CHAPTER 7

DESIGN OF LEAF IMAGE CLASSIFICATION MODELS FOR PLANT IDENTIFICATION

Image classification is an area in image processing where the primary goal is to separate a set of images according to their visual content into one of a number of predefined categories. It is the problem of finding a mapping from images to a set of classes, not necessarily object categories. Each class is represented by a set of features (feature vector) and the algorithm that maps these feature vectors to a class uses machine learning techniques. The ability to perform multi-class image classification as an automatic task using computers is increasingly becoming important. This is due to the huge volume of image data available, which are proving to be difficult for manual analysis. The difficulty arises due to lack of human experts, time complexity and poor quality images. The current market need is to have techniques which can classify images with minimum intervention from the users in an efficient and effective manner.

The last phase of the study is the task of identifying the plant to which the input leaf belongs and is very vital for botanical field. This is the most time consuming part of CAP-LR system, as the algorithm mainly revolves round an iterative recognition procedure that matches the features extracted from the input image with feature vectors representing the leaf images of the pre-built database. Classification, a task of machine learning algorithms, was used for this purpose.

It is very important for botanical industry to develop plant classification systems to map a leaf image to a plant category. In literature, statistical and machine learning techniques have been extensively studied for this purpose. Recent studies focusing on hybrid models that combine different machine learning techniques have also shown promising results. There are various types of combination methods to develop hybrid models. It is not known that which

hybrid machine learning model can perform the best, but, in general, have been proved to improve the classification/prediction performance over single learning approaches. Most of these hybrid models combine two or more predictive or descriptive algorithms, in order to improve the performance of identification and recognition. For example, a hybrid model can be built by combining the merits of Support Vector Machine (supervised) and K-Nearest Neighbour (unsupervised) method (Zhang, 2006).

In this study, another unique way of combining machine learning algorithms is proposed. Here, a two-level procedure that combines two machine learning algorithms is designed. The first level classifier is used to improve the performance of the second level classifier. The first classifier is used to preprocess the training data, and a second classifier learns from this preprocessed data and performs the identification and recognition tasks. Preprocessing step identifies failed classification and removes them from the training set to form a refined feature set. Thus, the second step uses only the correctly classified result to train the second classifier. This algorithm has the advantage of increasing the accuracy, reducing the error rate and reducing the time complexity of the classifiers. This 2 level classifier is termed as CL-CL (Classifier-Classifier) classifier in this study. This chapter begins with an introduction to machine learning algorithm and image classification, followed by the description of the steps used during the design of CL-CL models.

7.1. MACHINE LEARNING

Machine learning is the process of automating the development of some part of a system which performs efficient task. The parameters to an algorithm or process can be learnt adaptively over a period of time. The overall structure of a machine learning approach to a problem involves three steps (Russell and Norvig, 2003):

(i) The generation of some representation of a solution to the problem.
(ii) The evaluation of the generated solution.

(iii) If the evaluated solution is not good enough, the solution is iterated (i.e. almost always improved) and the machine learning process goes to step 2.

7.1.1. Types of Generation and Representation

Given the selection of some representation of a solution to the problem, the initial generation is usually random but constrained by some parameters. For example, in a neural network the structure is fixed and the weight associated with each link is generated according to some algorithm which ensures that the initially generated solution will almost certainly not be the same from time-to-time. However, there are a wide variety of possible representations, including Feed-forward neural networks, Support vector machines, Simulated annealing, Genetic algorithms, Bayesian networks and Decision trees.

7.1.2. Types of Iteration

There are broadly three ways in which the iteration from one solution to the next can be performed. This iteration is how to search for a good solution is carried out. Each of the three types of iteration will be summarized briefly in this section.

(i) Unsupervised - Unsupervised learning is normally used to locate patterns in the input data. No information is given to the system, which finds the patterns as to the correctness or incorrectness of the patterns. The patterns it finds may therefore be arbitrary or they may actually be representative of some real underlying process which caused them to appear gives for details on unsupervised classification problem.

(ii) Reinforcement - Reinforcement in terms of the quantity of information given to the system regarding the correctness of its output. Reinforcement learning is intermediary between supervised

and unsupervised learning. When reinforcement learning is used some information is given to the system at some time regarding the correctness of a prediction it made. This information ranges in precision from a categorization of a response as ”right“ or ”wrong“ to a precise amount of error, expressed numerically. At the latter degree of precision it differs from supervised learning only in the way the information is presented.

(iii) Supervised - When supervised learning is used the precise, correct output which should have been given for any particular training input is known to the system and used by the system to adjust the answer it will give to other training samples.

This study uses the supervised machine learning algorithms for classifying the leaf images for plant identification.

7.1.3. Types of Evaluation

The performance of the classifiers is determined using three performance parameters, namely, accuracy, speed and error rate.

7.2. IMAGE CLASSIFICATION

Assigning images to pre-defined categories by analyzing the contents is defined as ‘Image classification or ‘Image categorization’. The process of image classification allows users to find desired information faster by searching only the relevant categories and not the whole information space. Image classification normally involves the processing of two main tasks:-

(i) Feature extraction task – Extracts image features and forms a feature vector and

(ii) Classification task – Uses the extracted features to discriminate the classes.

Three paradigms can be identified during the classification and are listed below in (Figure 7.1)

(i) Binary case

(ii) Multi-class case

(iii) Multi-label case

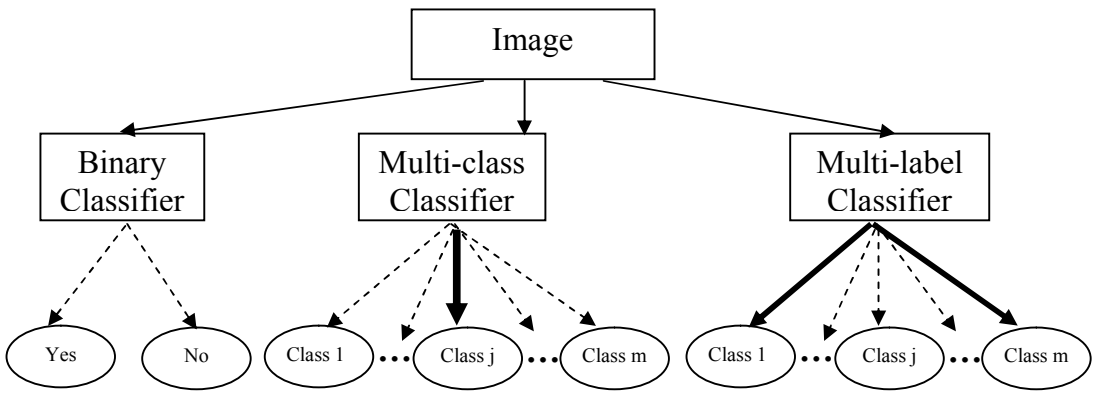


Figure 7.1: Paradigms in Image Classification

The binary case classification classifies images into exactly two predefined classes. Here, a sample image belongs exactly to one of the two given classes. The classifier has to determine to which of the two sets the new image goes (Mehta *et al.*, 2008). In mutli-class case, an image belongs exactly to just one class of a set of 'm' classes (Foody and Mathur, 2004; Joshi *et al.,* 2009). Finally, in the multi-label case, an image may belong to several classes at the same time, that is, classes may overlap (Li *et al.*, 2004).

In binary classification a classifier is trained, by means of supervised algorithms, to assign a sample document to one of the two possible sets. These two sets are usually referred to as belonging samples (positive) and not belonging samples (negative) respectively. This method is otherwise termed as the one-against all approach or one-against one approach. Several algorithms exist for this type of classification. They are Linear Regression, Support Vector

Machines (SVM) (Joachims, 1998), Naïve Bayes and LVQ (Martin-Valdivia *et al.*, 2003). The binary case has been a base case from which the two more cases, namely, multi-class and multi-label, are built.

In multi-class and multi-label cases, the traditional approach consists of training a binary classifier for every class and then whenever the binary base case returns a measure of confidence on the classification, assigning either the top ranked one (multi-class assignment) or a given number of the top ranked ones (multi-label assignment). More details about these three paradigms can be found in (Allwein *et al.*, 2000). The proposed CL-CL models for leaf recognition and plant identification defines multi-class classifiers.

In machine learning, multiclass or multi-label classification is the special case within statistical classification of assigning one of several class labels to an input image. Unlike the well-understood problem of binary classification, the multiclass one is a more complex and less researched problem. Multiclass classification is often carried out by serially applying binary classification. This can be accomplished by various strategies, including "One versus All" (OvA), "All versus All" (AvA) or more sophisticated information theoretic approaches. The OvA approach relies on the existence of a single computationally simple criterion which separates one class from the rest and makes use of standard binary classifiers. The AvA approach assumes the existence of simple separators between each pair of classes (Becker *et al.*, 2003).

In the past, multi-label classification was mainly motivated by the tasks of text categorization and medical diagnosis. Today, multi-label classification methods are seeing increased use in applications, like face recognition, optical character recognition, disease identification and text document classification. In this research, the multi-class classification is used to recognize the leaves, which in turn is used to identify the plant species to which it belongs to.

7.3. PROPOSED CL-CL MODEL

The CL-CL model is designed as a hybrid model that combines two machine learning classifiers to recognize the leaves for plant identification. It consists of three major steps as given below.

(i) Feature Extraction

(ii) Partitioning Feature Set

(iii) Design of CL-CL Models

7.3.1. Feature Extraction

From the feature extraction and selection task (Phase III) of CAP-LR, the five single feature vectors (geometric, color, texture, fractals and leaf specialized) and four fusion feature vectors (Geometric and Leaf Feature set (GLFS), Color and Leaf Feature set (CLFS), Texture and Leaf Feature set (TLFS) and Fractals and Leaf Feature set (FLFS) were obtained. The method of extraction, fusion and selection was described in the previous chapter.

7.3.2. Partitioning Method

These feature sets are partitioned using a hold-out method. Here, given a data set Z of size N x n, containing n-dimensional feature vectors describing N images, it is desirable to use as much as possible of the data to build the classifier (training) and also as much as possible unseen data to test its performance (testing). However, using the same data for training and testing, results in “over-training” of the classifier. In such a situation, the classifier perfectly learns the available data, but fails with unseen data. Thus, it becomes important to have a separate data set to train and test a classifier and make the best use of Z. Several methods used for separating the data into training and testing sets exist, like, Re-substitution (R-Method), Hold-Out method (H-Method), Bootstrap method and Cross-validation method. The proposed CL-CL models use the hold-out method for splitting the dataset into training and testing samples. The procedure used by the hold-out method is given in Figure (7.2).

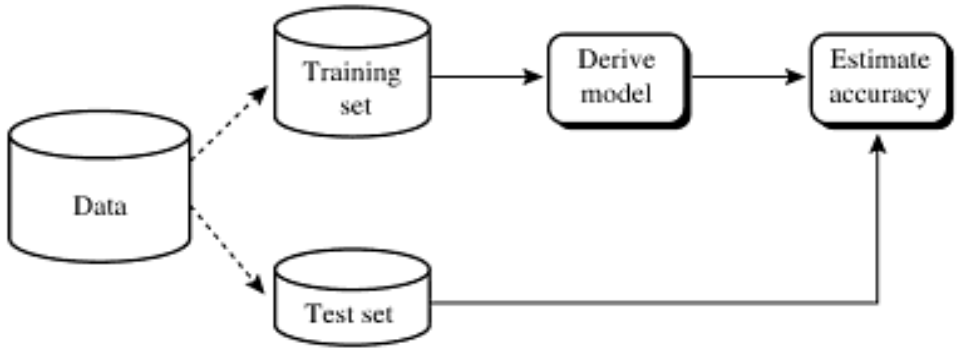


Figure 7.2 : Hold-out Method

The hold-out method randomly partitions the dataset into two independent sets, training and testing. Generally, 2/third of the data are allocated to be the training set and remaining is allocated as test set (Figure 7.3). The method is pessimistic because only a portion of the initial data is used to derive the model.

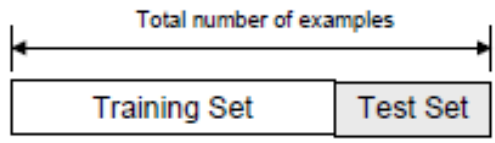


Figure 7.3 : Dataset Split into Training and Testing set

7.3.3. Design of CL-CL Models

After extracting the features, the CL-CL methods partitions the datasets into training and testing sets using the method explained in previous section. The design of the CL-CL models consists of two main steps, namely, design of level 1 classifier and design of level 2 classifier. The study proposes the use of three classifiers as follows.

(i) Back Propagation Neural Network (BPNN)
(ii) Support Vector Machine (SVM)
(iii) Wavelet Neural Network(WNN)

These three classifiers were selected because of popularity and proven record to produce accurate results. This section presents the description of each of these classifiers followed by the details of CL-CL models.

A) Classifiers Used

This section first presents a description of the three selected classifiers (BPNN, SVM, WNN) followed by the CL-CL model architecture.

- **Back Propagation Neural Networks**

The neural networks are quite famous to be well adapted for problems of classification, where the task of recognizing the leaf image for plant identification is addressed as a multi-class classification problem among various leaf species, in a given feature space. This approach consists of training a classification algorithm (neural networks) on a training set made up of labeled leaf images, to find a decision function in the considered feature space. Such decision function is then used at operation phase to label new images. In the case of leaf recognition and neural network, it is enough to choose a characteristic vector larger than that of the training sample to ensure the convergence. However, such practice will heavily weigh down the computation.

In the present research work, a supervised Back Propagation Neural Network is used to classify the leaf images into various predefined categories. The BPNN process is discussed in the following section.

- **Process of BPNN**

A BPNN is typically represented with the following notation:

$$R\text{-}S^1\text{-}S^2\text{-}S^3$$

where R is the number of inputs and S^i is the number of neurons at layer I Figure (7.4).

In Figure 7.4, the input vector to the network is shown by P_i. Transfer function is represented by f^i and b^i represents bias. Depending on the selected transfer function, the BPNN can be used to solve linear or nonlinear problems. In order to solve a nonlinear problem, a nonlinear transfer function should be selected.

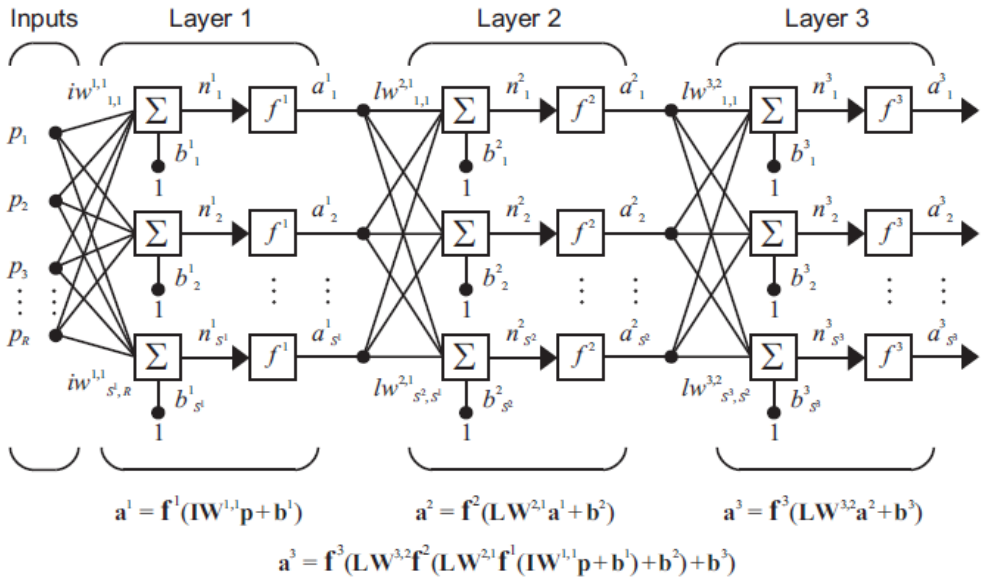


Figure 7.4 : Back Propagation Neural Network

The first step in BPNN is to transfer the input forward through the network. The output of one layer becomes the input to the following layer. Thus, the output of a network can be defined as

$$a^M = f^M (w^M * a^{(M-1)} + b^M) \qquad (7.1)$$

where M = 1,…….., R and R is the number of layers in the network.

In Figure 7.4, for the first layer, a(M-1) is a(0), which is the input of the network, and a(3) is the output of the last layer. The second step is to propagate the errors, or in other words, sensitivities backward through the network from the last layer all the way to the first layer. The BPNN algorithm is a generalization of the Least-Mean Square (LMS) algorithm, which uses the mean square error as a performance index. As each input is applied to the network, the output, a, is compared with the associated target value, t. The algorithm then adjusts the network parameters in order to minimize the mean square error. The error is defined as:

$$e = [t - a] \qquad (7.2)$$

and the expectation of mean square error is defined as

$$f(x) = E([e^T e]) = E[(t-a)^T (t-a)] \tag{7.3}$$

the expectation of the squared error at iteration k becomes

$$f(x) = E[t(k) - a(k))^T (t(k) - a(k))] \tag{7.4}$$

The algorithm adjusts weights and biases as follows

$$w_{i,j}^M(k+1) = w_{i,j}^M(k) - \alpha \frac{\partial f}{\partial w_{i,j}^M} \tag{7.5}$$

$$b_i^M(k+1) = b_i^M(k) - \alpha \frac{\partial f}{\partial b_i^M} \tag{7.6}$$

where α is the learning rate. This shows that the weights at any given iteration are equal to the weights at the previous iteration adjusted by some fraction, α, of the sensitivity of the error to that weight. In other words, the weights at each iteration are adjusted in a way that reduces the error at the previous iteration.

The selection of α is usually done by trial and error. Normally,too large of α often leads to divergence of the learning algorithm while too small α results in a slow learning process. Note that the above equations include partial derivatives. Since the error is an indirect function of the weights in the hidden layer, a chain rule is used to compute these partial derivatives. These partial derivatives are called sensitivities and defined as:

$$s_i^M = \frac{\partial f}{\partial n_i^M} \tag{7.7}$$

In the above equation, s_i is called the sensitivity of f to changes at layer M in the i^{th} element of the net input. The next step is to propagate the sensitivities backward

$$S^{M-1} = \dot{f}\, n^{m-1}(w^M)^T S^M \text{ , for } M = R, \ldots 1 \tag{7.8}$$

The final step is to adjust weight and biases with respect to these sensitivities, which directs to the following two equations.

$$w^M(k+1) = w^M(k) - \alpha S^M(a^{M-1})^T \quad (7.9)$$

$$b^M(k+1) = b^M(k) - \alpha S^M \quad (7.10)$$

Although the BPNN network is a very powerful technique, it usually requires a long training time. However, using some tricks and heuristics can often help to obtain an efficient BPNN implementation. In summary, the BPNN is a commonly used technique for solving nonlinear estimation problems, provided that a nonlinear transfer function is selected.

The BPNN algorithm is based on a Least Mean Square (LMS) algorithm that minimizes the squared error, where the chain rule is used to compute the derivatives of the error with respect to weights and biases in the hidden layers. The methodology in the BPNN network is a three-step process. First, the input is propagated forward through the network and then the sensitivities are computed. Second, starting from the last layer, these sensitivities are propagated backward through the network. Finally, weights and biases are updated using these sensitivities.

- **BPNN Architecture used in CAP-LR**

This research work is implemented using MATLAB to construct Back Propagation Neural Network (BPNN) that classifies the various leaf images for plant identification. The BPNN architecture used is given in Figure 7.5.

The BPNN is a multi-layer shape with input, hidden and output layer. In BPNN, the network structure influences the performance efficiency. Network structure is nothing but deciding the number of layers, number of nodes in each layer, the training algorithm and the network parameters. The proposed architecture consists of one input layer, one output layer and two hidden layers.

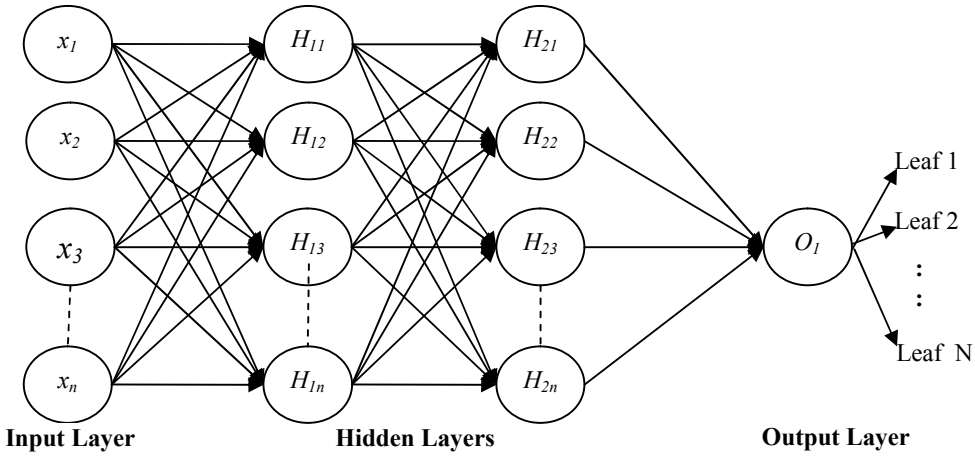


Figure 7.5 : BPNN architecture

Choosing the optimum number of hidden layers and nodes will improve the training efficiency. The layers and nodes selected are based on trial and error selection method. The input node depends on the features extracted from the image. The hidden layer consists of 20 neurons and output layer consist of single neuron. The activation function used in this research is tan sigmoid function. The learning algorithm employed in this study uses the Levenberg Marquardt (LM) method for minimizing the training error, which is defined as mean square difference between the desired output and the actual output. The target is fixed as one, when the leaf is recognized; otherwise, the target is fixed as zero. Network parameters are defined as follows:

net.trainParam.epochs = 500
net.trainParam.goal = 1e-5
net.trainParam.show = 50
net.trainParam.lr = 0.006
net.trainParam.mu = 0.6

The network is trained for ten times using the tan sigmoid activation function and the average result is considered. The model is tested by using the standard rule of 80/20, where 80% of the samples are used for training and 20% is used for testing. Here, the training process is considered to be successful when the MSE reaches the value 1e-005.

- **Support Vector Machine**

Support Vector Machines (SVMs) were introduced by Vapnik (1995) and have proved to be fast effective classifiers and works effectively with high dimensional datasets also (Eirinaki, 2009).

A Support Vector Machine (SVM) is a concept in computer science for a set of related supervised learning methods that analyze data and recognize patterns that are mainly used for classification and regression analysis. The standard SVM takes a set of input data and predicts, for each given input, which of two possible classes the input is a member of, which makes the SVM a non-probabilistic binary linear classifier(Burges,1998). Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

Although SVMs were originally designed as binary classifiers, approaches that address a multi-class problem as a single "all-together" optimization problem exist (Weston and Watkins, 1999). A multi-class classification task usually involves separating data into training and testing sets.Each instance in the training set contains one 'target value" (i.e. class labels) and several "attributes" (i.e. features). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.

Considering the binary classification case, let $((x_1, y_1) \ldots (x_n, y_n))$ be the training dataset where xi are the feature vectors that represent the observations and $y_i \in (-1, +1)$ be the two labels that each observation can be assigned to. From these observations, SVM builds an optimum hyperplane (a linear discriminant in the kernel transformed higher dimensional feature space) that

maximally separates the two classes by the widest margin by minimizing the objective function. An example for a linearly separable set of 2D-points which belong to one of two classes, find a separating straight line is shown in 7.6. In this example, there exist multiple straight lines that separate the data points into two groups. Deciding the optimal divider is an intuitive criterion.

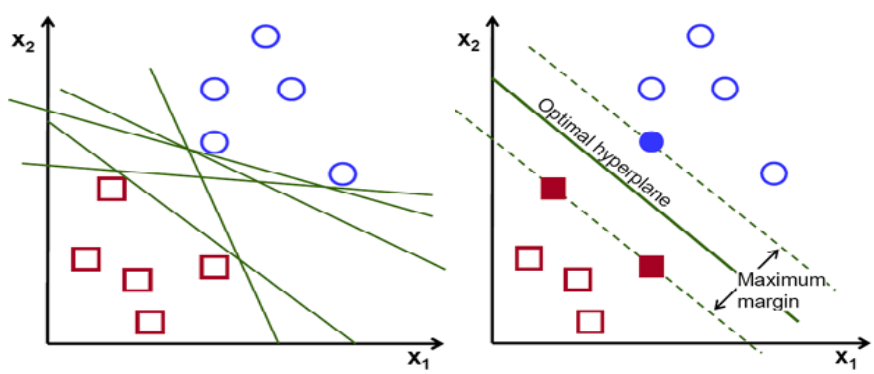
In general, a line is considered weak if it passes too close to the points because, it will be noise sensitive and will not generalize correctly. Therefore, the goal here is to find the line passing as far as possible from all points. Thus, the operation of the SVM algorithm (Chang et al., 2011) is based on finding the hyperplane that assigns the largest minimum distance to the training examples. Twice, this distance receives the important name of margin within SVM's theory. Therefore, the optimal separating hyperplane maximizes the margin of the training data. Example of an optimal hyperplane is shown in Figure 7.6.

- **Hyperplane Computation**

Let the hyperplane be defined as

$$f(x) = \beta_0 + \beta^T x \qquad (7.11)$$

where β is known as the weight factor and β_0 is the bias. The optimal hyperplane is represented in an infinite number of different ways by scaling of β and β_0.



(a) (b)

Figure 7.6 : Support Vector Machine Hyperplane

As a matter of convention, among all the possible representations of the hyperplane, the one chosen is

$$| \beta_0 + \beta^T x| = 1 \tag{7.12}$$

where x symbolizes the training examples closest to the hyperplane. In general, the training examples that are closest to the hyperplane are called support vectors and this representation is known as the canonical hyperplane. Now, the result of geometry that gives the distance between a point x and a hyperplane $\{\beta, \beta_0\}$ is estimated using Equation (7.13).

$$\text{distance} = \frac{| \beta_0 + \beta^T x |}{|| \beta ||} \tag{7.13}$$

In particular, for the canonical hyperplane, the numerator is equal to one and the distance to the support vectors is shown in Equation (7.14).

$$\text{distance}_{\text{support\_vectors}} = \frac{| \beta_0 + \beta^T x |}{|| \beta ||} = \frac{1}{|| \beta ||} \tag{7.14}$$

Let M denote the margin which is twice the distance to the closest examples (Equation 7.15).

$$M = \frac{2}{|| \beta ||} \tag{7.15}$$

Now, the problem of maximizing M is equivalent to the problem of minimizing a function $L(\beta)$ subject to some constraints. The constraint model is the requirement for the hyperplane to classify correctly all the training examples x_i. Formally, it can be defined as given in Equation (7.16).

$$\min_{\beta,\beta_0} L(\beta) = \frac{1}{2} || \beta ||^2 \qquad \text{subject to } y_i(\beta^T x_i + \beta_0) \geq 1 \ \forall i \tag{7.16}$$

where y_i represents each of the labels of the training examples.

SVM parameters are set as given below.

```
SVM_Params={'-b 0 -c 100 -d 3 -g 1 -t 3 '}
```

Here, -b indicates the probability estimates and takes the values 0 or 1 to indicate whether to train a SVC (Support Vector Classification or SVR (Support Vector Regression) model for probability estimates, 0 or 1).The study uses SVR. The next parameter is the cost which is set to 100, d is degree in kernel function which is assigned a value 3, while g (gamma in kernel function) is assigned a value of 1. The last parameter '-t' is used to identify the kernel function to be used. The kernel function, $K(x_i, xj) \equiv (x_i)^T \phi(x_j)$, can belong to any on e of the following four types available.

1. Linear Kernel : $K(x, x_j) = x_i^T x_j$
2. Polynomial Kernel : $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0.$
3. Radial Basis Function (RBF) : $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \gamma > 0.$
4. Sigmoid Kernel : $K(x_i , x_j) = \tanh(x_i^T x_j + r).$

Here, γ, r and d are kernel parameters. The study uses the radial basis function.

- **Wavelet Neural Network**

Introduced by Zhang and Benveniste (1992) WNNs have received tremendous attention from other researchers (Jiacong and Xingchun, 2008; Avci and Avci, 2008; Jie *et al.*, 2007), due to its great improvement over the weaknesses of neural networks. The usage of WNN has brought in the advantage of merging the structure of neural network into wavelets which extends the ability to approximate complicated patterns. The WNN can be considered as an expanded perceptron in which the neurons of the first layer are replaced by wavelet nodes (Zhang *et al.*, 1995; Delyon *et al.*, 1995). The wavelet nodes allow the detection of the transient, as well as the extraction and selection of a small number of meaningful features; the obtained features are then regarded as inputs to the subsequent neurons. The WNN used by CAP-LR system is designed as a three layer structure consisting of an input layer with n inputs, hidden layer with k wavelets and output layer with K outputs Figure (7.7).

The architecture of wavelet network is proximate with that of multi-layer perceptrons. The fundamental difference between the multi-layer perceptrons and the wavelet network is the nature of the activation functions (sigmoidal function and wavelet basis function respectively) used by the hidden neurons. Both input layer and output layer are completely connected to the hidden layer. From input layer to output layer, a feed-forward propagation algorithm is used. A back-forward propagation algorithm is used in the process of updating weights and parameters. In the basic back-propagation training algorithm the weights are moved in the direction of the negative gradient, which is one of the most suitable direction with rapid performance.

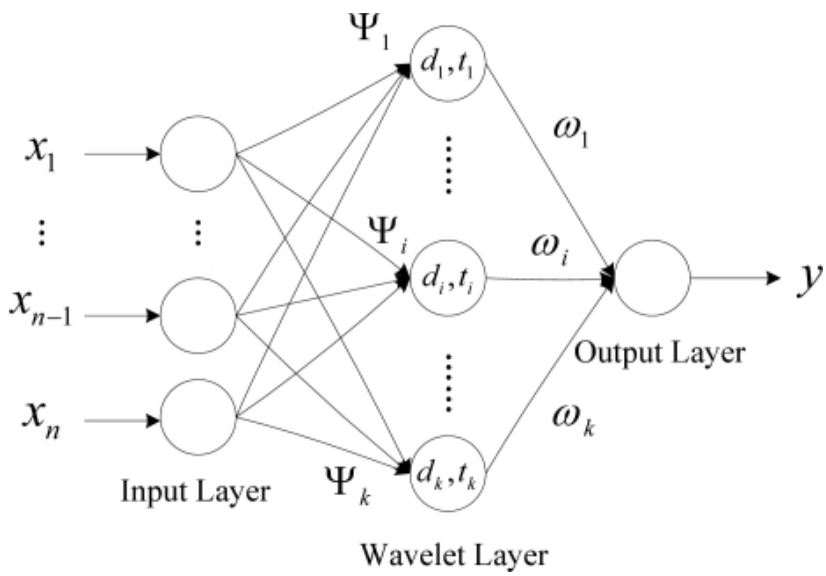


Figure 7. 7: Architecture of WNN

The hidden neurons have wavelet activation functions of different resolutions along with a weight connecting the hidden layer and output layer (ω_i). The weights of synapses of WNN are initialized by

(i) connecting input neurons and wavelet neurons

(ii) connecting wavelet neurons

(iii) connecting wavelet neurons and output neurons to small random values (–1 to +1).

The output of input layer is evaluated using wavelet activation function. ω_i is the weight connecting the hidden layer and output layer. For an input vector $x = [x_1, x_2, \ldots, x_n]$, the output of the ith wavelet layer neuron is described as follows:

$$\psi_k(x) = \sum_{i=1}^{n} \exp\left(-\left(\frac{x_i - d_k}{t_k}\right)^2\right) \cos\left(5\frac{x_i - d_k}{t_k}\right) \tag{7.17}$$

where x_i is the ith input vector and k is the number of wavelet nodes, d_k and t_k are translation and dilation parameters respectively. The output of the third layer is the weighted sum of $\psi_k(x)$ Equation (7.18).

$$y(x) = \sum_{m=1}^{k} \omega_m \psi_m(x) \tag{7.18}$$

Wavelet network training consists of minimizing the usual least-squares cost function:

$$E = \frac{1}{2} \sum_{j=1}^{s} \left(y_j - o_j\right)^2 \tag{7.19}$$

where s is the number of DCT coefficients and o_j is the optimal output of the jth input vector.

The learning algorithm of WNN consists of the following steps.

1. Initialize the values for D_i (Scaling vector) and t_i (translation vector)
2. Feed in the input vector X into WNN
3. Calculate the product of hidden layer using Equation (7.20)

$$\psi_j(\xi) = \psi(\|D_j(X-t_j)\|) \text{ where } j = 1, \ldots, m \tag{7.20}$$

4. Solve the weight matrix, $W = \psi^+\omega$, where ψ^+ is the pseudo inverse defined as $\psi^+ = (\psi^T\psi)^{-1}\psi^T$
5. Obtain the output value of WNN

6. Compare obtained output value with desired output value
7. Calculate cost using Equation (6.43)
8. Repeat steps (2) to (7) till stopping criterion is met

The learning of the WNN in the above procedure is by the method of solving the pseudo-inverse with fixed parameter initialization. Therefore, only the weight matrix W needs to be adjusted during the training of WNN, in order to map the underlying relationship between the input and output space. The Mean Square error of the difference between the network output and the desired output is calculated. This error is back propagated and the weight synapses of output and input neurons are adjusted. With the updated weights, error is calculated again. Iterations are carried out till the error is less than the tolerance.

In the WNN architecture, the number of layers used are input, wavelet and output layers with 13, 10 and 7 neurons respectively. The initial weights and biases are set randomly and the activation function used is Tangent sigmoid. The learning rule used is Levenberg-Marquardt back propagation algorithm with a learning rate of 0.75.

B) CL-CL Model

In order for the classification algorithms to learn to classify leaf images to a particular plant category, the training data must include both types of test cases. Let Tr and Te denote the training and testing partitions. A constant x is included to indicate the percentage of leaf feature data used for training and testing. The study analyzes different percentages of leaf feature data for this purpose and the experiments use x = 80%. That is, 80% of feature dataset is used to train level 1 and level 2 classifiers, while the rest 20% is used for testing. Now using Tr, level 1 classifier is trained.

Next, a new feature set is constructed using only those feature samples that produce positive results. This forms a new feature set, Tr'. Tr' has the

advantage of having a set of features that can improve the performance of the learning process of a classifier, as it includes only the positive results. The final step of CL-CL model, uses Tr' to identify the leaf images. As later will be proved through experiments, the refined training set improves the accuracy and reduces the error rate of the recognition process. The steps are consolidated in a diagrammatic form in Figure 7.8.

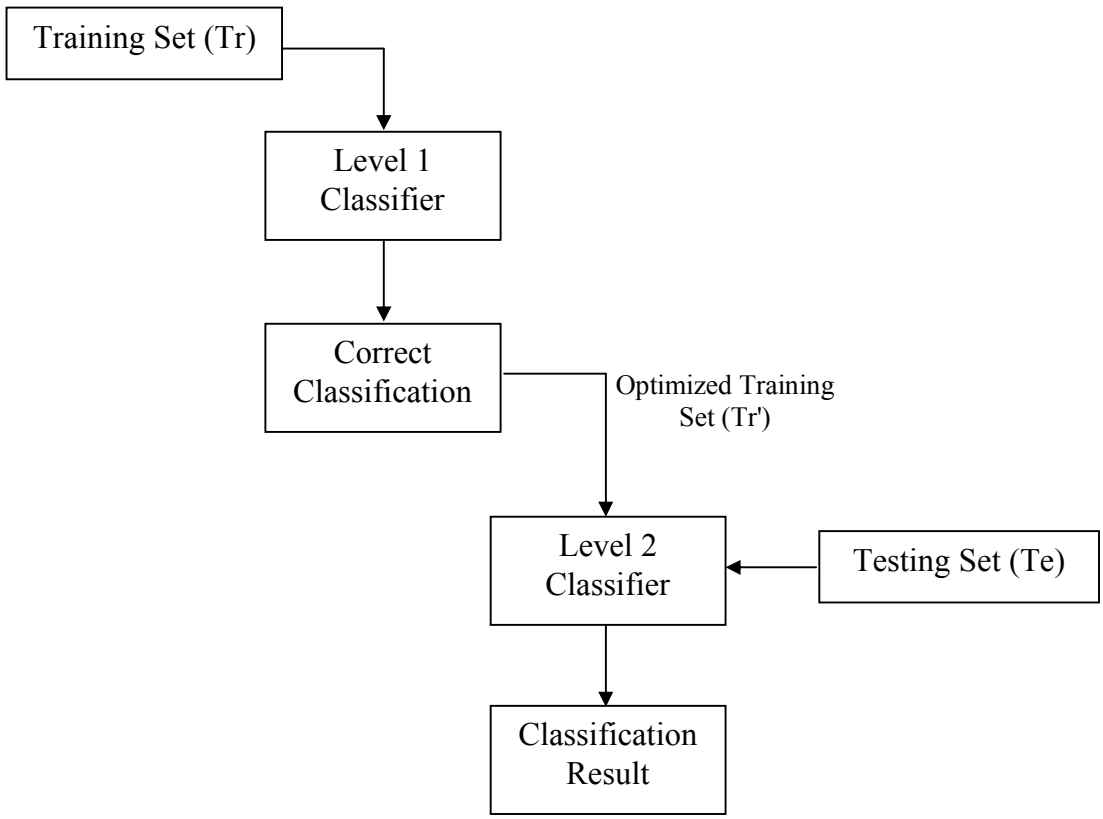


Figure 7.8 : Classification Process

The aim of the research is to find which of these classifiers is best suited for level 1 and which produce high performance when used for level 2 classification of CL-CL model. For this purpose, a combination of these algorithms was used to build nine CL-CL hybrid models Table (7.1). The nine models built are grouped into three categories based on the classifier used in level 1. The three categories are :

Group 1 : BPNN-Based CL-CL Models – Consisting of BPNN-BPNN, BPNN-SVM, BPNN-WNN models

Group 2 : SVM-Based CL-CL Models – Consisting of SVM-BPNN, SVM-SVM, SVM-WNN models

Group 3 : WNN-Based CL-CL Models – Consisting of WNN-BPNN, WNN-SVM, WNN -WNN models

TABLE 7.1

PROPOSED CL-CL TWO-LEVEL MODELS

I Level Classifier	**II Level Classifier**	**Code Used**
BPNN SVM WNN	BPNN	BPNN-BPNN
	BPNN	SVM- BPNN
	BPNN	WNN-BPNN
BPNN SVM WNN	SVM	BPNN -SVM
	SVM	SVM-SVM
	SVM	WNN-SVM
BPNN SVM WNN	WNN	BPNN-WNN
	WNN	SVMWNN
	WNN	WNN-WNN

7.4 CHAPTER SUMMARY

This chapter presented the methodology used during the design of the proposed 2-level machine learning classifier for leaf image classification. Three classifiers, namely, BPNN, SVM and WNN were used for this purpose, using which nine different CL-CL models were built. The effect of the various features (both single and fused) on recognition along with the performance of the 2-level classifier was analyzed using the two datasets, namely, real and standard, with several performance metrics were studied. The results of such experimentations are presented and discussed in the next chapter, **Results and Discussion.**