWidth();
MDCXXVI.                        int h = c.getHeight();
MDCXXVII.                       int[] data = c.getData();
MDCXXVIII.                      double[][] matrix = new double[h][w];
MDCXXIX.                        for(int i = 0; i < h; i++){
MDCXXX.                                 for(int j = 0; j < w; j++){
MDCXXXI.                                        matrix[i][j] = data[(i*w)+j];
MDCXXXII.                               }
MDCXXXIII.                      }
MDCXXXIV.
MDCXXXV.                        ArrayList<Double> moments = new ArrayList<Double>();
MDCXXXVI.                       Moments m = new Moments();
MDCXXXVII.                      moments.add(m.getNormalizedCentralMoment(0, 0, matrix));
MDCXXXVIII.                     moments.add(m.getNormalizedCentralMoment(1, 0, matrix));
MDCXXXIX.                       moments.add(m.getNormalizedCentralMoment(0, 1, matrix));
MDCXL.                  moments.add(m.getNormalizedCentralMoment(1, 1, matrix));
MDCXLI.
MDCXLII.                        return moments;
MDCXLIII.               }
MDCXLIV.
MDCXLV.         //extract image radii
MDCXLVI.                public ArrayList<Double> getRadii(CannyEdgeDetector c){
MDCXLVII.                       ArrayList<Double> radii = new ArrayList<Double>();
MDCXLVIII.                      c.process();
MDCXLIX.                        CentroidRadii cr = new CentroidRadii(c.getCentroidX(),c.getCentroidY(),c.getWidth());
MDCL.                   cr.setOutline(c.getOutline());
MDCLI.                  cr.determineRadii();
MDCLII.                 radii = cr.getRadiiSet();
MDCLIII.                BufferedImage edges = c.getEdgesImage();
```

```
MDCLIV.                 Image image = edges.getScaledInstance(350, 350, Image.SCALE_SMOOTH);
MDCLV.              sRightImg.setIcon(new ImageIcon(image));
MDCLVI.             sRightImg.setText("");
MDCLVII.                        return radii;
MDCLVIII.               }
MDCLIX.
MDCLX.          //extract features(moments,radii)
MDCLXI.         public CannyEdgeDetector extractFeatures(){
MDCLXII.                        CannyEdgeDetector c = new CannyEdgeDetector();
MDCLXIII.                       c.setLowThreshold(low);
MDCLXIV.                        c.setHighThreshold(high);
MDCLXV.                         try {
MDCLXVI.                                c.setSourceImage(ImageIO.read(new File(img)));
MDCLXVII.                       } catch (IOException e) {
MDCLXVIII.                              JOptionPane.showMessageDialog(frame,"An error occurred.");
MDCLXIX.                        }
MDCLXX.                         return c;
MDCLXXI.                }
MDCLXXII.
MDCLXXIII.              //main
MDCLXXIV.               public static void main(String[] args) throws ClassNotFoundException, SQLException,
            IOException{
MDCLXXV.                        try{
MDCLXXVI.                               @SuppressWarnings("resource")
MDCLXXVII.                              BufferedReader br = new BufferedReader(new FileReader("file/SQL.txt"));
MDCLXXVIII.                 String line = br.readLine();
MDCLXXIX.                   StringTokenizer tokens = new StringTokenizer(line,";");
MDCLXXX.                    url = tokens.nextToken();
MDCLXXXI.                   user = tokens.nextToken();
MDCLXXXII.                  pass = tokens.nextToken();
MDCLXXXIII.               } catch(IOException io){
MDCLXXXIV.                      JOptionPane.showMessageDialog(null,"File containing connection details does not
            exist.");
MDCLXXXV.                 }
MDCLXXXVI.
MDCLXXXVII.                     if(!url.equals("") || !user.equals("") || !pass.equals("")){
MDCLXXXVIII.                            ExpertGUI e = new ExpertGUI("jdbc:mysql://"+url,user,pass);
MDCLXXXIX.                              e.createFrame();
MDCXC.                          System.out.println("Operation completed.");
MDCXCI.                 }else{
MDCXCII.                                System.out.println("Cannot perform operation");
MDCXCIII.                       }
MDCXCIV.                }
MDCXCV.         }
MDCXCVI.
MDCXCVII.       ImageFade.java
MDCXCVIII.      package leafRec;
MDCXCIX.        import java.awt.*;
```

```
MDCC.
MDCCI.   import javax.swing.*;
MDCCII.  @SuppressWarnings("serial")
MDCCIII. class ImageFade extends JPanel {
MDCCIV.          private Image img;
MDCCV.           private float alpha = 0.0f;
MDCCVI.          private int trans = 0;
MDCCVII.                 private int y = 0;
MDCCVIII.
MDCCIX.          public ImageFade(String img,int trans) {
MDCCX.                   this(new ImageIcon(img).getImage(), trans);
MDCCXI.            this.trans = trans;
MDCCXII.                 }
MDCCXIII.
MDCCXIV.                 public ImageFade(Image img, int trans) {
MDCCXV.                    this.img = img;
MDCCXVI.                   this.trans = trans;
MDCCXVII.                  Dimension size = new Dimension(img.getWidth(null)+50, img.getHeight(null)+300);
MDCCXVIII.                 setPreferredSize(size);
MDCCXIX.                   setMinimumSize(size);
MDCCXX.                    setMaximumSize(size);
MDCCXXI.                   setSize(size);
MDCCXXII.                  setLayout(null);
MDCCXXIII.                 }
MDCCXXIV.
MDCCXXV.                 public ImageFade(){
MDCCXXVI.                        super();
MDCCXXVII.               }
MDCCXXVIII.
MDCCXXIX.                public void paintComponent(Graphics g) {
MDCCXXX.                   Graphics2D g2d = (Graphics2D) g;
MDCCXXXI.
MDCCXXXII.                 //set the opacity
MDCCXXXIII.                g2d.setComposite(AlphaComposite.getInstance(
MDCCXXXIV.                               AlphaComposite.SRC_OVER, alpha));
MDCCXXXV.
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS_ON);
MDCCXXXVI.
MDCCXXXVII.                //do the drawing here
MDCCXXXVIII.               g.drawImage(img, 0, y, null);
MDCCXXXIX.                 y=y-trans;
MDCCXL.
MDCCXLI.                   //increase the opacity and repaint
MDCCXLII.                  alpha += 0.01f;
MDCCXLIII.                 if (alpha >= 1.0f) alpha = 1.0f;
MDCCXLIV.                  else repaint();
MDCCXLV.
MDCCXLVI.                        //sleep for a bit
MDCCXLVII.                       try {
```

```
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                System.out.println("Error");
            }
            }
        }

```

ImagePanel.java

```
package leafRec;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;

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

@SuppressWarnings("serial")
class ImagePanel extends JPanel {

  private Image img;

  public ImagePanel(String img) {
    this(new ImageIcon(img).getImage());
  }

  public ImagePanel(Image img) {
    this.img = img;
    Dimension size = new Dimension(img.getWidth(null), img.getHeight(null));
    setPreferredSize(size);
    setMinimumSize(size);
    setMaximumSize(size);
    setSize(size);
    setLayout(null);
  }

  public void paintComponent(Graphics g) {
    g.drawImage(img, 0, 0, null);
  }

            }

```

Leaf.java

```
package leafRec;
import java.util.ArrayList;

public class Leaf {
        //parameters
        private int plantID;
        private ArrayList<Double> moments;
        private ArrayList<Double> radii;
```

```
MDCCXCVIII.
MDCCXCIX.               //getters
MDCCC.          public int getID(){
MDCCCI.                 return plantID;
MDCCCII.                }
MDCCCIII.
MDCCCIV.                public ArrayList<Double> getMoments(){
MDCCCV.                         return moments;
MDCCCVI.                }
MDCCCVII.
MDCCCVIII.              public ArrayList<Double> getRadii(){
MDCCCIX.                        return radii;
MDCCCX.                 }
MDCCCXI.
MDCCCXII.               //setters
MDCCCXIII.              public void setID(int plantID){
MDCCCXIV.                       this.plantID = plantID;
MDCCCXV.                }
MDCCCXVI.
MDCCCXVII.              public void setMoments(ArrayList<Double> moments){
MDCCCXVIII.                     this.moments = moments;
MDCCCXIX.               }
MDCCCXX.
MDCCCXXI.               public void setRadii(ArrayList<Double> radii){
MDCCCXXII.                      this.radii = radii;
MDCCCXXIII.             }
MDCCCXXIV.      }
MDCCCXXV.
MDCCCXXVI.      LeafImage.java
MDCCCXXVII.
MDCCCXXVIII.    package leafRec;
MDCCCXXIX.
MDCCCXXX.       import java.sql.Blob;
MDCCCXXXI.
MDCCCXXXII.     public class LeafImage {
MDCCCXXXIII.            private Blob b;
MDCCCXXXIV.             private int caseNo;
MDCCCXXXV.
MDCCCXXXVI.             public Blob getImg(){
MDCCCXXXVII.                            return b;
MDCCCXXXVIII.                   }
MDCCCXXXIX.
MDCCCXL.                public int getCaseNo(){
MDCCCXLI.                       return caseNo;
MDCCCXLII.              }
MDCCCXLIII.
MDCCCXLIV.              public void setImg(Blob b){
MDCCCXLV.                       this.b = b;
```

```
MDCCCXLVI.          }
MDCCCXLVII.
MDCCCXLVIII.        public void setCaseNo(int caseNo){
MDCCCXLIX.                  this.caseNo = caseNo;
MDCCCL.             }
MDCCCLI.    }
MDCCCLII.
MDCCCLIII.  Moments.java
MDCCCLIV.   package leafRec;
MDCCCLV.    import java.util.logging.Logger;
MDCCCLVI.
MDCCCLVII.  /**
MDCCCLVIII.  * Image moments
MDCCCLIX.    * @author Arlington
MDCCCLX.     * @author Saulo (scsm@ecomp.poli.br)<p>
MDCCCLXI.    * {@link http://en.wikipedia.org/wiki/Image_moment}
MDCCCLXII.   */
MDCCCLXIII.
MDCCCLXIV.  public class Moments {
MDCCCLXV.
MDCCCLXVI.          private static Logger logger = Logger.getLogger(Moments.class.getName());
MDCCCLXVII.
MDCCCLXVIII.        public static double getRawMoment(int p, int q, double[][] matrix) {
MDCCCLXIX.                  double m = 0;
MDCCCLXX.                   logger.finest("Calculating raw moment for p="+p+" and q="+q);
MDCCCLXXI.                  for (int i = 0, k = matrix.length; i < k; i++) {
MDCCCLXXII.                         for (int j = 0, l = matrix[i].length; j < l; j++) {
MDCCCLXXIII.                                m += Math.pow(i, p) * Math.pow(j, q) * matrix[i][j];
MDCCCLXXIV.                         }
MDCCCLXXV.                  }
MDCCCLXXVI.                 return m;
MDCCCLXXVII.        }
MDCCCLXXVIII.
MDCCCLXXIX.         public static double getCentralMoment(int p, int q, double[][] img) {
MDCCCLXXX.                  double mc = 0;
MDCCCLXXXI.                 double m00 = Moments.getRawMoment(0, 0, img);
MDCCCLXXXII.                double m10 = Moments.getRawMoment(1, 0, img);
MDCCCLXXXIII.                       double m01 = Moments.getRawMoment(0, 1, img);
MDCCCLXXXIV.                        double x0 = m10 / m00;
MDCCCLXXXV.                 double y0 = m01 / m00;
MDCCCLXXXVI.                        for (int i = 0, k = img.length; i < k; i++) {
MDCCCLXXXVII.                               for (int j = 0, l = img[i].length; j < l; j++) {
MDCCCLXXXVIII.                                     mc += Math.pow((i - x0), p) * Math.pow((j - y0), q) *
        img[i][j];
MDCCCLXXXIX.                                }
MDCCCXC.                    }
MDCCCXCI.                   return mc;
MDCCCXCII.          }
MDCCCXCIII.
```

```
MDCCCXCIV.              public static double getCovarianceXY(int p, int q, double[][] matrix) {
MDCCCXCV.                       double mc00 = Moments.getCentralMoment(0, 0, matrix);
MDCCCXCVI.                      double mc11 = Moments.getCentralMoment(1, 1, matrix);
MDCCCXCVII.                     return mc11 / mc00;
MDCCCXCVIII.            }
MDCCCXCIX.
MCM.        /**
MCMI.               * Returns the variance in x-direction
MCMII.              * @param p
MCMIII.             * @param q
MCMIV.              * @param matrix containing pixel map for one layer
MCMV.               * @return
MCMVI.              */
MCMVII.             public static double getVarianceX(int p, int q, double[][] matrix) {
MCMVIII.                        double mc00 = Moments.getCentralMoment(0, 0, matrix);
MCMIX.                  double mc20 = Moments.getCentralMoment(2, 0, matrix);
MCMX.                   return mc20 / mc00;
MCMXI.              }
MCMXII.
MCMXIII.                /**
MCMXIV.             * Returns the variance in y-direction
MCMXV.              * @param p
MCMXVI.                  * @param q
MCMXVII.                 * @param matrix containing pixel map for one layer
MCMXVIII.                * @return
MCMXIX.                  */
MCMXX.              public static double getVarianceY(int p, int q, double[][] matrix) {
MCMXXI.                         double mc00 = Moments.getCentralMoment(0, 0, matrix);
MCMXXII.                        double mc02 = Moments.getCentralMoment(0, 2, matrix);
MCMXXIII.                       return mc02 / mc00;
MCMXXIV.                }
MCMXXV.
MCMXXVI.                /**
MCMXXVII.                * Normalized Central Moment
MCMXXVIII.               * @param p
MCMXXIX.                 * @param q
MCMXXX.                  * @param matrix the pixel map
MCMXXXI.                 * @return Normalized Central Moment n_pq
MCMXXXII.                */
MCMXXXIII.              public static double getNormalizedCentralMoment(int p, int q, double[][] matrix) {
MCMXXXIV.                       double gama = ((p + q) / 2) + 1;
MCMXXXV.                        double mpq = Moments.getCentralMoment(p, q, matrix);
MCMXXXVI.                       double m00gama = Math.pow(Moments.getCentralMoment(0, 0, matrix), gama);
MCMXXXVII.                      return mpq / m00gama;
MCMXXXVIII.             }
MCMXXXIX.
MCMXL.              /**
```

```
MCMXLI.     * Hu invariant moments
MCMXLII.             * @param matrix the pixel map
MCMXLIII.            * @param n the Hu moment horder
MCMXLIV.             * @return n-order Hu moment
MCMXLV.              */
MCMXLVI.             public static double getHuMoment (double[][] matrix,int n) {
MCMXLVII.                    double result = 0.0;
MCMXLVIII.
MCMXLIX.                     double
MCML.                n20 = Moments.getNormalizedCentralMoment(2, 0, matrix),
MCMLI.               n02 = Moments.getNormalizedCentralMoment(0, 2, matrix),
MCMLII.              n30 = Moments.getNormalizedCentralMoment(3, 0, matrix),
MCMLIII.             n12 = Moments.getNormalizedCentralMoment(1, 2, matrix),
MCMLIV.              n21 = Moments.getNormalizedCentralMoment(2, 1, matrix),
MCMLV.               n03 = Moments.getNormalizedCentralMoment(0, 3, matrix),
MCMLVI.              n11 = Moments.getNormalizedCentralMoment(1, 1, matrix);
MCMLVII.
MCMLVIII.                    switch (n) {
MCMLIX.              case 1:
MCMLX.                       result = n20 + n02;
MCMLXI.                      break;
MCMLXII.                     case 2:
MCMLXIII.                            result = Math.pow((n20 - 02), 2) + Math.pow(2 * n11, 2);
MCMLXIV.                             break;
MCMLXV.                      case 3:
MCMLXVI.                             result = Math.pow(n30 - (3 * (n12)), 2)
MCMLXVII.                                           + Math.pow((3 * n21 - n03), 2);
MCMLXVIII.                           break;
MCMLXIX.                     case 4:
MCMLXX.                              result = Math.pow((n30 + n12), 2) + Math.pow((n12 + n03), 2);
MCMLXXI.                             break;
MCMLXXII.                    case 5:
MCMLXXIII.                           result = (n30 - 3 * n12) * (n30 + n12)
MCMLXXIV.                                           * (Math.pow((n30 + n12), 2) - 3 * Math.pow((n21 +
        n03), 2))
MCMLXXV.                                            + (3 * n21 - n03) * (n21 + n03)
MCMLXXVI.                                           * (3 * Math.pow((n30 + n12), 2) - Math.pow((n21 +
        n03), 2));
MCMLXXVII.                           break;
MCMLXXVIII.                  case 6:
MCMLXXIX.                            result = (n20 - n02)
MCMLXXX.                                            * (Math.pow((n30 + n12), 2) - Math.pow((n21 + n03),
        2))
MCMLXXXI.                                           + 4 * n11 * (n30 + n12) * (n21 + n03);
MCMLXXXII.                           break;
MCMLXXXIII.                  case 7:
MCMLXXXIV.                           result = (3 * n21 - n03) * (n30 + n12)
```

```
MCMLXXXV.                                                   * (Math.pow((n30 + n12), 2) - 3 * Math.pow((n21 +
        n03), 2))
MCMLXXXVI.                                                  + (n30 - 3 * n12) * (n21 + n03)
MCMLXXXVII.                                                 * (3 * Math.pow((n30 + n12), 2) - Math.pow((n21 +
        n03), 2));
MCMLXXXVIII.                            break;
MCMLXXXIX.
MCMXC.                  default:
MCMXCI.                                 throw new IllegalArgumentException("Invalid number for Hu moment.");
MCMXCII.                        }
MCMXCIII.                       return result;
MCMXCIV.                }
MCMXCV.         }
MCMXCVI.
MCMXCVII.
MCMXCVIII.      OutputSpecies.java
MCMXCIX.        package leafRec;
MM.     public class OutputSpecies {
MMI.            private int ID;
MMII.           private double outputValue;
MMIII.
MMIV.           public void setID(int ID){
MMV.                    this.ID = ID;
MMVI.                   }
MMVII.
MMVIII.                 public void setOutput(double outputValue){
MMIX.                           this.outputValue = outputValue;
MMX.    }
MMXI.
MMXII.                  public int getID(){
MMXIII.                         return ID;
MMXIV.                  }
MMXV.
MMXVI.                  public double getOutput(){
MMXVII.                         return outputValue;
MMXVIII.                        }
MMXIX.
MMXX.   }
MMXXI.
MMXXII. Species.java
MMXXIII.                package leafRec;
MMXXIV.import java.sql.Blob;
MMXXV.
MMXXVI.                 public class Species {
MMXXVII.                        //plant properties
MMXXVIII.                       private int plantID;
MMXXIX.                         private Blob img;
MMXXX.                  private String name;
MMXXXI.
```

```
MMXXXII.                    //leaf properties
MMXXXIII.                   private String leafType;
MMXXXIV.                    private String leafArrangement;
MMXXXV.                     private String leafBlade;
MMXXXVI.                    private String leafMargin;
MMXXXVII.                   private String leafVenation;
MMXXXVIII.
MMXXXIX.                    //leaf record
MMXL.               private Leaf leaf;
MMXLI.
MMXLII.             //getters
MMXLIII.            public int getID(){
MMXLIV.                     return plantID;
MMXLV.              }
MMXLVI.
MMXLVII.                    public Blob getBlob(){
MMXLVIII.                           return img;
MMXLIX.                     }
MML.
MMLI.       public String getName(){
MMLII.                      return name;
MMLIII.             }
MMLIV.
MMLV.               public String getType(){
MMLVI.                      return leafType;
MMLVII.             }
MMLVIII.
MMLIX.              public String getArrangement(){
MMLX.                       return leafArrangement;
MMLXI.              }
MMLXII.
MMLXIII.            public String getBlade(){
MMLXIV.                     return leafBlade;
MMLXV.              }
MMLXVI.
MMLXVII.                    public String getMargin(){
MMLXVIII.                           return leafMargin;
MMLXIX.                     }
MMLXX.
MMLXXI.                     public String getVenation(){
MMLXXII.                            return leafVenation;
MMLXXIII.                   }
MMLXXIV.
MMLXXV.                     public Leaf getLeaf(){
MMLXXVI.                            return leaf;
MMLXXVII.                   }
```

```
MMLXXVIII.
MMLXXIX.                    //setters
MMLXXX.                     public void setID(int plantID){
MMLXXXI.                            this.plantID = plantID;
MMLXXXII.                   }
MMLXXXIII.
MMLXXXIV.                   public void setImg(Blob img){
MMLXXXV.                            this.img = img;
MMLXXXVI.                   }
MMLXXXVII.
MMLXXXVIII.                 public void setName(String name){
MMLXXXIX.                           this.name = name;
MMXC.               }
MMXCI.
MMXCII.             public void setType(String leafType){
MMXCIII.                            this.leafType = leafType;
MMXCIV.             }
MMXCV.
MMXCVI.                     public void setArrangement(String leafArrangement){
MMXCVII.                            this.leafArrangement = leafArrangement;
MMXCVIII.                   }
MMXCIX.
MMC.        public void setBlade(String leafBlade){
MMCI.                               this.leafBlade = leafBlade;
MMCII.              }
MMCIII.
MMCIV.              public void setMargin(String leafMargin){
MMCV.                               this.leafMargin = leafMargin;
MMCVI.              }
MMCVII.
MMCVIII.                    public void setVenation(String leafVenation){
MMCIX.                              this.leafVenation = leafVenation;
MMCX.               }
MMCXI.
MMCXII.             public void setLeaf(Leaf leaf){
MMCXIII.                            this.leaf = leaf;
MMCXIV.             }
MMCXV.  }
MMCXVI.
MMCXVII.            SQL.java
MMCXVIII.           package leafRec;
MMCXIX.
MMCXX.  import java.io.BufferedReader;
MMCXXI.             import java.io.File;
MMCXXII.            import java.io.FileInputStream;
MMCXXIII.           import java.io.FileNotFoundException;
MMCXXIV.            import java.io.FileOutputStream;
```

```
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.sql.*;
import java.util.ArrayList;
import java.util.StringTokenizer;

import javax.swing.JOptionPane;

public class SQL {
	//moments = 4, radii = 36
	private final int mCount = 4;
	private final int rCount = 36;

	//connection parameters
private String url;
private String user;
	private String pass;
	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 blob NOT NULL,"+
"leafName varchar(40) NOT NULL,"+
"leafType varchar(30) NOT NULL,"+
"leafArrangement varchar(30) NOT NULL,"+
"leafBlade varchar(30) NOT NULL,"+
"leafMargin varchar(30) NOT NULL,"+
"leafVenation varchar(30) NOT NULL,"+
```

```
MMCLXXI.                                                    "PRIMARY KEY (plantID))";
MMCLXXII.                   sment.executeUpdate(pTable);
MMCLXXIII.                  sment.close();
MMCLXXIV.           }
MMCLXXV.
MMCLXXVI.           //create leaf table
MMCLXXVII.          public void createLeafTable() throws SQLException{
MMCLXXVIII.                 Statement sment = conn.createStatement();
MMCLXXIX.
MMCLXXX.                    String lTable = "CREATE TABLE IF NOT EXISTS leaf("+
MMCLXXXI.                                                   "leafID int NOT NULL
    AUTO_INCREMENT, plantID int NOT NULL,"+
MMCLXXXII.                                                  "moment1 double NOT NULL, moment2
    double NOT NULL,"+
MMCLXXXIII.                                                 "moment3 double NOT NULL, moment4
    double NOT NULL,"+
MMCLXXXIV.                                                  "radius1 double NOT NULL, radius2 double
    NOT NULL,"+
MMCLXXXV.                                                   "radius3 double NOT NULL, radius4 double
    NOT NULL,"+
MMCLXXXVI.                                                  "radius5 double NOT NULL, radius6 double
    NOT NULL,"+
MMCLXXXVII.                                                 "radius7 double NOT NULL, radius8 double
    NOT NULL,"+
MMCLXXXVIII.                                                "radius9 double NOT NULL, radius10 double
    NOT NULL,"+
MMCLXXXIX.                                                  "radius11 double NOT NULL, radius12
    double NOT NULL,"+
MMCXC.                                                  "radius13 double NOT NULL, radius14 double NOT
    NULL,"+
MMCXCI.                                                     "radius15 double NOT NULL, radius16
    double NOT NULL,"+
MMCXCII.                                                    "radius17 double NOT NULL, radius18
    double NOT NULL,"+
MMCXCIII.                                                   "radius19 double NOT NULL, radius20
    double NOT NULL,"+
MMCXCIV.                                                    "radius21 double NOT NULL, radius22
    double NOT NULL,"+
MMCXCV.                                                     "radius23 double NOT NULL, radius24
    double NOT NULL,"+
MMCXCVI.                                                    "radius25 double NOT NULL, radius26
    double NOT NULL,"+
MMCXCVII.                                                   "radius27 double NOT NULL, radius28
    double NOT NULL,"+
MMCXCVIII.                                                  "radius29 double NOT NULL, radius30
    double NOT NULL,"+
MMCXCIX.                                                    "radius31 double NOT NULL, radius32
    double NOT NULL,"+
MMCC.                                                   "radius33 double NOT NULL, radius34 double NOT
    NULL,"+
MMCCI.                                                  "radius35 double NOT NULL, radius36 double NOT
    NULL,"+
MMCCII.                                                 "PRIMARY KEY(leafID),"+
```

```
MMCCIII.                                                    "FOREIGN KEY(plantID) REFERENCES
        plant(plantID))";
MMCCIV.
MMCCV.                  sment.executeUpdate(lTable);
MMCCVI.                         sment.close();
MMCCVII.
MMCCVIII.               }
MMCCIX.
MMCCX.          //create table network
MMCCXI.                 public void createNetworkTable() throws SQLException{
MMCCXII.                        Statement sment = conn.createStatement();;
MMCCXIII.
MMCCXIV.                        String nTable = "CREATE TABLE IF NOT EXISTS network ("+
MMCCXV.                                                     "networkID int NOT NULL
        AUTO_INCREMENT,"+
MMCCXVI.                                                    "weights longblob NOT NULL,"+
MMCCXVII.                                                   "PRIMARY KEY (networkID))";
MMCCXVIII.
MMCCXIX.                        sment.executeUpdate(nTable);
MMCCXX.                         sment.close();
MMCCXXI.                }
MMCCXXII.
MMCCXXIII.              //create table network
MMCCXXIV.               public void createImageTable() throws SQLException{
MMCCXXV.                        Statement sment = conn.createStatement();
MMCCXXVI.                       String nTable = "CREATE TABLE IF NOT EXISTS leafImg (caseNo int NOT NULL
        AUTO_INCREMENT,img blob NOT NULL, PRIMARY KEY(caseNo))";
MMCCXXVII.                      sment.executeUpdate(nTable);
MMCCXXVIII.                     sment.close();
MMCCXXIX.               }
MMCCXXX.
MMCCXXXI.               //add a record to database (plant table)
MMCCXXXII.              public void addToPlant(Species s) throws SQLException{
MMCCXXXIII.                     PreparedStatement ps = conn.prepareStatement("INSERT INTO plant
        VALUES(null,?,?,?,?,?,?,?)");
MMCCXXXIV.                      ps.setBlob(1, s.getBlob());
MMCCXXXV.                       ps.setString(2, s.getName());
MMCCXXXVI.                      ps.setString(3, s.getType());
MMCCXXXVII.                     ps.setString(4, s.getArrangement());
MMCCXXXVIII.                    ps.setString(5, s.getBlade());
MMCCXXXIX.                      ps.setString(6, s.getMargin());
MMCCXL.                         ps.setString(7, s.getVenation());
MMCCXLI.                        ps.executeUpdate();
MMCCXLII.                       ps.close();
MMCCXLIII.              }
MMCCXLIV.
MMCCXLV.                //add a record to database (leaf table)
MMCCXLVI.               public boolean addToLeaf(Leaf s) throws SQLException{
```

```
MMCCXLVII.                  PreparedStatement ps = conn.prepareStatement("INSERT INTO leaf VALUES" +
MMCCXLVIII.                 "(null,?,?,?,?,?,?,?,?,?,?,?," +
MMCCXLIX.                   "?,?,?,?,?,?,?,?,?,?,?," +
MMCCL.              "?,?,?,?,?,?,?,?,?,?,?," +
MMCCLI.             "?,?,?,?,?,?,?,?,?,?)");
MMCCLII.                    if(s.getID() <= 0) return false;
MMCCLIII.                   ps.setInt(1,s.getID());
MMCCLIV.                    for(int i = 0; i < mCount; i++){
MMCCLV.                             ps.setDouble(i+2, s.getMoments().get(i));
MMCCLVI.                    }
MMCCLVII.
MMCCLVIII.                  for(int i = 0; i < rCount; i++){
MMCCLIX.                            ps.setDouble(i+6, s.getRadii().get(i));
MMCCLX.                     }
MMCCLXI.
MMCCLXII.                   ps.executeUpdate();
MMCCLXIII.                  ps.close();
MMCCLXIV.                   return true;
MMCCLXV.            }
MMCCLXVI.
MMCCLXVII.          //add a record to database (network table)
MMCCLXVIII.         public void addToNetwork(File file) throws FileNotFoundException, SQLException{
MMCCLXIX.                   PreparedStatement ps = conn.prepareStatement("INSERT INTO network
        VALUES(null,?)");
MMCCLXX.                    //ps.setDate();
MMCCLXXI.                   FileInputStream is = new FileInputStream(file);
MMCCLXXII.                  ps.setBinaryStream(1, is, (int) file.length());
MMCCLXXIII.                 ps.executeUpdate();
MMCCLXXIV.                  ps.close();
MMCCLXXV.           }
MMCCLXXVI.
MMCCLXXVII.         //add a record to database (plant table)
MMCCLXXVIII.        public void addToLeafImg(Blob b) throws SQLException{
MMCCLXXIX.                  PreparedStatement ps = conn.prepareStatement("INSERT INTO leafImg
        VALUES(null,?)");
MMCCLXXX.                   ps.setBlob(1, b);
MMCCLXXXI.                  ps.executeUpdate();
MMCCLXXXII.                 ps.close();
MMCCLXXXIII.        }
MMCCLXXXIV.
MMCCLXXXV.          //query all from database (leaf table)
MMCCLXXXVI.         public ArrayList<Leaf> queryLeaf() throws SQLException{
MMCCLXXXVII.                        String sql = "SELECT * from leaf";
MMCCLXXXVIII.                       Statement sment = conn.createStatement();
MMCCLXXXIX.                 ResultSet rs = sment.executeQuery(sql);
MMCCXC.                     ArrayList<Leaf> leaves = new ArrayList<Leaf>();
MMCCXCI.
```

```
MMCCXCII.               while(rs.next()){
MMCCXCIII.                      Leaf l = new Leaf();
MMCCXCIV.                       ArrayList<Double> m = new ArrayList<Double>();
MMCCXCV.                        ArrayList<Double> r = new ArrayList<Double>();
MMCCXCVI.
MMCCXCVII.                      for(int i = 0; i < mCount; i++){
MMCCXCVIII.                             Double mTemp = rs.getDouble("moment"+(i+1));
MMCCXCIX.                               m.add(mTemp);
MMCCC.                  }
MMCCCI.
MMCCCII.                        for(int i = 0; i < rCount; i++){
MMCCCIII.                               Double rTemp = rs.getDouble("radius"+(i+1));
MMCCCIV.                                r.add(rTemp);
MMCCCV.                         }
MMCCCVI.                        l.setID(rs.getInt(2));
MMCCCVII.                       l.setMoments(m);
MMCCCVIII.                      l.setRadii(r);
MMCCCIX.                        leaves.add(l);
MMCCCX.                 }
MMCCCXI.
MMCCCXII.               rs.close();
MMCCCXIII.              sment.close();
MMCCCXIV.
MMCCCXV.                return leaves;
MMCCCXVI.       }
MMCCCXVII.
MMCCCXVIII.     //query all from database (plant table)
MMCCCXIX.       public ArrayList<Species> queryPlant() throws SQLException{
MMCCCXX.                String sql = "SELECT * FROM plant";
MMCCCXXI.               Statement sment = conn.createStatement();
MMCCCXXII.              ResultSet rs = sment.executeQuery(sql);
MMCCCXXIII.             ArrayList<Species> species = new ArrayList<Species>();
MMCCCXXIV.
MMCCCXXV.               while(rs.next()){
MMCCCXXVI.                      Species s = new Species();
MMCCCXXVII.                     s.setID(rs.getInt(1));
MMCCCXXVIII.                    s.setImg(rs.getBlob(2));
MMCCCXXIX.                      s.setName(rs.getString(3));
MMCCCXXX.                       s.setType(rs.getString(4));
MMCCCXXXI.                      s.setArrangement(rs.getString(5));
MMCCCXXXII.                     s.setBlade(rs.getString(6));
MMCCCXXXIII.                    s.setMargin(rs.getString(7));
MMCCCXXXIV.                     s.setVenation(rs.getString(8));
MMCCCXXXV.                      species.add(s);
MMCCCXXXVI.             }
MMCCCXXXVII.                    return species;
```

```
MMCCCXXXVIII.                   }
MMCCCXXXIX.
MMCCCXL.            //query a record from database by ID (plant table)
MMCCCXLI.           public Species queryByID(int plID) throws SQLException{
MMCCCXLII.                      String sql = "SELECT * FROM plant WHERE plantID = " + plID;
MMCCCXLIII.                     Statement sment = conn.createStatement();
MMCCCXLIV.                      ResultSet rs = sment.executeQuery(sql);
MMCCCXLV.                       Species s = new Species();
MMCCCXLVI.                      while(rs.next()){
MMCCCXLVII.                             s.setID(rs.getInt(1));
MMCCCXLVIII.                            s.setImg(rs.getBlob(2));
MMCCCXLIX.                              s.setName(rs.getString(3));
MMCCCL.                                 s.setType(rs.getString(4));
MMCCCLI.                                s.setArrangement(rs.getString(5));
MMCCCLII.                               s.setBlade(rs.getString(6));
MMCCCLIII.                              s.setMargin(rs.getString(7));
MMCCCLIV.                               s.setVenation(rs.getString(8));
MMCCCLV.                        }
MMCCCLVI.
MMCCCLVII.                      return s;
MMCCCLVIII.         }
MMCCCLIX.
MMCCCLX.            //query by name and return species
MMCCCLXI.           public Species querySpec(String name) throws SQLException{
MMCCCLXII.                      String sql = "SELECT * FROM plant WHERE leafName = '" + name + "'";
MMCCCLXIII.                     Statement sment = conn.createStatement();
MMCCCLXIV.                      ResultSet rs = sment.executeQuery(sql);
MMCCCLXV.                       Species s = new Species();
MMCCCLXVI.                      while(rs.next()){
MMCCCLXVII.                             s.setID(rs.getInt(1));
MMCCCLXVIII.                            s.setImg(rs.getBlob(2));
MMCCCLXIX.                              s.setName(rs.getString(3));
MMCCCLXX.                               s.setType(rs.getString(4));
MMCCCLXXI.                              s.setArrangement(rs.getString(5));
MMCCCLXXII.                             s.setBlade(rs.getString(6));
MMCCCLXXIII.                            s.setMargin(rs.getString(7));
MMCCCLXXIV.                             s.setVenation(rs.getString(8));
MMCCCLXXV.                      }
MMCCCLXXVI.
MMCCCLXXVII.                            return s;
MMCCCLXXVIII.                   }
MMCCCLXXIX.
MMCCCLXXX.          //query a record from database by name (plant table)
MMCCCLXXXI.         public int queryByName(String nme) throws SQLException{
MMCCCLXXXII.                            String sql = "SELECT * FROM plant WHERE leafName = '" + nme + "'";
MMCCCLXXXIII.                           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;
}

//query all leaves for verification
public ArrayList<LeafImage> queryImages() throws SQLException, IOException{
        String sql = "SELECT * FROM leafImg";
        Statement sment = conn.createStatement();
        ResultSet rs = sment.executeQuery(sql);
        ArrayList<LeafImage> l = new ArrayList<LeafImage>();

        while(rs.next()){
```

```
MMCDXXX.                                LeafImage li = new LeafImage();
MMCDXXXI.                               li.setCaseNo(rs.getInt(1));
MMCDXXXII.                          li.setImg(rs.getBlob(2));
MMCDXXXIII.                             l.add(li);
MMCDXXXIV.                      }
MMCDXXXV.
MMCDXXXVI.                      return l;
MMCDXXXVII.             }
MMCDXXXVIII.
MMCDXXXIX.              //query an image
MMCDXL.                 public Blob queryImage(int caseNum) throws SQLException, IOException{
MMCDXLI.                        String sql = "SELECT * FROM leafImg WHERE caseNo = " + caseNum;
MMCDXLII.                       Statement sment = conn.createStatement();
MMCDXLIII.                      ResultSet rs = sment.executeQuery(sql);
MMCDXLIV.                       Blob b = null;
MMCDXLV.                        while(rs.next()){
MMCDXLVI.                               b = rs.getBlob(2);
MMCDXLVII.                      }
MMCDXLVIII.
MMCDXLIX.                       return b;
MMCDL.          }
MMCDLI.
MMCDLII.                //delete an image
MMCDLIII.               public void deleteImg(int caseNum) throws SQLException, IOException{
MMCDLIV.                        String sql = "DELETE FROM leafImg WHERE caseNo = " + caseNum;
MMCDLV.                         Statement sment = conn.createStatement();
MMCDLVI.                        sment.executeUpdate(sql);
MMCDLVII.               }
MMCDLVIII.
MMCDLIX.                //get number of records, table = plant/leaf
MMCDLX.                 public int getCount(String table) throws SQLException{
MMCDLXI.                        Statement sment = conn.createStatement();
MMCDLXII.                       String sql = "SELECT COUNT(*) from " + table;
MMCDLXIII.                      ResultSet rs = sment.executeQuery(sql);
MMCDLXIV.                       int count = 0;
MMCDLXV.
MMCDLXVI.                       while(rs.next()){
MMCDLXVII.                              count = rs.getInt(1);
MMCDLXVIII.                     }
MMCDLXIX.
MMCDLXX.                        return count;
MMCDLXXI.               }
MMCDLXXII.
MMCDLXXIII.             //close connection to database
MMCDLXXIV.              public void closeConnection() throws SQLException{
MMCDLXXV.                       conn.close();
```

```
MMCDLXXVI.          }
MMCDLXXVII.
MMCDLXXVIII.        //main
MMCDLXXIX.          public static void main(String[] args) throws ClassNotFoundException, SQLException{
MMCDLXXX.                   String url = "", user = "", pass = "";
MMCDLXXXI.                  try{
MMCDLXXXII.                         @SuppressWarnings("resource")
MMCDLXXXIII.                        BufferedReader br = new BufferedReader(new FileReader("file/SQL.txt"));
MMCDLXXXIV.             String line = br.readLine();
MMCDLXXXV.              StringTokenizer tokens = new StringTokenizer(line,";");
MMCDLXXXVI.             url = tokens.nextToken();
MMCDLXXXVII.                    user = tokens.nextToken();
MMCDLXXXVIII.                   pass = tokens.nextToken();
MMCDLXXXIX.           } catch(IOException io){
MMCDXC.                     JOptionPane.showMessageDialog(null,"File containing connection details does not
          exist.");
MMCDXCI.              }
MMCDXCII.
MMCDXCIII.                  if(!url.equals("") || !user.equals("") || !pass.equals("")){
MMCDXCIV.                           SQL s = new SQL("jdbc:mysql://"+url, user,          pass);
MMCDXCV.                            s.connect();
MMCDXCVI.                           s.createPlantTable();
MMCDXCVII.                          s.createLeafTable();
MMCDXCVIII.                         s.createNetworkTable();
MMCDXCIX.                           s.createImageTable();
MMD.                s.closeConnection();
MMDI.                       System.out.println("Operation completed.");
MMDII.              }else{
MMDIII.                     System.out.println("Cannot perform operation");
MMDIV.              }
MMDV.         }
MMDVI.
MMDVII. }
```

MMDVIII.

MMDIX. **UserGUI.java**

```
MMDX.   package leafRec;
MMDXI.  import java.awt.*;
MMDXII. import java.awt.event.ActionEvent;
MMDXIII.        import java.awt.event.ActionListener;
MMDXIV.import java.awt.image.BufferedImage;
MMDXV.  import java.io.BufferedReader;
MMDXVI.         import java.io.ByteArrayInputStream;
MMDXVII.        import java.io.File;
MMDXVIII.       import java.io.FileNotFoundException;
MMDXIX.         import java.io.FileReader;
MMDXX. import java.io.IOException;
MMDXXI.         import java.nio.file.CopyOption;
```

```
MMDXXII.    import java.nio.file.Files;
MMDXXIII.   import java.nio.file.Path;
MMDXXIV.    import java.nio.file.Paths;
MMDXXV.     import java.nio.file.StandardCopyOption;
MMDXXVI.    import java.sql.Blob;
MMDXXVII.   import java.sql.SQLException;
MMDXXVIII.  import java.util.ArrayList;
MMDXXIX.    import java.util.StringTokenizer;
MMDXXX.
MMDXXXI.    import javax.imageio.ImageIO;
MMDXXXII.   import javax.sql.rowset.serial.SerialBlob;
MMDXXXIII.  import javax.swing.*;
MMDXXXIV.   import javax.swing.border.Border;
MMDXXXV.    import javax.swing.border.CompoundBorder;
MMDXXXVI.   import javax.swing.border.EmptyBorder;
MMDXXXVII.  import javax.swing.border.LineBorder;
MMDXXXVIII. import javax.swing.event.ListSelectionEvent;
MMDXXXIX.   import javax.swing.event.ListSelectionListener;
MMDXL.  import javax.swing.table.DefaultTableModel;
MMDXLI.
MMDXLII.    import org.neuroph.core.NeuralNetwork;
MMDXLIII.   import org.neuroph.core.learning.DataSet;
MMDXLIV.    import org.neuroph.core.learning.DataSetRow;
MMDXLV.
MMDXLVI.    public class UserGUI implements  ActionListener{
MMDXLVII.           private final int WIDTH = 850;
MMDXLVIII.          private final int HEIGHT = 500;
MMDXLIX.
MMDL.       //network
MMDLI.      private int inputCount = 40;
MMDLII.     private int outputCount = 15;
MMDLIII.
MMDLIV.     //results
MMDLV.      private float th = 0;
MMDLVI.
MMDLVII.            //canny
MMDLVIII.           private float low = 0.1f;
MMDLIX.             private float high = 10f;
MMDLX.
MMDLXI.             //connection parameters
MMDLXII.            private static String url;
MMDLXIII.           private static String user;
MMDLXIV.            private static String pass;
MMDLXV.
MMDLXVI.            //image files
MMDLXVII.           private String speciesFileName;
MMDLXVIII.
```

```
MMDLXIX.        //output
MMDLXX.         private ArrayList<OutputSpecies> os = new ArrayList<OutputSpecies>();
MMDLXXI.
MMDLXXII.       //main frame
MMDLXXIII.      private JFrame frame = new JFrame("Plant Leaf Recognition - Unregistered user");
MMDLXXIV.       private JTabbedPane tabbedPane = new JTabbedPane();
MMDLXXV.
MMDLXXVI.       //menu
MMDLXXVII.      JMenuBar menu = new JMenuBar();
MMDLXXVIII.     JMenu leafImg = new JMenu("Leaf");
MMDLXXIX.       JMenu canny = new JMenu("Canny");
MMDLXXX.        JMenu net = new JMenu("Network");
MMDLXXXI.       JMenu exitWindow = new JMenu("Exit");
MMDLXXXII.      JMenu results = new JMenu("Results");
MMDLXXXIII.     JMenuItem lowT = new JMenuItem("Low Threshold");
MMDLXXXIV.      JMenuItem highT = new JMenuItem("High Threshold");
MMDLXXXV.       JMenuItem detect = new JMenuItem("Detect Edge");
MMDLXXXVI.      JMenuItem verify = new JMenuItem("Verify");
MMDLXXXVII.     JMenuItem upNet = new JMenuItem("Update");
MMDLXXXVIII.    JMenuItem close = new JMenuItem("Close window");
MMDLXXXIX.      JMenuItem thold = new JMenuItem("Acceptance percentage");
MMDXC.
MMDXCI.         //panels OR tabs
MMDXCII.        private JPanel trPanel = new JPanel(new BorderLayout());
MMDXCIII.       private JPanel tsPanel = new JPanel(new BorderLayout());
MMDXCIV.
MMDXCV.         //other panels, l = left, r = right, i = img, b = button, c = collection
MMDXCVI.        private JPanel tsLeftPanel = new JPanel(new GridLayout(2,1));
MMDXCVII.       private JPanel tsCenPanel = new JPanel(new BorderLayout());
MMDXCVIII.      private JPanel tsRightPanel = new JPanel(new BorderLayout());
MMDXCIX.
MMDC.       private JPanel tsLCPanel = new JPanel(new BorderLayout());
MMDCI.      private JPanel tsLIPanel = new JPanel(new BorderLayout());
MMDCII.     private JPanel tsLIBPanel = new JPanel(new GridLayout(2,1));
MMDCIII.
MMDCIV.     private JPanel tsRBottomPanel  = new JPanel(new BorderLayout());
MMDCV.      private JPanel tsRBPanel = new JPanel(new GridLayout(1,2));
MMDCVI.         private JPanel tsRUPanel = new JPanel(new GridLayout(5,2));
MMDCVII.
MMDCVIII.       private JPanel trLeftPanel = new JPanel(new BorderLayout());
MMDCIX.         private JPanel trCenPanel = new JPanel(new BorderLayout());
MMDCX.      private JPanel trRightPanel = new JPanel(new BorderLayout());
MMDCXI.
MMDCXII.        private JPanel trRPropPanel = new JPanel(new GridLayout(6,2));
MMDCXIII.
MMDCXIV.        //scroll panes, c = collection, s = scroll, r = results
```

```
MMDCXV.     JScrollPane tsCSPane = new JScrollPane();
MMDCXVI.    JScrollPane tsRSPane = new JScrollPane();
MMDCXVII.   JScrollPane trCSPane = new JScrollPane();
MMDCXVIII.
MMDCXIX.    //labels
MMDCXX.     private JLabel tsSet = new JLabel("Testing Set");
MMDCXXI.    private JLabel trSet = new JLabel("Training Set");
MMDCXXII.   private JLabel newSpec = new JLabel("New Species");
MMDCXXIII.  private JLabel resLabel = new JLabel("Results");
MMDCXXIV.   private JLabel prop = new JLabel("Properties");
MMDCXXV.    private JLabel trProp = new JLabel("Properties");
MMDCXXVI.
MMDCXXVII.  private JLabel trName = new JLabel("Name");
MMDCXXVIII. private JLabel trType = new JLabel("Type");
MMDCXXIX.   private JLabel trArr = new JLabel("Arr");
MMDCXXX.    private JLabel trBlade = new JLabel("Blade");
MMDCXXXI.   private JLabel trMargin = new JLabel("Margin");
MMDCXXXII.  private JLabel trVena = new JLabel("Vena");
MMDCXXXIII.
MMDCXXXIV.  //text field
MMDCXXXV.   private JTextField tsType = new JTextField("");
MMDCXXXVI.  private JTextField tsArr = new JTextField("");
MMDCXXXVII. private JTextField tsBlade = new JTextField("");
MMDCXXXVIII.private JTextField tsMargin = new JTextField("");
MMDCXXXIX.  private JTextField tsVena = new JTextField("");
MMDCXL.
MMDCXLI.    //buttons
MMDCXLII.   private JButton tsLeftImg = new JButton("Image Preview");
MMDCXLIII.  private JButton tsRightImg = new JButton("Image Preview");
MMDCXLIV.   private JButton trRightImg = new JButton("Training Set Image Preview");
MMDCXLV.    private JButton loadImage = new JButton("Load Image");
MMDCXLVI.   private JButton addSpecies = new JButton("Add Species");
MMDCXLVII.  private JButton deleteSpecies = new JButton("Delete Species");
MMDCXLVIII. private JButton recognize = new JButton("Recognize");
MMDCXLIX.   private JButton reduce = new JButton("Reduce");
MMDCL.  private JButton clear = new JButton("Clear");
MMDCLI.
MMDCLII.    //table
MMDCLIII.   private JTable resultTable = new JTable();
MMDCLIV.
MMDCLV.     //DefaultModel
MMDCLVI.    private DefaultTableModel tableModel;
MMDCLVII.   private DefaultListModel<String> tsListModel = new DefaultListModel<String>();
MMDCLVIII.  private DefaultListModel<String> trListModel = new DefaultListModel<String>();
MMDCLIX.
MMDCLX.     //list c = collection, s = scroll
```

```
MMDCLXI.    private JList<String> tsCSList = new JList<String>(tsListModel);
MMDCLXII.   private JList<String> trCSList = new JList<String>(trListModel);
MMDCLXIII.
MMDCLXIV.   //constructor
MMDCLXV.    @SuppressWarnings("static-access")
MMDCLXVI.   public UserGUI(String url, String user, String pass){
MMDCLXVII.          this.url = url;
MMDCLXVIII.         this.user = user;
MMDCLXIX.           this.pass = pass;
MMDCLXX.    }
MMDCLXXI.
MMDCLXXII.  //create frame
MMDCLXXIII. public void createFrame() throws ClassNotFoundException, SQLException{
MMDCLXXIV.          tabbedPane.add("Training Set",trPanel);
MMDCLXXV.           tabbedPane.add("Testing Set",tsPanel);
MMDCLXXVI.
MMDCLXXVII.         menu.add(leafImg);
MMDCLXXVIII.        menu.add(net);
MMDCLXXIX.          menu.add(results);
MMDCLXXX.           menu.add(exitWindow);
MMDCLXXXI.
MMDCLXXXII.         leafImg.add(canny);
MMDCLXXXIII.        leafImg.add(verify);
MMDCLXXXIV.
MMDCLXXXV.          canny.add(lowT);
MMDCLXXXVI.         canny.add(highT);
MMDCLXXXVII.                canny.add(detect);
MMDCLXXXVIII.               net.add(upNet);
MMDCLXXXIX.         results.add(thold);
MMDCXC.             exitWindow.add(close);
MMDCXCI.
MMDCXCII.           lowT.addActionListener(this);
MMDCXCIII.          highT.addActionListener(this);
MMDCXCIV.           detect.addActionListener(this);
MMDCXCV.            verify.addActionListener(this);
MMDCXCVI.           upNet.addActionListener(this);
MMDCXCVII.          thold.addActionListener(this);
MMDCXCVIII.         close.addActionListener(this);
MMDCXCIX.
MMDCC.      initTr();
MMDCCI.             initTs();
MMDCCII.
MMDCCIII.           createTrTab();
MMDCCIV.            createTsTab();
MMDCCV.
MMDCCVI.
MMDCCVII.           Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
```

```
MMDCCVIII.                      JWindow window = new JWindow();
MMDCCIX.            ImageFade fade = new ImageFade("img/uuser.gif",0);
MMDCCX.             window.getContentPane().add(fade, SwingConstants.CENTER);
MMDCCXI.            window.setBounds(screenSize.width/2 - (510/2), screenSize.height/2 - (176/2),510,176);
MMDCCXII.           window.setVisible(true);
MMDCCXIII.          try {
MMDCCXIV.                Thread.sleep(5000);
MMDCCXV.            } catch (InterruptedException e) {
MMDCCXVI.               System.out.println("An error has occurred.");
MMDCCXVII.          }
MMDCCXVIII.         window.setVisible(false);
MMDCCXIX.                       frame.add(tabbedPane);
MMDCCXX.                        frame.setJMenuBar(menu);
MMDCCXXI.                       frame.pack();
MMDCCXXII.                      frame.setVisible(true);
MMDCCXXIII.                     frame.setSize(WIDTH, HEIGHT);
MMDCCXXIV.                      frame.setResizable(false);
MMDCCXXV.                       frame.setLocation(screenSize.width/2 - (WIDTH/2), screenSize.height/2 - (HEIGHT/2));
MMDCCXXVI.                      frame.setIconImage(new
        ImageIcon(getClass().getResource("img/leafIcon.png")).getImage());
MMDCCXXVII.                     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
MMDCCXXVIII.                    window.dispose();
MMDCCXXIX.              }
MMDCCXXX.
MMDCCXXXI.              //initialize vars in training set tab
MMDCCXXXII.             public void initTr() throws ClassNotFoundException, SQLException{
MMDCCXXXIII.                    Border line = new LineBorder(Color.BLACK);
MMDCCXXXIV.                     Border margin = new EmptyBorder(5, 15, 5, 15);
MMDCCXXXV.                      Border compound = new CompoundBorder(line, margin);
MMDCCXXXVI.
MMDCCXXXVII.                            trRightImg.setContentAreaFilled(false);
MMDCCXXXVIII.                           trRightImg.setOpaque(false);
MMDCCXXXIX.                     trRightImg.setFocusable(false);
MMDCCXL.                        trRightImg.setBorderPainted(true);
MMDCCXLI.                       trRightImg.setForeground(Color.BLACK);
MMDCCXLII.                      trRightImg.setBackground(Color.WHITE);
MMDCCXLIII.                     trRightImg.setBorder(compound);
MMDCCXLIV.
MMDCCXLV.                       trLeftPanel.setBackground(Color.WHITE);
MMDCCXLVI.                      trCenPanel.setBackground(Color.WHITE);
MMDCCXLVII.                     trRightPanel.setBackground(Color.WHITE);
MMDCCXLVIII.
MMDCCXLIX.                      trCSPane.setBackground(Color.WHITE);
MMDCCL.                         trRightPanel.setBackground(Color.WHITE);
MMDCCLI.                        trRPropPanel.setBackground(Color.WHITE);
MMDCCLII.
MMDCCLIII.                      trRightImg.setBorder(compound);
```

```
MMDCCLIV.                       trSet.setBorder(compound);
MMDCCLV.                        trProp.setBorder(compound);
MMDCCLVI.
MMDCCLVII.                      displayTrSet();
MMDCCLVIII.                     trCSList.setFixedCellHeight(30);
MMDCCLIX.           trCSList.setFixedCellWidth(235);
MMDCCLX.            trCSList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
MMDCCLXI.           trCSList.addListSelectionListener(new ListSelectionListener() {
MMDCCLXII.                  public void valueChanged(ListSelectionEvent event) {
MMDCCLXIII.                     if(trCSList.getSelectedIndex() > -1){
MMDCCLXIV.                          SQL s = new SQL
MMDCCLXV.                                       (url,user,pass);
MMDCCLXVI.                          Species sp = new Species();
MMDCCLXVII.                         try {
MMDCCLXVIII.                                        s.connect();
MMDCCLXIX.                                          sp =
        s.querySpec(trCSList.getSelectedValue());
MMDCCLXX.                                           trName.setText(sp.getName());
MMDCCLXXI.                                          trType.setText(sp.getType());
MMDCCLXXII.                                         trArr.setText(sp.getArrangement());
MMDCCLXXIII.                                        trBlade.setText(sp.getBlade());
MMDCCLXXIV.                                         trMargin.setText(sp.getMargin());
MMDCCLXXV.                                          trVena.setText(sp.getVenation());
MMDCCLXXVI.                                     } catch (ClassNotFoundException | SQLException e) {
MMDCCLXXVII.
            JOptionPane.showMessageDialog(frame,"An error occurred!");
MMDCCLXXVIII.                                       }
MMDCCLXXIX.
MMDCCLXXX.                          try {
MMDCCLXXXI.                                         byte[] imgData = sp.getBlob().getBytes(1,
        (int) sp.getBlob().length());
MMDCCLXXXII.                                    BufferedImage img = ImageIO.read(new
        ByteArrayInputStream(imgData));
MMDCCLXXXIII.                                   ImageIcon icon = new ImageIcon(img);
MMDCCLXXXIV.                                            Image image =
        icon.getImage().getScaledInstance(350, 350, Image.SCALE_SMOOTH);
MMDCCLXXXV.                                             icon.setImage(image);
MMDCCLXXXVI.                                            trRightImg.setIcon(icon);
MMDCCLXXXVII.                                   trRightImg.setText(""); //set the label to null ready for
        the image to be displayed
MMDCCLXXXVIII.                                          } catch (SQLException | IOException e) {
MMDCCLXXXIX.
            JOptionPane.showMessageDialog(frame,"An error occurred!");
MMDCCXC.                                        }
MMDCCXCI.                           }
MMDCCXCII.              }});
MMDCCXCIII.         trCSPane.setViewportView(trCSList);
MMDCCXCIV.          trCSPane.setBorder(null);
MMDCCXCV.               }
```

```
MMDCCXCVI.
MMDCCXCVII.     //create training tab
MMDCCXCVIII.    public void createTrTab(){
MMDCCXCIX.              trPanel.add(trLeftPanel, BorderLayout.WEST);
MMDCCC.                 trPanel.add(trCenPanel, BorderLayout.CENTER);
MMDCCCI.                trPanel.add(trRightPanel, BorderLayout.EAST);
MMDCCCII.
MMDCCCIII.              trLeftPanel.add(trSet, BorderLayout.NORTH);
MMDCCCIV.               trLeftPanel.add(trCSPane, BorderLayout.CENTER);
MMDCCCV.
MMDCCCVI.               trCenPanel.add(trRightImg, BorderLayout.CENTER);
MMDCCCVII.
MMDCCCVIII.             trRightPanel.add(trProp, BorderLayout.NORTH);
MMDCCCIX.               trRightPanel.add(trRPropPanel, BorderLayout.CENTER);
MMDCCCX.
MMDCCCXI.               trRPropPanel.add(new JLabel("Leaf Name: "));
MMDCCCXII.              trRPropPanel.add(trName);
MMDCCCXIII.             trRPropPanel.add(new JLabel("Leaf Type: "));
MMDCCCXIV.              trRPropPanel.add(trType);
MMDCCCXV.               trRPropPanel.add(new JLabel("Leaf Arrangement: "));
MMDCCCXVI.              trRPropPanel.add(trArr);
MMDCCCXVII.             trRPropPanel.add(new JLabel("Leaf Blade: "));
MMDCCCXVIII.            trRPropPanel.add(trBlade);
MMDCCCXIX.              trRPropPanel.add(new JLabel("Leaf Margin: "));
MMDCCCXX.               trRPropPanel.add(trMargin);
MMDCCCXXI.              trRPropPanel.add(new JLabel("Leaf Venation: "));
MMDCCCXXII.             trRPropPanel.add(trVena);
MMDCCCXXIII.    }
MMDCCCXXIV.
MMDCCCXXV.      //display training set collection
MMDCCCXXVI.     public void displayTrSet() throws ClassNotFoundException, SQLException{
MMDCCCXXVII.                    trListModel.clear();
MMDCCCXXVIII.                   ArrayList<Species> spec = new ArrayList<Species>();
MMDCCCXXIX.             SQL s = new SQL(url, user, pass);
MMDCCCXXX.              s.connect();
MMDCCCXXXI.             spec = s.queryPlant();
MMDCCCXXXII.                    for(Species sp : spec){
MMDCCCXXXIII.                           trListModel.addElement(sp.getName());
MMDCCCXXXIV.                    }
MMDCCCXXXV.             }
MMDCCCXXXVI.
MMDCCCXXXVII.           //initialize vars in testing set tab
MMDCCCXXXVIII.          public void initTs(){
MMDCCCXXXIX.                    Border line = new LineBorder(Color.BLACK);
MMDCCCXL.               Border margin = new EmptyBorder(5, 15, 5, 15);
MMDCCCXLI.              Border compound = new CompoundBorder(line, margin);
```

```
MMDCCCXLII.
MMDCCCXLIII.        tsLeftImg.setContentAreaFilled(false);
MMDCCCXLIV.         tsLeftImg.setOpaque(false);
MMDCCCXLV.          tsLeftImg.setFocusable(false);
MMDCCCXLVI.         tsLeftImg.setBorderPainted(true);
MMDCCCXLVII.                tsLeftImg.setForeground(Color.BLACK);
MMDCCCXLVIII.               tsLeftImg.setBackground(Color.WHITE);
MMDCCCXLIX.         tsLeftImg.setBorder(compound);
MMDCCCL.
MMDCCCLI.           tsRightImg.setContentAreaFilled(false);
MMDCCCLII.          tsRightImg.setOpaque(false);
MMDCCCLIII.         tsRightImg.setFocusable(false);
MMDCCCLIV.          tsRightImg.setBorderPainted(true);
MMDCCCLV.           tsRightImg.setForeground(Color.BLACK);
MMDCCCLVI.          tsRightImg.setBackground(Color.WHITE);
MMDCCCLVII.         tsRightImg.setBorder(compound);
MMDCCCLVIII.
MMDCCCLIX.          tsLeftPanel.setBackground(Color.WHITE);
MMDCCCLX.           tsCenPanel.setBackground(Color.WHITE);
MMDCCCLXI.          tsRightPanel.setBackground(Color.WHITE);
MMDCCCLXII.
MMDCCCLXIII.        tsLIPanel.setBackground(Color.WHITE);
MMDCCCLXIV.         tsLIBPanel.setBackground(Color.WHITE);
MMDCCCLXV.          tsLCPanel.setBackground(Color.WHITE);
MMDCCCLXVI.
MMDCCCLXVII.                tsRSPane.setBackground(Color.WHITE);
MMDCCCLXVIII.               tsRightPanel.setBackground(Color.WHITE);
MMDCCCLXIX.         tsRBottomPanel.setBackground(Color.WHITE);
MMDCCCLXX.          tsRBPanel.setBackground(Color.WHITE);
MMDCCCLXXI.         tsRUPanel.setBackground(Color.WHITE);
MMDCCCLXXII.
MMDCCCLXXIII.               tsLeftImg.setBorder(compound);
MMDCCCLXXIV.                tsRightImg.setBorder(compound);
MMDCCCLXXV.                 tsSet.setBorder(compound);
MMDCCCLXXVI.                newSpec.setBorder(compound);
MMDCCCLXXVII.               resLabel.setBorder(compound);
MMDCCCLXXVIII.              prop.setBorder(compound);
MMDCCCLXXIX.
MMDCCCLXXX.                 tableModel = new DefaultTableModel(new Object[]{"Species","Match
        (%)"},0);
MMDCCCLXXXI.                initTable();
MMDCCCLXXXII.               resultTable.setModel(tableModel);
MMDCCCLXXXIII.              tsRSPane.setViewportView(resultTable);
MMDCCCLXXXIV.
            resultTable.setPreferredScrollableViewportSize(resultTable.getPreferredSize());
MMDCCCLXXXV.
MMDCCCLXXXVI.               retrieveFiles();
```

```
MMDCCCLXXXVII.                          tsCSList.setFixedCellHeight(30);
MMDCCCLXXXVIII.          tsCSList.setFixedCellWidth(235);
MMDCCCLXXXIX.            tsCSList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
MMDCCCXC.          tsCSList.addListSelectionListener(new ListSelectionListener() {
MMDCCCXCI.               public void valueChanged(ListSelectionEvent event) {
MMDCCCXCII.                    if(tsListModel.size()>0 && tsCSList.getSelectedValue()!=null){
MMDCCCXCIII.                                                    speciesFileName =
        "species/"+tsCSList.getSelectedValue().toString();
MMDCCCXCIV.                                                     changePreviewLeft(speciesFileName);
MMDCCCXCV.                                                      changePreviewRight(speciesFileName);
MMDCCCXCVI.                                                     recognize.setEnabled(true);
MMDCCCXCVII.                                                            detect.setEnabled(true);
MMDCCCXCVIII.                                                           verify.setEnabled(true);
MMDCCCXCIX.                    }
MMCM.            });
MMCMI.       tsCSPane.setViewportView(tsCSList);
MMCMII.      tsCSPane.setBorder(null);
MMCMIII.
MMCMIV.                  }
MMCMV.
MMCMVI.                  //initialize results tab
MMCMVII.                 public void initTable(){
MMCMVIII.                      for(int i = 0; i < outputCount; i++){
MMCMIX.                                tableModel.addRow(new Object[]{"",""});
MMCMX.                   }
MMCMXI.                  }
MMCMXII.
MMCMXIII.                //disable/enable reduction
MMCMXIV.                 public void reductionProp(boolean flag){
MMCMXV.                        tsType.setEditable(flag);
MMCMXVI.                       tsArr.setEditable(flag);
MMCMXVII.                      tsBlade.setEditable(flag);
MMCMXVIII.                     tsMargin.setEditable(flag);
MMCMXIX.                       tsVena.setEditable(flag);
MMCMXX.                        reduce.setEnabled(flag);
MMCMXXI.                       clear.setEnabled(flag);
MMCMXXII.                }
MMCMXXIII.
MMCMXXIV.                //create testing tab
MMCMXXV.                 public void createTsTab(){
MMCMXXVI.                      loadImage.addActionListener(this);
MMCMXXVII.                     addSpecies.addActionListener(this);
MMCMXXVIII.                    deleteSpecies.addActionListener(this);
MMCMXXIX.                      recognize.addActionListener(this);
MMCMXXX.                       reduce.addActionListener(this);
MMCMXXXI.                      clear.addActionListener(this);
MMCMXXXII.                     addSpecies.setEnabled(false);
```

```
MMCMXXXIII.                 recognize.setEnabled(false);
MMCMXXXIV.                  detect.setEnabled(false);
MMCMXXXV.                   verify.setEnabled(false);
MMCMXXXVI.                  reductionProp(false);
MMCMXXXVII.
MMCMXXXVIII.                        tsPanel.add(tsLeftPanel, BorderLayout.WEST);
MMCMXXXIX.                  tsPanel.add(tsCenPanel, BorderLayout.CENTER);
MMCMXL.                     tsPanel.add(tsRightPanel, BorderLayout.EAST);
MMCMXLI.
MMCMXLII.                   tsLeftPanel.add(tsLIPanel);
MMCMXLIII.                  tsLeftPanel.add(tsLCPanel);
MMCMXLIV.
MMCMXLV.                    tsLIPanel.add(newSpec, BorderLayout.NORTH);
MMCMXLVI.                   tsLIPanel.add(tsLeftImg, BorderLayout.CENTER);
MMCMXLVII.                  tsLIPanel.add(tsLIBPanel, BorderLayout.SOUTH);
MMCMXLVIII.
MMCMXLIX.                   tsLIBPanel.add(loadImage);
MMCML.              tsLIBPanel.add(addSpecies);
MMCMLI.
MMCMLII.                    tsLCPanel.add(tsSet, BorderLayout.NORTH);
MMCMLIII.                   tsLCPanel.add(tsCSPane, BorderLayout.CENTER);
MMCMLIV.                    tsLCPanel.add(deleteSpecies,BorderLayout.SOUTH);
MMCMLV.
MMCMLVI.                    tsCenPanel.add(tsRightImg, BorderLayout.CENTER);
MMCMLVII.                   tsCenPanel.add(recognize, BorderLayout.SOUTH);
MMCMLVIII.
MMCMLIX.                    tsRightPanel.add(resLabel, BorderLayout.NORTH);
MMCMLX.                     tsRightPanel.add(tsRSPane, BorderLayout.CENTER);
MMCMLXI.                    tsRightPanel.add(tsRBottomPanel, BorderLayout.SOUTH);
MMCMLXII.
MMCMLXIII.                  tsRBottomPanel.add(prop, BorderLayout.NORTH);
MMCMLXIV.                   tsRBottomPanel.add(tsRUPanel, BorderLayout.CENTER);
MMCMLXV.                    tsRBottomPanel.add(tsRBPanel, BorderLayout.SOUTH);
MMCMLXVI.
MMCMLXVII.                  tsRUPanel.add(new JLabel("Leaf Type: "));
MMCMLXVIII.                 tsRUPanel.add(tsType);
MMCMLXIX.                   tsRUPanel.add(new JLabel("Leaf Arrangement: "));
MMCMLXX.                    tsRUPanel.add(tsArr);
MMCMLXXI.                   tsRUPanel.add(new JLabel("Leaf Blade: "));
MMCMLXXII.                  tsRUPanel.add(tsBlade);
MMCMLXXIII.                 tsRUPanel.add(new JLabel("Leaf Margin: "));
MMCMLXXIV.                  tsRUPanel.add(tsMargin);
MMCMLXXV.                   tsRUPanel.add(new JLabel("Leaf Venation: "));
MMCMLXXVI.                  tsRUPanel.add(tsVena);
MMCMLXXVII.
MMCMLXXVIII.                        tsRBPanel.add(reduce);
```

```
MMCMLXXIX.                  tsRBPanel.add(clear);
MMCMLXXX.
MMCMLXXXI.          }
MMCMLXXXII.
MMCMLXXXIII.                //listener
MMCMLXXXIV.         public void actionPerformed(ActionEvent e) {
MMCMLXXXV.                  if(e.getSource() == loadImage){
MMCMLXXXVI.                                 JFileChooser chooser = new JFileChooser();
MMCMLXXXVII.                                try {
MMCMLXXXVIII.                                       File f = new File(new File("Choose
        file").getCanonicalPath());
MMCMLXXXIX.                                   chooser.setSelectedFile(f);
MMCMXC.                                 int result = chooser.showOpenDialog(frame);
MMCMXCI.                                if(result == JFileChooser.APPROVE_OPTION){
MMCMXCII.                                   File curFile = chooser.getSelectedFile();
MMCMXCIII.                                  speciesFileName = curFile.toString();
MMCMXCIV.
MMCMXCV.                                    int index = speciesFileName.indexOf(".");
MMCMXCVI.                                   if(speciesFileName.substring(index+1).equals("jpg")){
MMCMXCVII.                                          changePreviewLeft(speciesFileName);

MMCMXCVIII.                                         changePreviewRight(speciesFileName);
MMCMXCIX.                                           addSpecies.setEnabled(true);
MMM.                                    recognize.setEnabled(false);
MMMI.                                       verify.setEnabled(true);
MMMII.                                      reductionProp(false);
MMMIII.                                 }else{
MMMIV.                                      JOptionPane.showMessageDialog(frame,"Please select file of type
        jpg");
MMMV.                                   }
MMMVI.                                }
MMMVII.                     } catch (IOException e1) {
MMMVIII.                                    JOptionPane.showMessageDialog(frame,"Cannot perform
        operation.");
MMMIX.                      }
MMMX.               }else if(e.getSource()==addSpecies){
MMMXI.                      //copy file
MMMXII.                     String curFile=speciesFileName;
MMMXIII.                            String tempLeaf
        =curFile.toString().substring(curFile.toString().lastIndexOf("\\")+1);
MMMXIV.                             if(speciesFileName!=""){
MMMXV.                              if(!tsListModel.contains(tempLeaf)){
MMMXVI.                                         try {
MMMXVII.                                                CopyOption[] options = new CopyOption[]{
MMMXVIII.
        StandardCopyOption.REPLACE_EXISTING,
MMMXIX.
        StandardCopyOption.COPY_ATTRIBUTES
MMMXX.                                                          };
```

```
MMMXXI.
            Files.copy(Paths.get(speciesFileName),Paths.get("species/"+tempLeaf),options);
MMMXXII.                                                    tsListModel.addElement(tempLeaf);
MMMXXIII.                                                } catch (IOException e1) {
MMMXXIV.
            JOptionPane.showMessageDialog(frame,"Cannot perform operation.");
MMMXXV.                                                  }
MMMXXVI.                                            }else{
MMMXXVII.                                                JOptionPane.showMessageDialog(frame,"File already
        exists.");
MMMXXVIII.                                          }
MMMXXIX.                                    }
MMMXXX.
MMMXXXI.                                    if(!tsListModel.contains(tempLeaf)){
MMMXXXII.                                           if(tsListModel.size()>0)
        tsCSList.setSelectedIndex(tsListModel.size()-1);
MMMXXXIII.                                          addSpecies.setEnabled(false);
MMMXXXIV.                                           recognize.setEnabled(true);
MMMXXXV.                                            detect.setEnabled(true);
MMMXXXVI.                                   }
MMMXXXVII.                              }else if(e.getSource()==deleteSpecies){
MMMXXXVIII.                                 //copy file
MMMXXXIX.                                   String file = tsCSList.getSelectedValue().toString();
MMMXL.                                  int response = JOptionPane.showConfirmDialog(frame, "Do you want to continue?",
        "Confirm",
MMMXLI.                                     JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
MMMXLII.                                  if (response == JOptionPane.YES_OPTION) {
MMMXLIII.                                       int i = tsCSList.getSelectedIndex();
MMMXLIV.                                        if(tsListModel.size() >1){
MMMXLV.                                             if(i!=0)
MMMXLVI.                                                    tsCSList.setSelectedIndex(i);
MMMXLVII.                                           else
MMMXLVIII.                                                  tsCSList.setSelectedIndex(i-1);
MMMXLIX.                                        }
MMML.                                           File f1 = new File("species/"+file);
MMMLI.                                          boolean success = f1.delete();
MMMLII.                                         if (!success){
MMMLIII.
            JOptionPane.showMessageDialog(frame,"Deletion failed.");
MMMLIV.                                                 }else{
MMMLV.                                                  JOptionPane.showMessageDialog(frame, "File deleted.");
MMMLVI.                                                 }
MMMLVII.                                                retrieveFiles();
MMMLVIII.                                               if(tsListModel.size() >0){
MMMLIX.                                                     if(i!=0)
MMMLX.                                                      tsCSList.setSelectedIndex(i-1);
MMMLXI.                                                     else
MMMLXII.                                                            tsCSList.setSelectedIndex(i);
MMMLXIII.                                               }else{
```

MMMLXIV.
MMMLXV.
MMMLXVI.
MMMLXVII.
MMMLXVIII.
MMMLXIX.
MMMLXX.
MMMLXXI.
MMMLXXII.
MMMLXXIII.
MMMLXXIV.
MMMLXXV.
MMMLXXVI.
MMMLXXVII.
MMMLXXVIII.
MMMLXXIX.
MMMLXXX.
MMMLXXXI.
MMMLXXXII.
MMMLXXXIII.
MMMLXXXIV.
MMMLXXXV.
MMMLXXXVI.
MMMLXXXVII.
MMMLXXXVIII.
MMMLXXXIX.
MMMXC.
MMMXCI.
MMMXCII.
MMMXCIII.
MMMXCIV.
MMMXCV.
MMMXCVI.
MMMXCVII.
MMMXCVIII.
MMMXCIX.
MMMC.
MMMCI.
MMMCII.
MMMCIII.
MMMCIV.
MMMCV.
MMMCVI.
MMMCVII.
MMMCVIII.
MMMCIX.

```
                                        tsLeftImg.setIcon(null);
                                        tsRightImg.setIcon(null);
                                        tsRightImg.setText("Image Preview");
                                        tsLeftImg.setText("Image Preview");
                                        recognize.setEnabled(false);
                                        detect.setEnabled(false);
                                        deleteSpecies.setEnabled(false);
                        }
                }
        }else if(e.getSource() == recognize){
                try {
                        recognize();
                } catch (ClassNotFoundException | SQLException | IOException e1) {
                        JOptionPane.showMessageDialog(frame,"An error occurred!");
                }
        }else if(e.getSource() == reduce){
                try {
                        reduce();
                } catch (ClassNotFoundException | SQLException e1) {
                        JOptionPane.showMessageDialog(frame,"An error occurred!");
                }
        }else if(e.getSource() == clear){
                tsType.setText("");
                tsArr.setText("");
                        tsBlade.setText("");
                tsMargin.setText("");
        tsVena.setText("");
        }else if(e.getSource() == upNet){
                try {
                        downloadNetworkFile();
                        displayTrSet();
                        trCSList.setSelectedIndex(0);
                } catch (ClassNotFoundException | SQLException | IOException e1) {
                        JOptionPane.showMessageDialog(frame,"An error occurred!");
                }
        }else if(e.getSource() == thold){
        boolean flag = false;
        float t = th;
        try{
                        t = Float.parseFloat(JOptionPane.showInputDialog(frame,"Enter
                acceptance percentage: ",th));
                }catch(NumberFormatException n){
                JOptionPane.showMessageDialog(frame, "Invalid input!");
                        flag = true;
                }catch(NullPointerException n){
                        flag = true;
                }
```

```
MMMCX.          if(!flag) th= t;
MMMCXI.         }else if(e.getSource() == close){
MMMCXII.                int response = JOptionPane.showConfirmDialog(frame, "Do you want to
        exit?", "Confirm",
MMMCXIII.                   JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
MMMCXIV.                if (response == JOptionPane.YES_OPTION) {
MMMCXV.                         System.exit(0);
MMMCXVI.                }
MMMCXVII.       }else if(e.getSource() == lowT){
MMMCXVIII.              boolean flag = false;
MMMCXIX.                float l = low;
MMMCXX.                 try{
MMMCXXI.                        l = Float.parseFloat(JOptionPane.showInputDialog(frame,"Enter
        low threshold: ",low));
MMMCXXII.               }catch(NumberFormatException n){
MMMCXXIII.                      JOptionPane.showMessageDialog(frame, "Invalid input!");
MMMCXXIV.                       flag = true;
MMMCXXV.                }catch(NullPointerException n){
MMMCXXVI.                       flag = true;
MMMCXXVII.              }
MMMCXXVIII.             if(!flag) low = l;
MMMCXXIX.       }else if(e.getSource() == highT){
MMMCXXX.                boolean flag = false;
MMMCXXXI.               float h = high;
MMMCXXXII.              try{
MMMCXXXIII.                     h = Float.parseFloat(JOptionPane.showInputDialog(frame,"Enter
        high threshold: ",high));
MMMCXXXIV.              }catch(NumberFormatException n){
MMMCXXXV.                       JOptionPane.showMessageDialog(frame, "Invalid input!");
MMMCXXXVI.                      flag = true;
MMMCXXXVII.             }catch(NullPointerException n){
MMMCXXXVIII.                            flag = true;
MMMCXXXIX.              }
MMMCXL.                 if(!flag) high = h;
MMMCXLI.        }else if(e.getSource() == detect){
MMMCXLII.               CannyEdgeDetector c = extractFeatures();
MMMCXLIII.              c.process();
MMMCXLIV.               BufferedImage edges = c.getEdgesImage();
MMMCXLV.                Image image = edges.getScaledInstance(350, 350, Image.SCALE_SMOOTH);
MMMCXLVI.         tsRightImg.setIcon(new ImageIcon(image));
MMMCXLVII.        tsRightImg.setText("");
MMMCXLVIII.     }else if(e.getSource() == verify){
MMMCXLIX.               try {
MMMCL.                  Path path = Paths.get(speciesFileName);
MMMCLI.                         byte[] b = Files.readAllBytes(path);
MMMCLII.                        Blob blob = null;
MMMCLIII.                       blob = new SerialBlob(b);
```

```
                                        SQL s = new SQL(url, user, pass);
                                        s.connect();
                                        s.addToLeafImg(blob);
                                        s.closeConnection();

                                        JOptionPane.showMessageDialog(frame,"Plant leaf pending for
        verification by expert. Thank you!");
                                } catch (IOException | SQLException | ClassNotFoundException e1) {
                                        JOptionPane.showMessageDialog(frame,"Verification failed!");
                                }
                        }
        }

        //query from database, download trained network file
        public void downloadNetworkFile() throws ClassNotFoundException, SQLException, IOException{
                SQL s = new SQL(url,user,pass);
                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(frame, "Update successful.");

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

                //retrieve filenames
                public void retrieveFiles(){
                        tsListModel.clear();

                File[] files = new File("species/").listFiles();
                for (File file : files) {
                        if (file.isFile()) {
                                tsListModel.addElement(file.getName());
                   }
                }
        }

        //change image preview-right
```

```
public void changePreviewRight(String file){
                ImageIcon icon = new ImageIcon(file);
    Image image = icon.getImage().getScaledInstance(350, 350, Image.SCALE_SMOOTH);
    icon.setImage(image);
    tsRightImg.setIcon(icon);
    tsRightImg.setText("");
}

//change image preview-left
public void changePreviewLeft(String file){
    ImageIcon icon2 = new ImageIcon(file);
    Image image2 = icon2.getImage().getScaledInstance(150, 150, Image.SCALE_SMOOTH);
    icon2.setImage(image2);
    tsLeftImg.setIcon(icon2);
    tsLeftImg.setText("");
}

//extract image moments
@SuppressWarnings("static-access")
public ArrayList<Double> getMoments(CannyEdgeDetector c){
        c.preprocess();
        int w = c.getWidth();
        int h = c.getHeight();
        int[] data = c.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> moments = new ArrayList<Double>();
        Moments m = new Moments();
                moments.add(m.getNormalizedCentralMoment(0, 0, matrix));
        moments.add(m.getNormalizedCentralMoment(1, 0, matrix));
        moments.add(m.getNormalizedCentralMoment(0, 1, matrix));
                moments.add(m.getNormalizedCentralMoment(1, 1, matrix));


                return moments;
        }

//extract image radii
public ArrayList<Double> getRadii(CannyEdgeDetector c){
        ArrayList<Double> radii = new ArrayList<Double>();
        c.process();
        CentroidRadii cr = new CentroidRadii(c.getCentroidX(),c.getCentroidY(),c.getWidth());
```

```
MMMCCXLVI.              cr.setOutline(c.getOutline());
MMMCCXLVII.             cr.determineRadii();
MMMCCXLVIII.            radii = cr.getRadiiSet();
MMMCCXLIX.              BufferedImage edges = c.getEdgesImage();
MMMCCL.                 Image image = edges.getScaledInstance(350, 350, Image.SCALE_SMOOTH);
MMMCCLI.              tsRightImg.setIcon(new ImageIcon(image));
MMMCCLII.             tsRightImg.setText("");
MMMCCLIII.              return radii;
MMMCCLIV.           }
MMMCCLV.
MMMCCLVI.               //extract features(moments,radii)
MMMCCLVII.          public CannyEdgeDetector extractFeatures(){
MMMCCLVIII.             CannyEdgeDetector c = new CannyEdgeDetector();
MMMCCLIX.               c.setLowThreshold(low);
MMMCCLX.                c.setHighThreshold(high);
MMMCCLXI.               try {
MMMCCLXII.                      c.setSourceImage(ImageIO.read(new File(speciesFileName)));
MMMCCLXIII.             } catch (IOException e) {
MMMCCLXIV.                      e.printStackTrace();
MMMCCLXV.               }
MMMCCLXVI.              return c;
MMMCCLXVII.         }
MMMCCLXVIII.
MMMCCLXIX.          //load neural net configuration and test
MMMCCLXX.           public void testNeuralNet() throws ClassNotFoundException, IOException{
MMMCCLXXI.
MMMCCLXXII.             File f = new File("file/Weights.nnet");
MMMCCLXXIII.            if(f.exists()){
MMMCCLXXIV.                     ArrayList<Double> feat = new ArrayList<Double>();
MMMCCLXXV.                      os.removeAll(os);
MMMCCLXXVI.                             double[] in = new double[inputCount];
MMMCCLXXVII.                            int i = 0;
MMMCCLXXVIII.
MMMCCLXXIX.                             CannyEdgeDetector c = extractFeatures();
MMMCCLXXX.                      feat.addAll(getMoments(c));
MMMCCLXXXI.
MMMCCLXXXII.                            feat.addAll(getRadii(c));
MMMCCLXXXIII.                           for(double a : feat){
MMMCCLXXXIV.                                    in[i] = a;
MMMCCLXXXV.                                     ++i;
MMMCCLXXXVI.                            }
MMMCCLXXXVII.                           DataSet testSet = new DataSet(inputCount);
MMMCCLXXXVIII.                          testSet.addRow(in);
MMMCCLXXXIX.                NeuralNetwork mlp = NeuralNetwork.load("file/Weights.nnet");
MMMCCXC.
MMMCCXCI.               for(DataSetRow testSetRow : testSet.getRows()) {
MMMCCXCII.                mlp.setInput(testSetRow.getInput());
```

```
MMMCCXCIII.                 mlp.calculate();
MMMCCXCIV.
MMMCCXCV.                   double[] networkOutput = mlp.getOutput();
MMMCCXCVI.                  for(i = 0; i < networkOutput.length; i++){
MMMCCXCVII.                     OutputSpecies s = new OutputSpecies();
MMMCCXCVIII.                            s.setID(i+1);
MMMCCXCIX.                      s.setOutput(networkOutput[i]*100);
MMMCCC.                         os.add(s);
MMMCCCI.                    }
MMMCCCII.               }
MMMCCCIII.                              reductionProp(true);
MMMCCCIV.                       }else{
MMMCCCV.                                JOptionPane.showMessageDialog(frame,"Network file not found. Please
        update trained network file.");
MMMCCCVI.                       }
MMMCCCVII.          }
MMMCCCVIII.
MMMCCCIX.           //sort output species
MMMCCCX.            public void sortSpecies(){
MMMCCCXI.                   for(int i = 0; i < os.size(); i++){
MMMCCCXII.                      for(int j = 0; j < os.size()-1; j++){
MMMCCCXIII.                             if(os.get(j).getOutput() < os.get(j+1).getOutput()){
MMMCCCXIV.                                      OutputSpecies temp = os.get(j+1);
MMMCCCXV.                                       os.set(j+1,os.get(j));
MMMCCCXVI.                                      os.set(j,temp);
MMMCCCXVII.                             }
MMMCCCXVIII.                            }
MMMCCCXIX.                  }
MMMCCCXX.           }
MMMCCCXXI.
MMMCCCXXII.         //initialize table
MMMCCCXXIII.                public void emptyTable(){
MMMCCCXXIV.                 for(int i = 0; i < outputCount; i++){
MMMCCCXXV.                  tableModel.setValueAt("",i,0);
MMMCCCXXVI.                             tableModel.setValueAt("",i,1);
MMMCCCXXVII.                  }
MMMCCCXXVIII.               }
MMMCCCXXIX.
MMMCCCXXX.          //populate results table
MMMCCCXXXI.                 public void displayMatch() throws SQLException, ClassNotFoundException{
MMMCCCXXXII.                            emptyTable();
MMMCCCXXXIII.                           SQL s = new SQL(url,user,pass);
MMMCCCXXXIV.                            s.connect();
MMMCCCXXXV.                             for(int i = 0; i < os.size(); i++){
MMMCCCXXXVI.                            Species spec = s.queryByID(os.get(i).getID());
MMMCCCXXXVII.                           if(os.get(i).getOutput() < th) break;
MMMCCCXXXVIII.                          if(tableModel.getRowCount() < os.size()){
```

```
MMMCCCXXXIX.                                    tableModel.addRow(new Object[]{"",""});
MMMCCCXL.                               outputCount++;
MMMCCCXLI.                          }
MMMCCCXLII.                         tableModel.setValueAt(spec.getName(),i,0);
MMMCCCXLIII.                        tableModel.setValueAt(String.format("%.10f",os.get(i).getOutput()),i,1);
MMMCCCXLIV.                   }
MMMCCCXLV.                          s.closeConnection();
MMMCCCXLVI.                 }
MMMCCCXLVII.
MMMCCCXLVIII.                       //recognition process: extraction of features, testing neural net
MMMCCCXLIX.                 public void recognize() throws ClassNotFoundException, SQLException, IOException{
MMMCCCL.                            testNeuralNet();
MMMCCCLI.                   sortSpecies();
MMMCCCLII.                          displayMatch();
MMMCCCLIII.                 }
MMMCCCLIV.
MMMCCCLV.                   //reduce list of matches
MMMCCCLVI.                  public void reduce() throws ClassNotFoundException, SQLException{
MMMCCCLVII.                         SQL s = new SQL(url, user, pass);
MMMCCCLVIII.                        s.connect();
MMMCCCLIX.                          ArrayList<OutputSpecies> reduce = new ArrayList<OutputSpecies>();
MMMCCCLX.                   for(OutputSpecies spec : os){
MMMCCCLXI.                              Species p = s.queryByID(spec.getID());
MMMCCCLXII.                             boolean match = true;
MMMCCCLXIII.                            //check each field
MMMCCCLXIV.                             if(!(tsType.getText().equals("") || tsType.getText().equals(null))){
MMMCCCLXV.                              if(!p.getType().toLowerCase().equals(tsType.getText().toLowerCase())) match
        = false;
MMMCCCLXVI.                             }
MMMCCCLXVII.                                    if(!(tsArr.getText().equals("") || tsArr.getText().equals(null))){
MMMCCCLXVIII.                                           if(!
        p.getArrangement().toLowerCase().equals(tsArr.getText().toLowerCase())) match = false;
MMMCCCLXIX.                             }
MMMCCCLXX.                              if(!(tsBlade.getText().equals("") || tsBlade.getText().equals(null))){
MMMCCCLXXI.                                     if(!
        p.getBlade().toLowerCase().equals(tsBlade.getText().toLowerCase())) match = false;
MMMCCCLXXII.                                    }
MMMCCCLXXIII.                                   if(!(tsMargin.getText().equals("") ||
        tsMargin.getText().equals(null))){
MMMCCCLXXIV.                                            if(!
        p.getMargin().toLowerCase().equals(tsMargin.getText().toLowerCase())) match = false;
MMMCCCLXXV.                                     }
MMMCCCLXXVI.                                    if(!(tsVena.getText().equals("") || tsVena.getText().equals(null))){
MMMCCCLXXVII.                                           if(!
        p.getVenation().toLowerCase().equals(tsVena.getText().toLowerCase())) match = false;
MMMCCCLXXVIII.                                  }
MMMCCCLXXIX.                                    if(!match )reduce.add(spec);
MMMCCCLXXX.                             }
```

```
MMMCCCLXXXI.                        os.removeAll(reduce);
MMMCCCLXXXII.                       emptyTable();
MMMCCCLXXXIII.                      displayMatch();
MMMCCCLXXXIV.                       s.closeConnection();
MMMCCCLXXXV.                }
MMMCCCLXXXVI.
MMMCCCLXXXVII.              //main
MMMCCCLXXXVIII.             public static void main(String[] args) throws ClassNotFoundException, SQLException{
MMMCCCLXXXIX.                       try{
MMMCCCXC.                           @SuppressWarnings("resource")
MMMCCCXCI.                          BufferedReader br = new BufferedReader(new FileReader("file/SQL.txt"));
MMMCCCXCII.                 String line = br.readLine();
MMMCCCXCIII.                StringTokenizer tokens = new StringTokenizer(line,";");
MMMCCCXCIV.                 url = tokens.nextToken();
MMMCCCXCV.                  user = tokens.nextToken();
MMMCCCXCVI.                     pass = tokens.nextToken();
MMMCCCXCVII.                  } catch(IOException io){
MMMCCCXCVIII.                       JOptionPane.showMessageDialog(null,"File containing connection details does
        not exist.");
MMMCCCXCIX.                   }
MMMCD.
MMMCDI.                     if(!url.equals("") || !user.equals("") || !pass.equals("")){
MMMCDII.                            UserGUI u = new UserGUI("jdbc:mysql://"+url,user,pass);
MMMCDIII.                           u.createFrame();
MMMCDIV.                    }else{
MMMCDV.                             System.out.println("Cannot perform operation");
MMMCDVI.                    }
MMMCDVII.           }
MMMCDVIII.      }
MMMCDIX.
MMMCDX.
MMMCDXI.
MMMCDXII.
MMMCDXIII.
MMMCDXIV.
MMMCDXV.
MMMCDXVI.
MMMCDXVII.
MMMCDXVIII.
MMMCDXIX.
MMMCDXX.
MMMCDXXI.
MMMCDXXII.
MMMCDXXIII.
MMMCDXXIV.
MMMCDXXV.
MMMCDXXVI.
MMMCDXXVII.
MMMCDXXVIII.
MMMCDXXIX.
MMMCDXXX.
MMMCDXXXI.
MMMCDXXXII.
MMMCDXXXIII.
MMMCDXXXIV.
MMMCDXXXV.
```

MMMCDXXXVI.
MMMCDXXXVII.
MMMCDXXXVIII.
MMMCDXXXIX.
MMMCDXL.
MMMCDXLI.
MMMCDXLII.
MMMCDXLIII.
MMMCDXLIV.
MMMCDXLV.
MMMCDXLVI.
MMMCDXLVII.
MMMCDXLVIII.
MMMCDXLIX.
MMMCDL.
MMMCDLI.
MMMCDLII.
MMMCDLIII.
MMMCDLIV.

MMMCDLV. Acknowledgement

MMMCDLVI.

MMMCDLVII. Four years of hardship and neverending struggles, my journey as an undergraduate student is about to end as I now deem my university my alma mater. Throughout these years, UP has made me question my own competency and pushed me to my limits. I went out of my comfort zone, proved my worth and disproved false accusations some threw upon me. There was not one semester I was sure of. I've always had worries that anytime soon, I'd have to accept the fact that I'll be extending for at least another semester. *Oble* would probably smirk at me.

MMMCDLVIII. I've always thought I'm taking the path that's not for me; that I should be changing direction. But I was too patient so I stayed. Sleepless nights were proof of hard work, dedication and determination. Every semester gets worse and worse that you'd just want to sleep all day and do nothing. Grades do matter. They tell us who excels in which field; they determine where we should venture or have ventured. But what matters most is what we've sacrificed to get to where we are. There are people who went against my way but I went on and continued to struggle. They are people who taught me lessons and made me realize my imperfections. They

are worth my thanking. Moreover, there are some who deserve more than just thanking. They are the ones who believed in the path I chose; ones whose trust I always have.

MMMCDLIX. I would like to acknowledge my adviser, Prof. Solano, for being supportive and accommodating. I'm very lucky to have him as an adviser. He assisted me and gave me encouraging words. I also acknowledge my parents' efforts to send me to school and ignore me whenever I say I'll soon fail a subject every semester. They provided me with comfort food whenever I feel like I want to jump out of the window because of stress frustration.

MMMCDLX. I'd also like to acknowledge the help of my blockmates, especially Loann and Zach, who willingly offered their notes to be photocopied. At last, exams are over. Bluebooks will surely be missed. I'd also like to thank Althom for helping me out with some editing and Phoebe for assisting me in data gathering.

MMMCDLXI. And to these very special people I love, Jenn, Vennedy, Nico and my mom who sincerely encouraged me with my thesis despite my worries of not being able to finish on time, I'd forever be grateful I've met them. They motivated me to do well and to never give up. They truly are an inspiration.

MMMCDLXII. My mom never believed in me whenever I tell her I'd surely fail a subject and went on with what she's doing. She heard a lot of rants from me but was very supportive.

MMMCDLXIII. Above all, I would like to thank the Lord for staying with me all throughout. He gave me wisdom and courage. All these would mean nothing if not for Him. May He continue to guide me even after I close a chapter in life and start a new one.

MMMCDLXIV. Lastly, salamat *Oble*, as of now, we're over. 'Til next time. □

MMMCDLXV.
MMMCDLXVI.
MMMCDLXVII.
MMMCDLXVIII.
MMMCDLXIX.