MOBILE LEAF CLASSIFICATION APPLICATION UTILIZING A CONVOLUTIONAL NEURAL NETWORK

A Thesis
by
TIMOTHY JOHN JASSMANN

Submitted to the Graduate School
at Appalachian State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

August 2015
Department of Computer Science

MOBILE LEAF CLASSIFICATION APPLICATION UTILIZING A CONVOLUTIONAL NEURAL NETWORK

A Thesis
by
TIMOTHY JOHN JASSMANN
AUGUST 2015

APPROVED BY:

Rahman Tashakkori
Chairperson, Thesis Committee

R. Mitchell Parry
Member, Thesis Committee

Dolores A. Parks
Member, Thesis Committee

James Wilkes
Chairperson, Department of Computer Science

Max C. Poole, Ph.D.
Dean, Cratis D. Williams School of Graduate Studies

Abstract

MOBILE LEAF CLASSIFICATION APPLICATION UTILIZING
A CONVOLUTIONAL NEURAL NETWORK

Timothy John Jassmann
B.S., Appalachian State University
M.S., Appalachian State University

Chairperson: Rahman Tashakkori

Plant classification is an important task in biological research. However, plant classification is a complex task that very few biologists are qualified experts to conduct. Therefore, an application to assist in this task would be extremely useful for biology students, researchers, and enthusiasts.

A significant amount of research has been done for the task of classifying plants based upon images of their leaves; however, all of that research has utilized images of single leaves on a white background for classification to allow easy extraction of shape features. This is not realistic for field work since a natural picture of a leaf will have a complex background.

This thesis applies a convolutional neural network to the problem in order to allow classification of images with natural backgrounds. A mobile application is built that can run this neural network on images taken by the device's camera. This tool can be used to assist in complex plant classification tasks anywhere as long as they have a mobile device with them.

Acknowledgments

I would like to first thank my advisor, Dr. Rahman Tashakkori, for his continuous support of my research and studies. I am thankful for his diligence, patience, and motivation to assist my research and writing of this thesis. I cannot imagine having completed this thesis without his assistance.

I would also like to thank my thesis committee, Dr. Mitchell Parry and Dr. Dee Parks for their assistance in producing this thesis.

I would also like to thank my family for their continued support throughout my studies. Their assistance has greatly helped me pursue my dreams and goals in life, and I could not have accomplished this without them.

Finally, I would like to thank all of my peers and friends for keeping me sane throughout this intense process. Their support through these past few years has been integral to me being able to focus and accomplish as much as I have.

Table of Contents

Chapter 1 Introduction

1.1 Background

Classification of plants can be a very complex task. Several studies have been done to utilize computer vision techniques to aid in the task of plant identification by classifying images of plant leaves. A discussion of several of these studies can be seen in Chapter 2. Kumar *et al.* developed a mobile application which can classify an image of a plant leaf [1]. However, that application, like the studies discussed on leaf classification, focuses on classifying images of leaves against a white background. This is not useful in practice, when the picture of the leaf has a natural background.

Convolutional Neural Networks have recently emerged as one of the top tools for computer vision due in part to their ability to perform classification tasks on images with natural backgrounds. Microsoft Research was able to obtain a 4.94% for the best five matches (top-5) test error on the ImageNet 2012 classification dataset [2]. This was the first result to surpass human-level performance, which had 5.1% top-5 test error.

By utilizing stacked convolutional layers, which convolve trained features over entire images, complex features are able to be recognized in images without any need for segmentation. Removing the need for segmentation allows much simpler classifications, since segmentation from natural images is an extremely complex process – especially in cases where objects overlap.

1.2 Overview of the Thesis

This thesis provides details on the design and implementation of a mobile application for the classification of pictures of leaves containing a natural background. The mobile application allows a user to either take a picture or upload an existing picture of a leaf and the mobile application displays a listing of potential classifications with percent likelihoods for each of the top three results and an image of each given classification.

The application computes these results utilizing a convolutional neural network. A library was written for convolutional neural networks that easily transfers onto a mobile device for computation. Several methodologies were used to reduce overfitting of the neural network, such as dropout [3, 4] and pixel alterations [4].

The results were observed by utilizing a dataset of pictures of leaves mostly taken at the Conservatory of Flowers in San Francisco, CA. Five-fold cross validation was performed on the dataset to measure the accuracy of the classifier's performance.

1.3 Thesis Organization

This research implements a mobile application for leaf classification utilizing convolutional neural networks. The application can classify a picture of a leaf and return a listing of potential classifications with percent likelihoods for the top three results. The application can either use an existing picture or take one to classify.

The work related to plant classification applications and convolutional neural networks is discussed in Chapter 2. Chapter 3 describes neural networks in detail and methods used to reduce overfitting with their use. Chapter 4 explains the methodology and implementation of the application designed for this thesis. Chapter 5 summarizes the results

observed from this research. Chapter 6 contains a discussion of the results as well as suggestions for future research.

Chapter 2 Related Work

2.1 Plant Classification

Plant classification is a complicated task, even for skilled biologists. Computer vision techniques can help automate this task to speed up the work of the biologists and be an educational tool for non-experts in the field. Several studies have been performed on automatic plant classification.

Qi and Yang utilized a support vector machine to extract the saw-tooth feature of edges of a leaf [5]. Li *et al.* utilized snakes technique and a cellular neural network to extract leaf veins and outlines [6]. Fu and Chi extracted the leaf's vein structure by combining an intensity histogram approach followed by neural network fine-tuning [7]. These three studies focused on extracting features to be used for classification of leaves of a plant.

Nam *et al*. created a shape-based leaf image retrieval system using the maximum power point algorithm along with a dynamic matching method [8]. The maximum power point algorithm reduced the points for shape representation, and the dynamic matching method efficiently calculated the result. Gu *et al.* utilized wavelet transform and Gaussian interpolation to extract the leaves' vein structures and contours of the leaves, and classified the leaves based on these features using a neural network [9]. Du *et al*. proposed the move median centers hypersphere classifier for classification of leaves based on several shape features such as aspect ratio, rectangularity, sphericity, and more [10]. These three studies utilized shape features of the leaves for classification of plants.

Heymans *et al.* utilized a neural network to distinguish between leaves of species in the Opuntia family [11]. Warren tested variation among images of types of Chrysanthemums [12]. There were 10 images per species of Chrysanthemum to be tested. Saito and Kaneko created a classifier for recognizing wild flowers using two images – one of the flower and one of the leaf [13]. Zhenjiang *et al.* created a rose-classification system utilizing object-oriented pattern recognition [14]. These four studies all attempted classification of similar species using extracted features of shape, color, and/or size of the leaf, petal, and/or flower of the plant to be classified.

Shrestha developed a leaf classifier that used K-means clustering to classify images into one of 38 species [15]. Aspect ratio, isoperimetric ratio, eccentricity, number of endpoints, and number of junction points were the features passed into the classifier. An accuracy of 89.65% was obtained on the test set. The network was trained and tested on leaf images from the Irvin Watson Carpenter, Jr. Herbarium located in the Department of Biology of Appalachian State University. The dataset contained only images of leaves on a white background.

In fact, all of the studies in this section required leaves be on a white background, or contained a segmentation step to get an image on a white background for classification. A white background was necessary for feature extraction and classification using those features. Background removal is a very complex task for natural images and any small error could significantly change the shape features used for classification, thus most of the algorithms focused on pre-segmented images.

2.2 Convolutional Neural Networks

Convolutional neural networks are an extremely useful tool for classification of objects in natural images without the need for background removal and feature extraction. Details of how convolutional neural networks work are described in Section 3.3.

Lee *et al.* were able to obtain 65.4% accuracy on the Caltech-101 object classification task using a convolutional neural network [16]. Le *et al.* trained a convolutional neural network on a set of 10 million images from the ImageNet database with 9,000 categories such as animals, furniture, musical instruments, and more to obtain an accuracy of 19.2% [17]. This result is better than the previous record of 16.7% and is significantly better than a random guess, which would result in less than 0.005% accuracy. Mehrota *et al.* implemented a classifier for Devanagari character recognition [18] that obtained 98.19% accuracy. Simard *et al.* trained a classifier for the MNIST handwriting recognition training set that performed with 99.6% accuracy [19].

Bell and Koren showed in the Netflix Prize Challenge that combining the results of several models improved accuracy in learning [20]. Hinton *et al.* built upon this concept to introduce the concept of dropout to convolutional neural networks [3]. By omitting a random set of neurons from each layer (usually around 50% of them), training the network is similar to training several networks with shared weights. They achieved an error rate on the ImageNet 1000 class dataset of 42.4%, significantly improving the previous record of 45.7%.

Krizhevsky *et al.* used data augmentation techniques to reduce overfitting [4]. Flipping images horizontally to increase the size of the dataset was one simple way they did this. Another way was randomly altering the RGB values of the image to reduce the effect of

intensity and color of the illumination of the image. These methods reduced their top-1 error rate by over 1%.

Glorot *et al.* showed that rectifier neurons generally outperform sigmoid and hyperbolic tangent neurons [21]. He *et al.* expanded the rectified linear unit to a parametric rectified linear unit, and improved upon layer initializations to achieve a 4.94% top-5 error on the ImageNet 2012 1000 class dataset. This was an improvement over the winner of the competition who achieved 6.66% top 5 error, and was the first to surpass human performance of classification on this dataset, which is 5.1% top-5 error.

Various research has been performed on leaf classification, however there hasn't been much done on classification of natural images of leaves. What has been done has required segmentation of the leaves from the natural background, which lacks consistency. This research aims to apply the convolutional neural network technique to create a mobile application that can classify pictures of leaves with their natural background taken by the device.

Chapter 3 Convolutional Neural Networks

3.1 Introduction

Neural networks consist of multiple layers of neurons connected together. The first set of neurons, known as the input layer, contains one neuron for each data point with the value for that data point as the activation value for that neuron. Neurons in deeper layers have their activations calculated using (3.2),

$$h(x) = f\left(\sum_{i=0}^{n} \theta_i x_i + b\right) \tag{3.2}$$

where *h(x)* is the value of the neuron, *n* is the number of connections, θ is a vector of weights, *x* is the input to the neuron, *b* is a bias term, and *f* is the activation function (activation functions are described in section 3.3.1). Figure 3.1 illustrates this relationship:

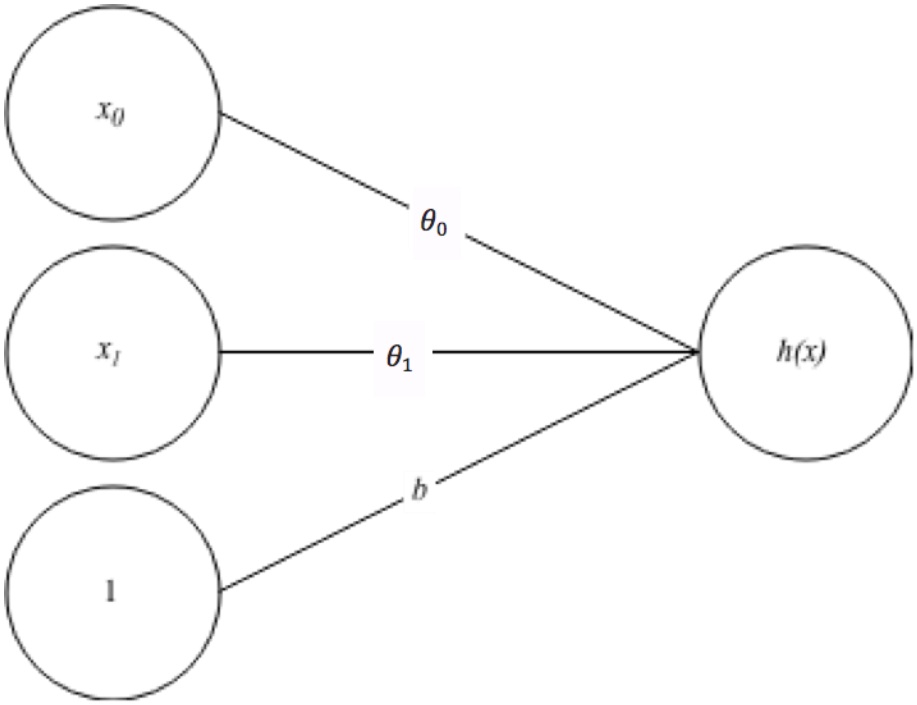

Figure 3.1: The connections between neurons

Neural networks can be used for classification tasks by feeding the data through the network using trained weights to an output layer consisting of a neuron for each classification. The calculated classification can be determined by taking the maximum activation value of the output neurons.

Convolutional neural networks utilize convolution and pooling layers, which significantly improve the power of neural networks for image processing. They are a useful tool for processing of images with natural backgrounds, and are therefore a great tool for processing natural images of leaves taken by a smartphone.

3.2 ZCA Whitening

Adjacent pixel values are highly correlated in images, causing redundancies in the input data. The biological eye, specifically the retina, performs decorrelation operations to reduce the redundancies in "pixels" picked up by the eye before transmitting to the brain [22]. Zero-phase component analysis (ZCA) whitening aims to produce a similar result by removing redundancies in the input data while preserving the spatial arrangement of the pixels in the image [23].

The formula for ZCA whitening is shown in (3.1),

$$x_{ZCAwhite} = UD^{-\frac{1}{2}}U^T x \tag{3.1}$$

where x is the mean-subtracted input data, *U* contains the eigenvectors of the covariance matrix of the input, and *D* is a matrix with the corresponding eigenvalues on the diagonal (and zeros everywhere else) [24]. The exponent applied to *D* is an element-wise operation only applied to the diagonal (the non-zero values) of the matrix. *U* and *D* are both square matrices with *n* rows and *n* columns where *n* is the number of features in *x*.

3.3 Training

Neural networks are trained using backpropagation of error along with an optimization algorithm such as gradient descent. For backpropagation, the error of each output node is calculated, and that error is propagated backwards throughout the network to all previous layers. The following sections have details on how this error is propagated backwards throughout the network for each type of layer.

The cost function of the network is shown in (3.3),

$$\varepsilon = \frac{1}{2m}\sum_{i=1}^{m}\left\|h_{\theta,b}\left(x^{(i)}\right) - y^{(i)}\right\|^2 \tag{3.3}$$

where ε is the error, m is the number of items in the train set, x is the train set, y is the set of labels corresponding to the items in the train set, and $h_{\theta,b}\left(x^{(i)}\right)$ is the output of the neural network for the input $x^{(i)}$.

Batch gradient descent is an optimization algorithm that performs a forward propagation of the training data, then uses backpropagation to calculate a gradient for each weight value in the network. These gradients, $\frac{\partial \varepsilon}{\partial \theta}$, can be used to update the weights according to (3.4)

$$\theta \leftarrow \theta - \alpha \frac{\partial \varepsilon}{\partial \theta} \tag{3.4}$$

where θ is the weights of the layer, and α is a small parameter used to adjust the step size of the function to ensure convergence.

While batch gradient descent is a good optimization function to use on small datasets, large datasets require too much memory to perform these calculations efficiently. For larger datasets, mini-batch gradient descent is often used. Mini-batch gradient descent splits the

data up into equal sized chunks and performs the exact same algorithm as batch gradient descent. This allows for training sets much larger than can fit into memory. However, it does add in a bias created by the order in which the data is presented to the algorithm, so it is common to randomly shuffle the training data prior to each epoch of training to minimize this issue [24].

Another way to optimize the training of a neural network is the classical momentum method [25, 26], which can be seen in (3.5),

$$v_{i+1} \leftarrow \mu v_i - \alpha \frac{\partial \mathcal{E}}{\partial \theta} \tag{3.5}$$

$$\theta_{i+1} \leftarrow \theta_i + v_{i+1}$$

where $\mu \in [0,1]$ is the momentum, and $v_0 = 0$. This is a two-stage update process, first updating the velocity v, and then using that to update the weight matrix θ. Classical momentum significantly increases the convergence of gradient descent, requiring $\sqrt{R}$ times fewer iterations to reach the same level of accuracy, where R is the condition number of the curvature at the minimum and μ is set to $\frac{\sqrt{R}-1}{\sqrt{R}+1}$ [26].

3.4 Activation Functions

The purpose of an activation function in a neural network is to introduce non-linearity into the network. Since the composition of two linear functions creates another linear function, having multiple layers to a neural network does not add any complexity unless a nonlinear activation function is applied.

Biological neurons have two states – firing or not firing. While this is a nonlinear activation function, it significantly reduces the complexity that can be modeled by the network by limiting the values of the neurons.

The sigmoid activation function is a close model to the biological neurons, but allows for a value anywhere in the range of zero to one. The sigmoid activation function can be seen in Figure 3.2 and (3.6).

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.6}$$

The gradient of the sigmoid function (used for training the network) can be calculated with (3.7).

$$\frac{\partial f(x)}{\partial x} = f(x)(1 - f(x)) \tag{3.7}$$

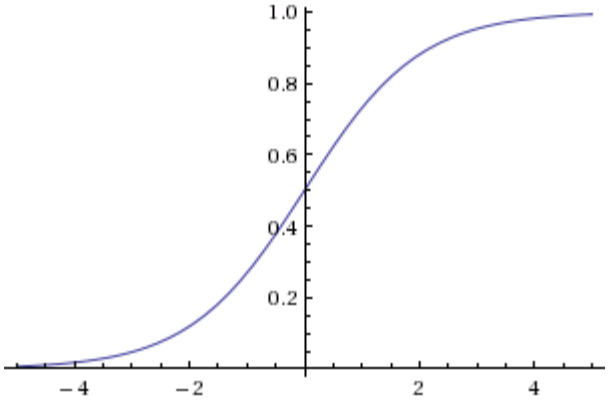

Figure 3.2: The sigmoid activation function

One of the problems with the sigmoid activation function is that as the magnitude of the input increases, the gradient approaches zero. It is also fairly computationally expensive. Since neurons are rarely in their maximum saturation, a rectifier can be used to approximate the activation function [27]. The Parametric Rectified Linear Unit (PReLU) [2] activation function, which addresses both of those issues, can be seen in Figure 3.3 and (3.8).

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ ax, & \text{if } x \leq 0 \end{cases} \tag{3.8}$$

In this function, a is another variable that will be trained with the network. The derivative of the PReLU function with respect to x can be seen in (3.9),

$$\frac{\partial f(x)}{\partial x} = \begin{cases} 1, & \text{if } x > 0 \\ a, & \text{if } x \leq 0 \end{cases} \tag{3.9}$$

and the gradient of the PReLU function with respect to a can be calculated with (3.10).

$$\frac{\partial f(x)}{\partial a} = \begin{cases} 0, & \text{if } x > 0 \\ x, & \text{if } x \leq 0 \end{cases} \tag{3.10}$$

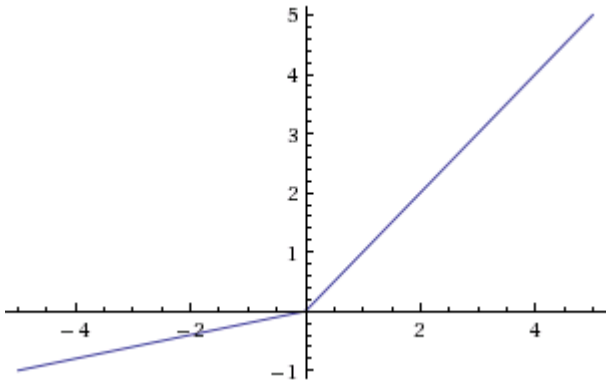

Figure 3.3: The PReLU activation function: a = 0.20

The gradient of a can be calculated using equation (3.11), where $\frac{\partial \varepsilon}{\partial f(x)}$ is the gradient propagated from the deeper layer, x is the input to the layer, and $f(x)$ is the output of the layer.

$$\frac{\partial \varepsilon}{\partial a} = \sum_{x} \frac{\partial \varepsilon}{\partial f(x)} \frac{\partial f(x)}{\partial a} \tag{3.11}$$

The activation function typically used for the output layer of a classifier is the softmax function. This function is similar to the sigmoid function, but gives the percent likelihood of each classification by ensuring the sum of all neurons totals one. The softmax function can be seen in (3.12).

$$f(x) = \frac{e^x}{\sum_{z \in N} e^z} \tag{3.12}$$

where *N* is the set of values of the neurons in the output layer before the activation function is applied, and $x \in N$.

3.5 Fully Connected Layers

The basic neural network layer is a fully connected layer. In this type of layer, each input neuron is connected to every output neuron. There is a separate weight associated with each connection, and a separate bias value for each output neuron.

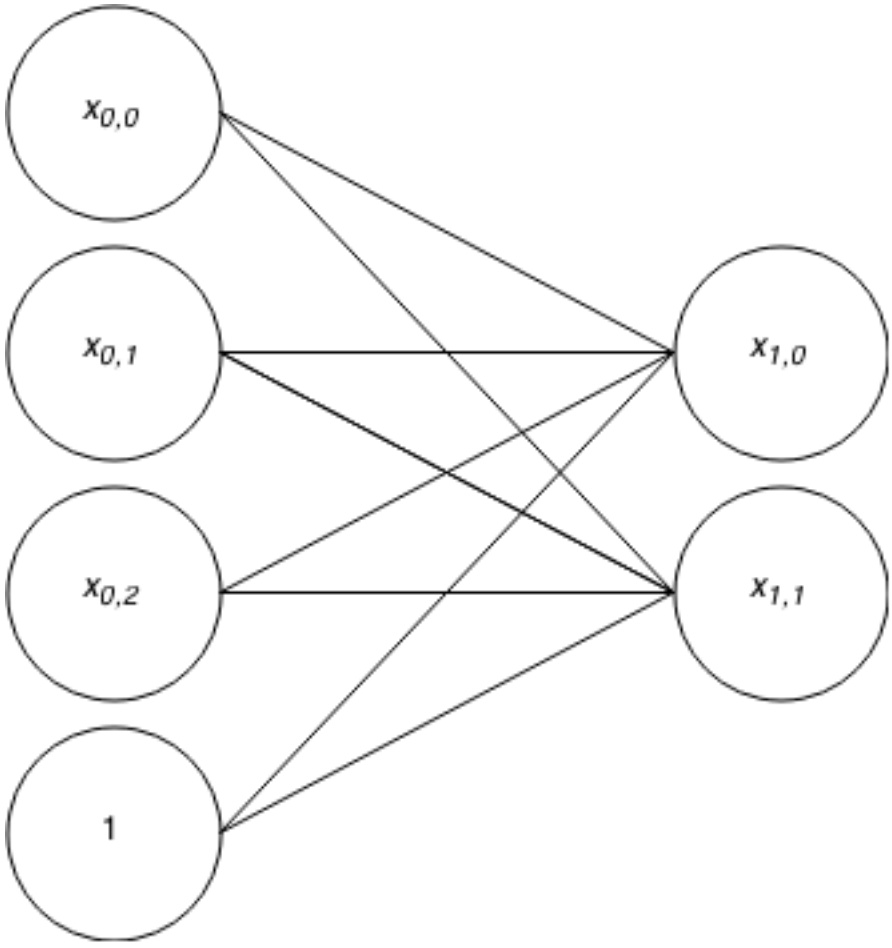

Figure 3.3: An example of a fully connected layer

Figure 3.3 shows a fully connected layer with 3 input neurons and 2 output neurons. Each connection has its own weight value, and the weights connected to the neuron labeled 1 are the bias terms.

The forward propagation of a fully connected layer is computed using (3.13),

$$z = f(x \times \theta + b) \tag{3.13}$$

where x is the input matrix containing one example per row, θ is the weight matrix, $\times$ is the matrix multiplication operator, b is the row vector containing the bias values, f is the activation function, and z is the matrix containing the activation of the output neurons.

The gradient of the weight matrix of the fully connected layer is shown in (3.14),

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{x^T \times \left(\frac{\partial \mathcal{E}}{\partial f(z)} \circ \frac{\partial f(z)}{\partial x} \right)}{m} \tag{3.14}$$

where $\frac{\partial \mathcal{E}}{\partial f(z)}$ is the error propagated from the deeper layer and $\frac{\partial f(z)}{\partial x}$ is the gradient of the activation of the output neurons of this layer, and m is the number of training examples. The gradient of the bias term is shown in (3.15).

$$\frac{\partial \mathcal{E}}{\partial b} = columnMeans\left(\frac{\partial \mathcal{E}}{\partial f(z)} \circ \frac{\partial f(z)}{\partial x} \right) \tag{3.15}$$

The propagation of the error through the layer can be seen in (3.16).

$$\frac{\partial \mathcal{E}}{\partial x} = \left(\frac{\partial \mathcal{E}}{\partial f(z)} \circ \frac{\partial f(z)}{\partial x} \right) \times \theta^T \tag{3.16}$$

In these equations, $\times$ is the matrix multiplication operator, and $\circ$ is the element-wise matrix multiplication operator.

3.6 The Softmax Classifier

The Softmax Classifier is used for the last layer of a neural network to output a likelihood value between zero and one for each possible classification. This layer is a fully connected layer that uses the softmax activation function and has no bias. The gradient of the weight matrix of the fully connected layer is shown in (3.17),

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{x^T \times (h(x \times \theta) - y)}{m} \tag{3.17}$$

where $\times$ is the matrix multiplication operation, θ is the weight matrix, x is the input to the layer, h is the softmax activation function, y is the expected output, and m is the number of training examples. The propagation of the error through the layer can be seen in (3.18).

$$\frac{\partial \mathcal{E}}{\partial x} = (h(x \times \theta) - y) \times \theta^T \tag{3.18}$$

3.7 Convolution Layers

A convolution layer consists of n features of size m-by-m that are convolved over an $h \times w \times c$ input, resulting in an $(h - m + 1) \times (w - m + 1) \times n$ output. Figure 3.4 shows a convolution of a 3×3 kernel over a 5×5 image.

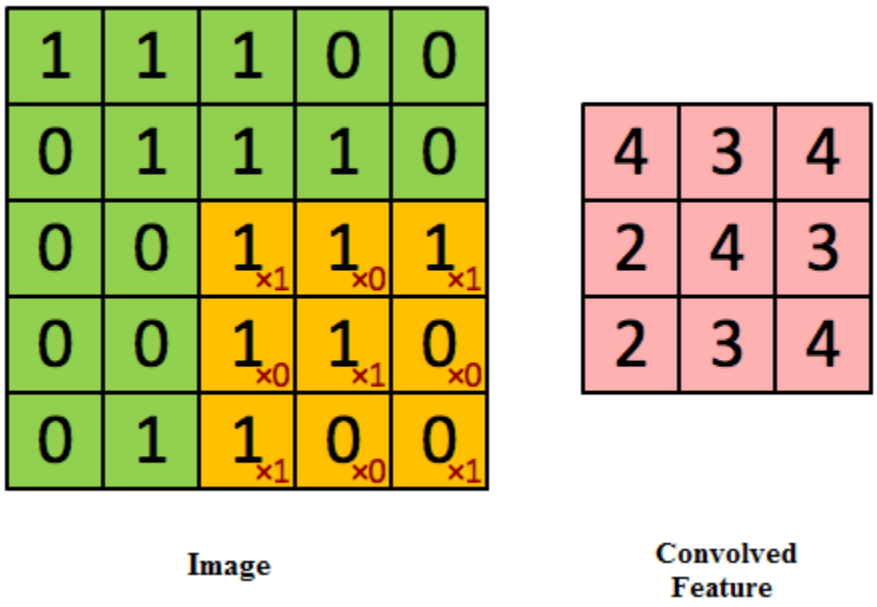

Figure 3.4: A valid convolution of 3×3 kernel over 5×5 image [24]

Similar to the fully connected layers, the result has a bias added to it (there is one bias value for each feature) and an activation function applied to it. The forward propagation of a convolution layer is computed using (3.19),

$$z_i = f\left(\left(\sum_{k=1}^{c} x_k * \theta_{i,}\right) + b_i\right) \tag{3.19}$$

where x_k is a training example, $\theta_{i,}$ is a feature matrix, b_i is the bias term, c is the number of channels in the input image, i is the feature number, f is the activation function, and z is the matrix containing the activation of the output neurons. The $*$ operator indicates a valid 2d convolution – a convolution in which the kernel fits entirely inside of the image it is being convolved over.

These convolution layers are feature detectors that map how well specific features fit in an image. Having multiple of these convolution layers in series allows for more complex features to be detected over the image.

The bias gradient is calculated the same way as the fully connected layers in convolution layers, but the weight gradients and the propagation of the error need to be calculated differently. The calculation of the weight gradients is shown in (3.20).

$$\frac{\partial \mathcal{E}}{\partial \theta_i} = \frac{x_i * \left(\frac{\partial \mathcal{E}}{\partial f(z)} \circ \frac{\partial f(z)}{\partial x}\right)}{m} \tag{3.20}$$

The calculation of the error propagated through the layer is shown in (3.21),

$$\frac{\partial \mathcal{E}}{\partial x_i} = \left(\frac{\partial \mathcal{E}}{\partial f(z)} \circ \frac{\partial f(z)}{\partial x}\right) \cdot* \theta_i' \tag{3.21}$$

where θ' is the weight matrix rotated 180° and $\cdot*$ indicates a full convolution – a convolution in which the input, *x*, has been padded with $m - 1$ zeros on each side, resulting in a $(h + m - 1) \times (w + m - 1)$ matrix.

3.8 Pooling Layers

A pooling layer takes an $h \times w \times n$ input and performs a pooling operation with a pooling size *m* to output an $\frac{h}{m} \times \frac{w}{m} \times n$ output. Common pooling operations are max and mean. Pooling layers reduce the size of the features in the network, which reduces calculation time of future layers, and makes the network less prone to over-fitting. It also adds a degree of translation invariance to the network.

In max pooling, error is propagated through the layer by upsampling the error to the size of the input and setting the argmax of each pooled region to the error for that region. In mean pooling, the error is upsampled to the size of the input by setting each value in the pooled region to the error divided by the number of neurons in the pooled region.

3.9 Addressing Overfitting

Convolutional neural networks end up having many features, so they are very prone to overfitting. Reducing overfitting while maintaining a complex representation of the data is important – especially when dealing with smaller data sets.

One way to reduce overfitting is to introduce a regularization, or weight decay, term into the cost function of the network. By adding the sum of the weight values to the cost function, the magnitude of the weights are pushed toward zero, which evens out the weight of each feature – reducing overfitting in the network. The updated cost function can be seen in (3.22) where *n* is the number of layers, *s* is the rows in the weight matrix of layer *l*, *t* is the columns of the weight matrix of layer *l*, and θ^l is the weight matrix of layer *l*.

$$\varepsilon = \frac{1}{2m}\sum_{i=1}^{m}\left\|h_{\theta,b}\left(x^{(i)}\right) - y^{(i)}\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{t_l}\theta_{ji}^{(l)^2} \tag{3.22}$$

This changes the update of the weight vector to (3.23).

$$v_{i+1} \leftarrow \mu v_i - \alpha\left(\frac{\partial \varepsilon}{\partial \theta} + \lambda\theta_i\right) \tag{3.23}$$

$$\theta_{i+1} \leftarrow \theta_i + v_{i+1}$$

Another way to reduce overfitting is to artificially increase the training size by augmenting input data to generate additional samples. Two beneficial forms of data augmentation are image reflections and pixel alterations [4]. Image reflections are simple transformations that give a different perspective on the same image, and therefore can be useful in learning different orientations of objects.

Pixel alteration is slightly more complex, but adds an invariance to intensity and color of illumination of the image. The formula for pixel alteration is (3.24) where *I* is an

individual RGB image pixel, p_i and λ_i are the ith eigenvector and eigenvalue of the 3 x 3 covariance matrix of the RGB pixel values, and a_i is a random variable from a normal distribution (uniform for all RGB pixels of a particular training image per epoch) [4].

$$\left[I_{xy}^{R}, I_{xy}^{G}, I_{xy}^{B}\right]^{T} += \ [p_1, p_2, p_3][a_1\lambda_1, a_2\lambda_2, a_3\lambda_3]^{T} \tag{3.24}$$

Overfitting can also be reduced by combining the predictions of different models [4, 20, 28]. However, training models with many features is very slow. Dropout [3, 4] emulates training thousands of models at a time, and doesn't run much slower than training a single network. Dropout works by randomly setting the activation of half of the output neurons in a layer to zero, removing them from the network for this training cycle. The error is not propagated back through these neurons either. Dropout causes each step of training the network to train a slightly different model of the network, but each model shares the same weights. This reduces co-adaptations of neurons in which neurons are only helpful in the context of specific other neurons. This significantly reduces overfitting and significantly improves test accuracy.

Chapter 4 Methodology

4.1 Introduction

A convolutional neural network approach to a mobile application for leaf classification is discussed in this section. Figure 4.1 shows the approach taken in this thesis. Section 4.2 provides details on the dataset gathered for this thesis. Section 4.3 discusses the process of loading and preprocessing of the dataset. Section 4.4 describes the neural network library written for this approach. Section 4.5 describes the structure of the neural networks trained for this thesis. Section 4.6 describes the training and testing of the neural networks. Section 4.7 describes the mobile application that is created for the purpose of classification in this research.

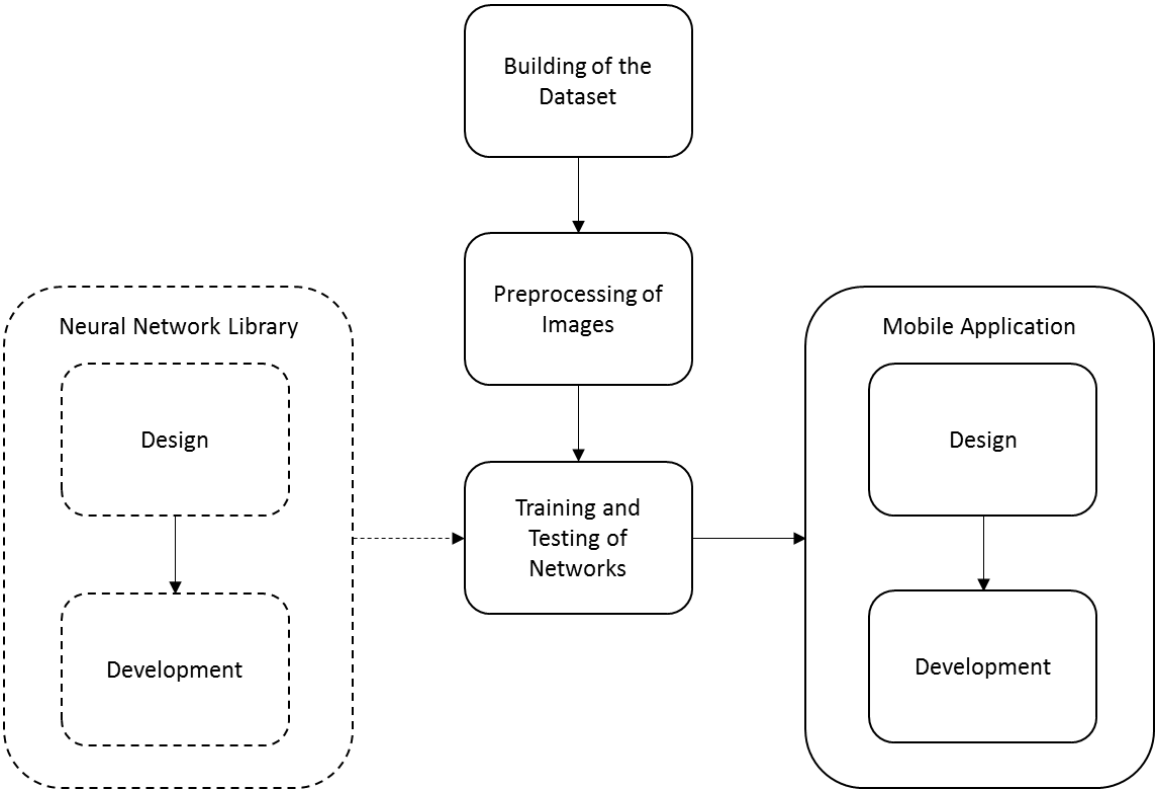

Figure 4.1: The leaf classification mobile application process

4.2 Building of the Dataset

The dataset was put together for this thesis by taking pictures of leaves using the camera on a Samsung Galaxy S4. Thirty pictures of the leaves for each of fifteen different species of plants were taken at a 4128x3096 resolution. Fourteen of the fifteen species of plants were taken at the Conservatory of Flowers in San Francisco, California. The last species of plant is an unidentified tree in Boone, North Carolina that was used for ensuring that the mobile application's camera feature worked well with new images. Figure 4.2 shows a single picture of each of the fifteen classes that are taken for the purpose of this research.

Figure 4.2: From left to right:
Row 1: Acalypha Hispida, Alternanthera Ficoidea, Arenga Hookeriana, Begonia Brevirimosa, unidentified tree from Boone
Row 2: Calathea Insignis, Calathea Warscewiczii, Coffee Arabica, Crescetia Cujete, Eugenia Uniflora
Row 3: Medinilla Magnifica, Ravenala Madagascariensis, Saraca Indica, Stanhopea Orchid, Vanda Orchid

4.3 Preprocessing of Images

The dataset for this thesis was organized into a folder structure with one folder for each species. The image loader went through each folder and generated the labels for the classes using this structure – all images in the same folder got the same label.

Each image was loaded into a BufferedImage and rotated if necessary to ensure that the height was the larger dimension using the imgscalr library's Scalr class. Next, the image was resized to a resolution of 80x60. The pixel data were then extracted from the BufferedImage and stored into three two-dimensional array of DoubleMatrix (from the JBLAS library) – an array of matrices for each image containing a matrix for each channel of input. The labels were stored in a single DoubleMatrix – one row per image.

The data were then normalized to have zero mean and unit variance. For every channel of each image, first the mean pixel value for the channel was subtracted from each pixel, then the pixel value was divided by the standard deviation of the pixels in that channel of that image. After normalization, ZCA Whitening was applied to the data.

4.4 Neural Network Library

As part of the research for this thesis, a neural network library was written in Java to allow for executing on an Android device, and training and testing on a desktop computer. The library allows for convolution layers, pooling layers, and fully connected layers with various options for cost functions and parameters. Figure 4.3 shows the organization of the library as well as the public methods of each class. Sections 4.4.1-4.4.7 describe the classes that were written for the purpose of this research.

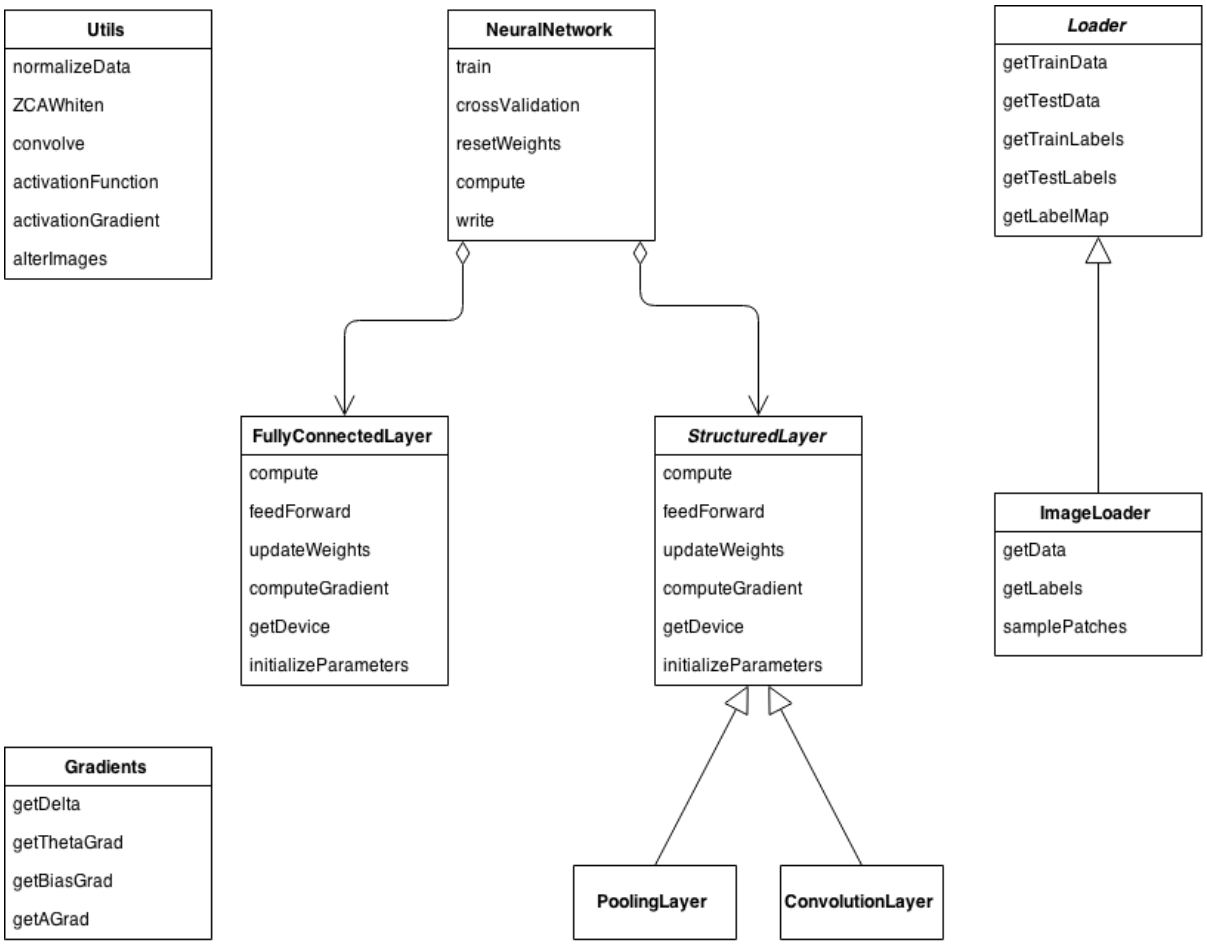

Figure 4.3: Organization of the neural network library

The desktop portion of the library uses the matrix library JBLAS for matrix operations. Since this library uses a platform specific compiled BLAS library that hasn't been ported to android, the android portion of the library uses the JAMA matrix library. The ConvolutionLayer, FullyConnectedLayer, NeuralNetwork, PoolingLayer, StructuredLayer, and Utils classes have also versions with fewer methods and that utilize the JAMA matrix library for computation on a mobile device.

4.4.1 The ConvolutionLayer Class

The ConvolutionLayer class represents a convolution layer of the neural network. There are public methods available to compute the output of the layer, perform a feedforward

pass of the layer, compute the gradients of the layer, initialize the weights of the layer, update the weights of the layer, and get the mobile convolution layer equivalent. The mobile ConvolutionLayer only has a method to compute the output of the layer.

The theta values for this layer are initialized using (4.1), where the size of the feature is *k*-by-*k*, *a* is the parameter for the PReLU activation function and *c* is the number of input channels [2]. The bias is initialized to all zeros. The value of *a* is initialized to 0.25 if PReLU is the activation function; otherwise it is initialized to 0.

$$\sqrt{\frac{2}{(1+a^2)*k^2*c}} \tag{4.1}$$

The convolution layer takes as input a two-dimensional array of matrices indexed by image and channel. Each matrix represents a single channel of an image. The output is a two-dimensional array of matrices indexed by image and feature. The activation function for the convolution layer is customizable with options for PReLU, ReLU, sigmoid, softmax, and none. Parameters for the number of features, size of features, weight decay, and dropout can also be set.

4.4.2 The FullyConnectedLayer Class

FullyConnectedLayer is the class that represents a fully connected layer of the network. There are public methods to compute the gradients of the layer, update the weights of the layer, compute the output of the layer, perform a feedforward pass of the layer, initialize the weights of the layer, and get the corresponding mobile fully connected layer. The mobile FullyConnectedLayer only has a method to compute the output of the layer.

The theta values for this layer are initialized using (4.2) [2] where *n* is the size of the input to the layer. The bias is initialized to all zeros. The value of *a* is initialized to 0.25 if PReLU is the activation function; otherwise it is initialized to 0.

$$\sqrt{\frac{2}{(1+a^2)*n}} \tag{4.2}$$

The input and output to the fully connected layer is a single matrix with each row containing the values for a separate image. The activation function for the fully connected layer is customizable with options for PReLU, ReLU, sigmoid, softmax, and none. Parameters for weight decay and dropout can also be set.

For a Softmax Classifier, a FullyConnectedLayer is used with the Softmax activation function. Since it behaves the same way except for activation, there is no need for a separate class for it.

4.4.3 The Gradients Class

The Gradients class is a data structure used for holding the gradients of a layer. It contains matrices for the theta gradient, bias gradient, the gradient of *a*, the gradient propagated through the layer, and the cost of the layer. There is no mobile class for Gradients since it is only used for training.

4.4.4 The Loader Class

Loader is an abstract class that defines a class that loads data for the network. It has abstract methods to get training data, get training labels, get test data, get test labels, and get a mapping from the numerical label to the corresponding text label. This is used by the NeuralNetwork class to get data for training the network or performing cross validation.

The library contains an implementation of this class called ImageLoader as well. ImageLoader has a constructor that loads data from a folder structured with each classification having its own folder to determine the label for each class. It additionally has methods to get all data and to get all labels.

4.4.5 The NeuralNetwork Class

The NeuralNetwork class encapsulates all of the layers of the network. It has methods to train the network using mini batch gradient descent, to compute the result of an input, perform cross validation of the network, reset the weights of the network, and write the network to a file. The mobile NeuralNetwork class builds the trained network using a file written by the desktop version. It then has a function to compute the result of the network on a given input matrix.

The neural network consists of an array of StructuredLayers and an array of FullyConnectedLayers. The network maintains the structure of the input while computing the StructuredLayers, then flattens the data before passing it to the FullyConnectedLayers. If used for classification, the last FullyConnectedLayers should be a softmax classifier (by utilizing the softmax activation function) to give results as a percent likelihood.

4.4.6 The PoolingLayer Class

The PoolingLayer class represents a pooling layer of the network. It has public methods for computing the output of the layer, performing a feedforward pass on the layer, expanding the gradient for backpropagation, and getting the equivalent device pooling layer. The layer can perform max or mean pooling. It additionally has methods to initialize parameters and update weights, which do nothing, in order to conform to the StructuredLayer

interface. The mobile pooling layer only has the method for computing the output of the layer.

4.4.7 The StructuredLayer Class

StructuredLayer is an abstract class for layers that maintain the structure of the image. This is used for polymorphism between convolution layers and pooling layers to allow them to be structured in any order when passed into the NeuralNetwork class. It contains abstract methods for computing the output, performing a feedforward pass, computing the gradients, updating the weights of the layer, initializing parameters, and getting the equivalent device structured layer.

4.4.8 The Utils Class

The Utils class contains various additional methods for the neural network. It contains public methods for ZCA whitening (Section 3.2), normalizing the data, the convolution operation, sampling patches from images, pixel alteration (Section 3.9), and activation functions and their gradients.

4.5 Training and Testing of the Neural Networks

This section discusses the training and testing of five different neural network configurations. Section 4.5.1 covers the design of the different neural networks, Section 4.5.2 discusses the training of the networks, and Section 4.5.3 discusses the testing of the neural networks.

4.5.1 Neural Network Design

For this thesis, five different neural network configurations are compared. Table 4-1 describes the configurations of each network. All of the networks trained on images that were scaled to a resolution of 80x60 pixels with a set weight decay of 0.00005. Dropout was

applied to two layers of each network. Additional dropout caused the networks to train too slowly.

Table 4-1: Neural Network Configurations

Network A	Network B	Network C	Network D	Network E
Convolution 16 3x3 Features Dropout 0.5 PReLU activation	Convolution 16 3x3 Features No dropout PReLU activation	Convolution 16 3x3 Features No dropout PReLU activation	Convolution 16 3x3 Features No dropout PReLU activation	Convolution 16 3x3 Features No dropout PReLU activation
Pooling 2x2 pooling Max pooling	Pooling 2x2 pooling Max pooling	Pooling 2x2 pooling Max pooling	Pooling 2x2 pooling Max pooling	Pooling 2x2 pooling Max pooling
Fully Connected 200 output neurons Dropout 0.5 PreLU activation	Fully Connected 200 output neurons Dropout 0.5 PreLU activation	Convolution 16 2x2 Features Dropout 0.5 PReLU activation	Convolution 16 2x2 Features No dropout PReLU activation	Convolution 16 2x2 Features No Dropout PReLU activation
Softmax Classifier 15 output neurons	Fully Connected 200 output neurons Dropout 0.5 PreLU activation	Pooling 2x2 pooling Max pooling	Pooling 2x2 pooling Max pooling	Pooling 2x2 pooling Max pooling
	Softmax Classifier 15 output neurons	Fully Connected 200 output neurons Dropout 0.5 PreLU activation	Fully Connected 200 output neurons Dropout 0.5 PreLU activation	Convolution 16 2x2 Features No dropout PReLU activation
		Softmax Classifier 15 output neurons	Fully Connected 200 output neurons Dropout 0.5 PreLU activation	Fully Connected 200 output neurons Dropout 0.5 PreLU activation
			Softmax Classifier 15 output neurons	Fully Connected 200 output neurons Dropout 0.5 PreLU activation
				Softmax Classifier 15 output neurons

4.5.2 Training of Neural Network

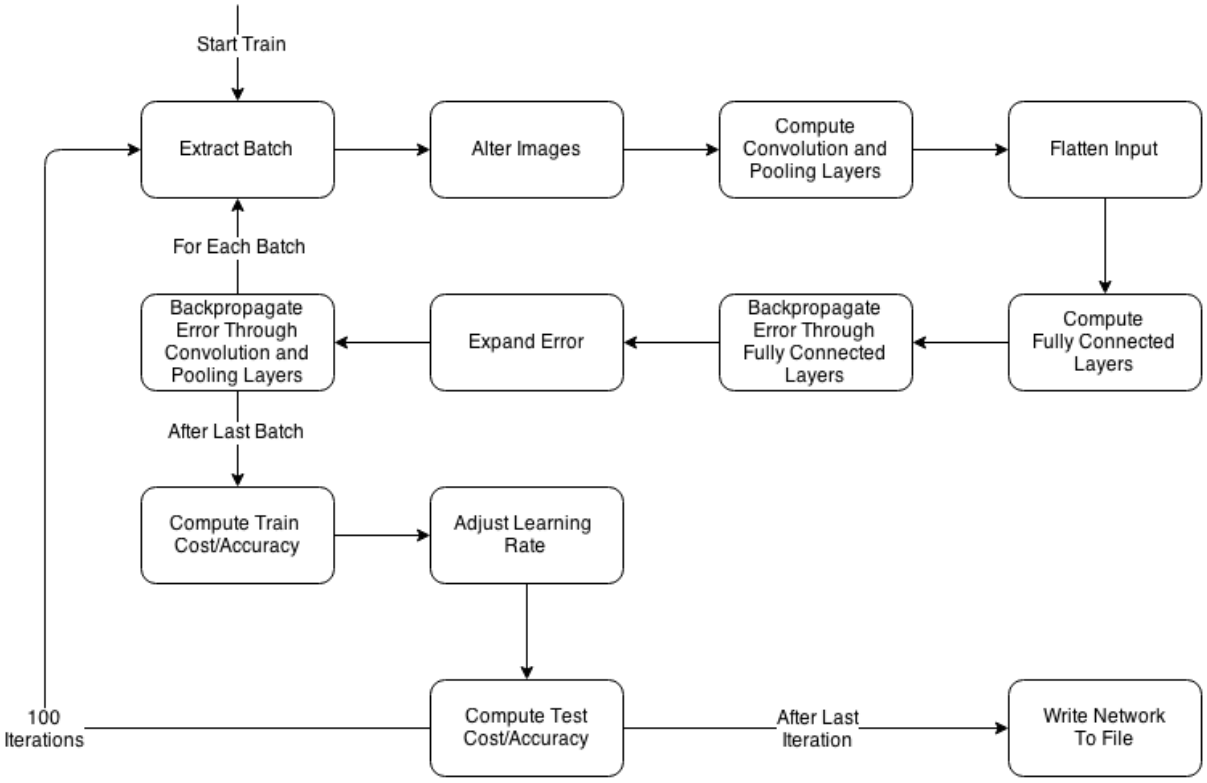

Figure 4.4: Training of neural networks

All networks are trained using minibatch gradient descent with a learning rate of 0.001. After each epoch, the train and test cost and accuracy are printed out to compare the performance of each network configuration as it trains. If the cost of the training set increased from the previous epoch, the learning rate is scaled by 0.75 to allow for fine-tuning of the weights. Each network is trained for 100 iterations.

During training, alterations (as described in Section 3.9) are made to images for each epoch to reduce overfitting. Pixel alterations are applied to each image to reduce the effect of the intensity and color of illumination caused by lighting in the image. The images also have a 50% chance to be flipped vertically and a 50% chance to be flipped horizontally. This helps add rotational invariance to the network.

4.5.3 Testing of Neural Network

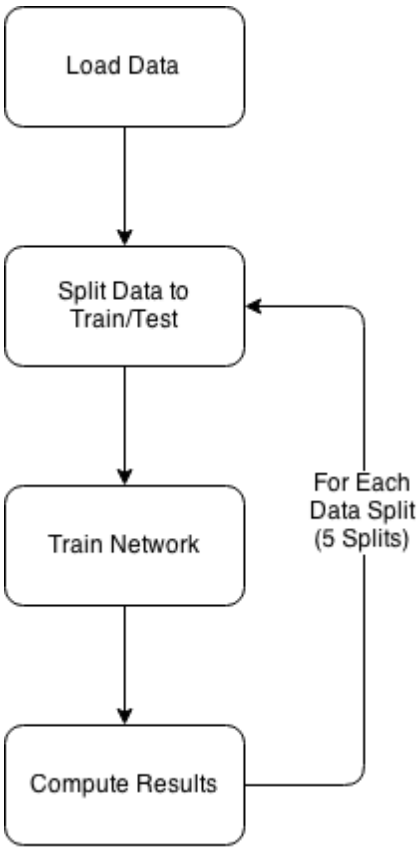

Figure 4.5: Cross validation of neural networks

Testing for each configuration is done using five-fold cross-validation. This is done by splitting the data into five parts. The network is trained five times – each time one part is left out and used as the test set and the other four parts are used for training the network. This way, each image ends up being used in one of the test sets, giving a better overall representation of how the network fits the data. After the entire network finishes training, the specific accuracy per classification is printed to compare which classes learned well and which did not.

4.2 The Mobile Application

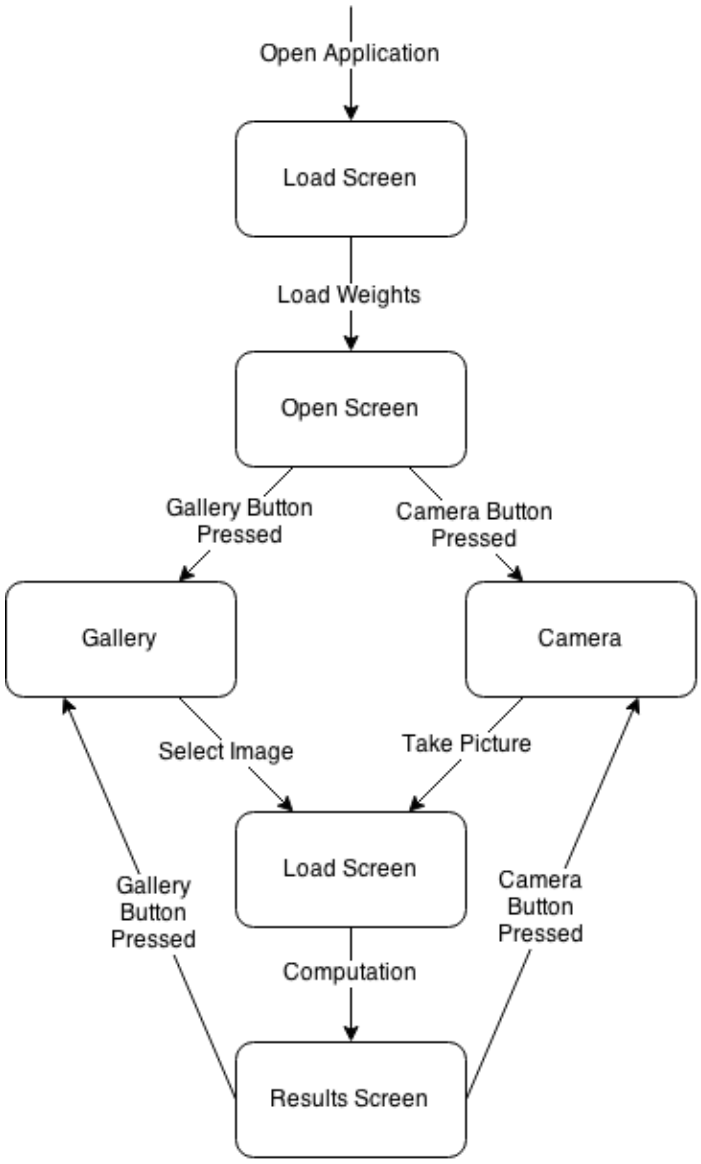

Figure 4.6: Flowchart of the mobile application

The mobile application has two options: 1) to classify an image already on the device, or 2) to take a picture of an image and then classify that image. Figure 4.7 shows the opening screen of the application.

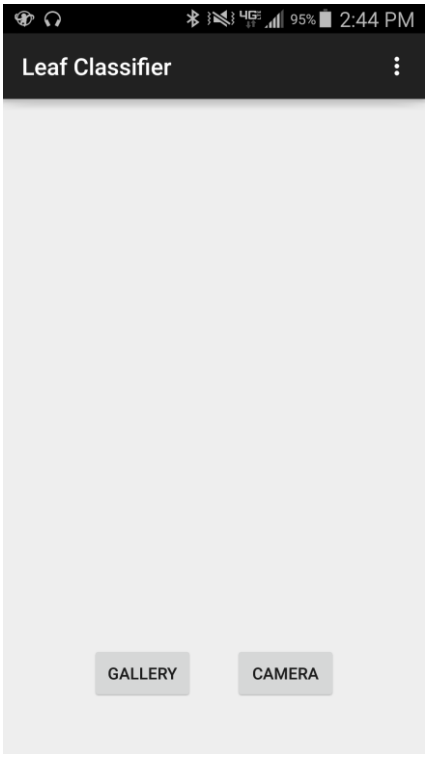

Figure 4.7: Opening screen of the application

If the users select Camera, their camera opens and they can take a picture of a leaf to classify as shown in Figure 4.8. When a picture is taken, it is saved on the phone as if the user took a picture regularly. This saved picture is then classified just as if it was originally selected from the gallery. If users select Gallery, their gallery opens and they can select an image of a leaf to classify as shown in Figure 4.9.

The selected picture is then run through the network. While the calculations are being performed, the app switches to a loading screen as can be seen in Figure 4.10.

Finally, once the classification is completed, the results are displayed on screen as can be seen in Figure 4.11. The best three results are presented with their percent likelihoods and an image of the species. This screen has options to select a new image in the gallery or take a new picture.

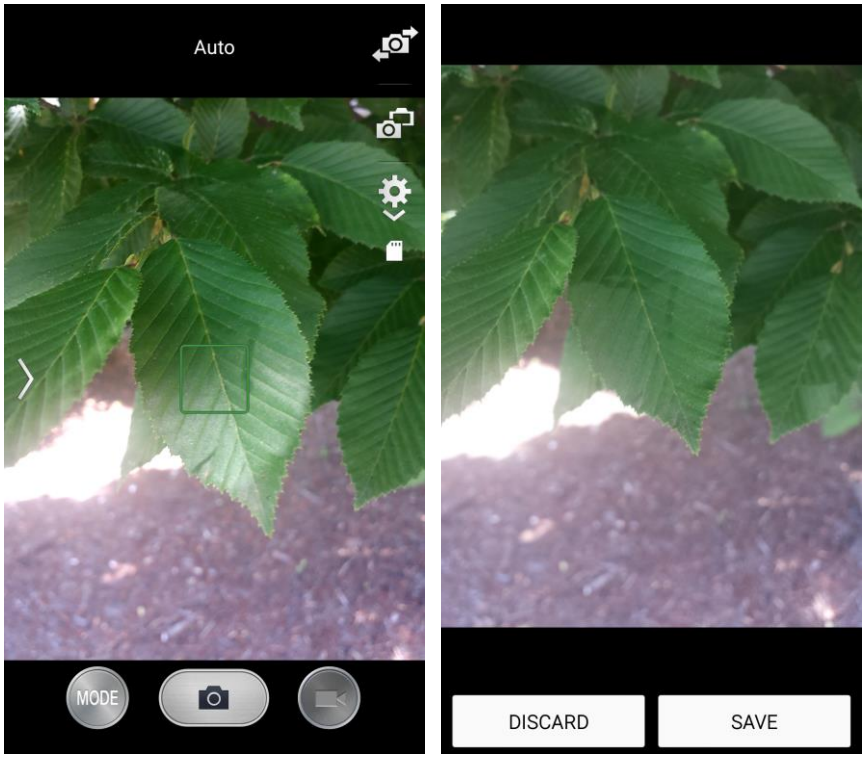

Figure 4.8: Sample picture of a leaf taken with a phone camera

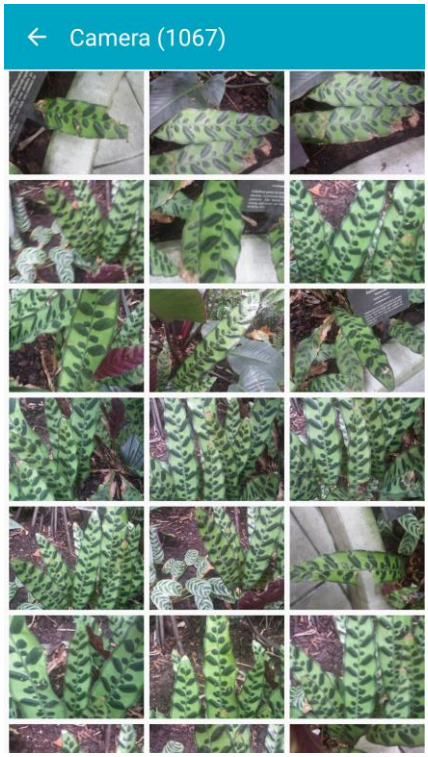

Figure 4.9: Image selection in the gallery

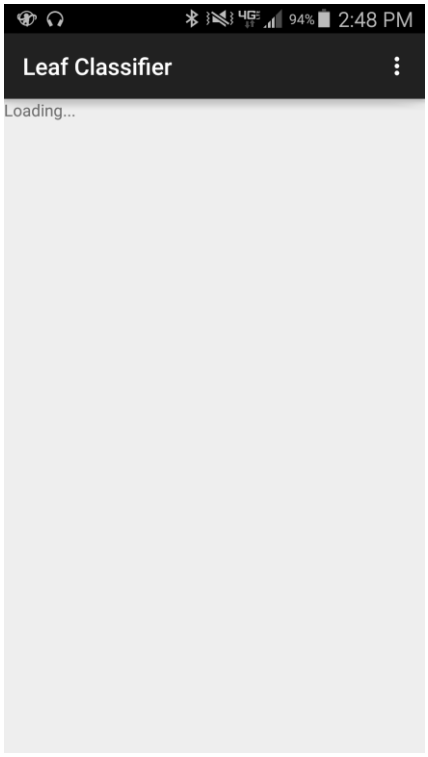

Figure 4.10: Loading screen

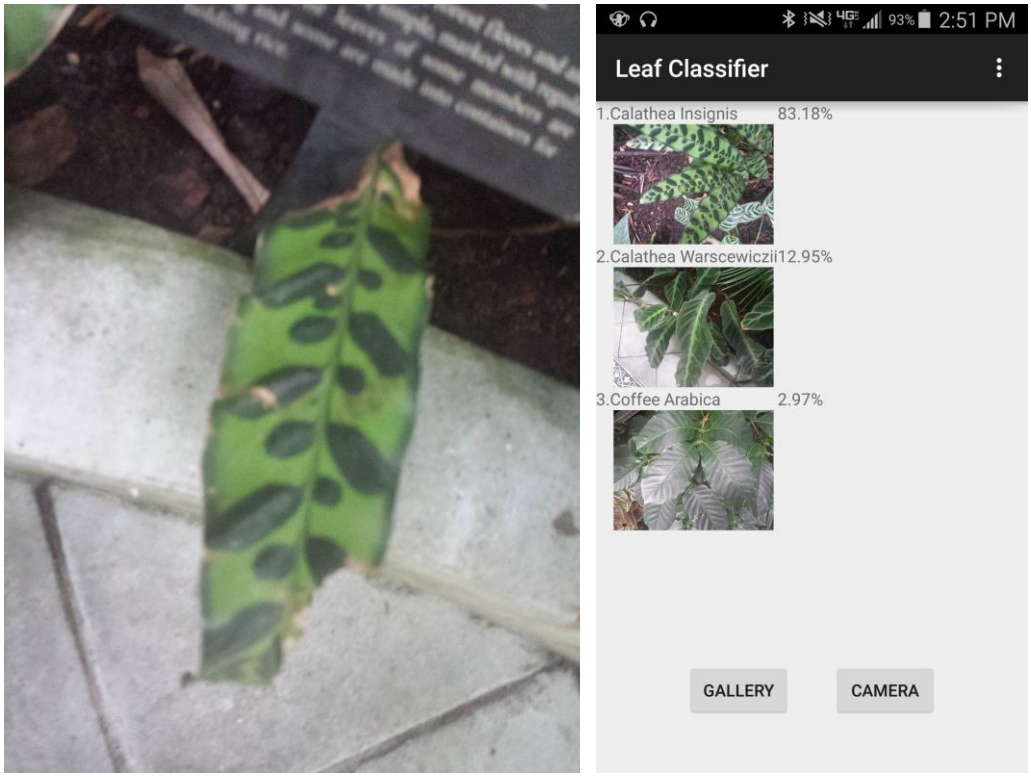

Figure 4.11: Results (right) for the original image (left)

Chapter 5 Results

5.1 Introduction

In this chapter, the results of the networks mentioned in Table 4-1 will be provided. Section 5.2 will discuss the accuracies of each species of leaves and shows several results of running the mobile application on different images. Section 5.3 will discuss the overall accuracies with respect to each network. Finally, Section 5.4 will discuss the load and computation times of each network.

The results were obtained through five-fold cross validation of each network. After every epoch of training, the rankings of each classification were compiled and printed such that the percentage of correct classifications in first, second, and so on could be seen. After training, the number of correct classifications with respect to each leaf were printed out to see how well classes of leaves were represented. These accuracies were compiled into several graphs, illustrating the results in Section 5.2 and Section 5.3.

The load times and computation times for running the networks on a mobile device were also gathered. The results of the load and computation times can be seen in Section 5.4.

5.2 Leaf Species Accuracies

It is also important to look at the individual species test accuracies to see where biases may lie in the network. Figures 5.1-5.15 show screenshots of classifications of the pictures that were taken for the purpose of this research with a correct first classification. Figures 5.16-5.20 show screenshots of some classifications with correct second or third classifications. Figures 5.21-5.24 show screenshots of some classifications with the top-3 results completely incorrect. Figure 5.25 shows the top-1 accuracies of each network for each classification. Table 5-1 shows the numerical values for these top-1 accuracies.

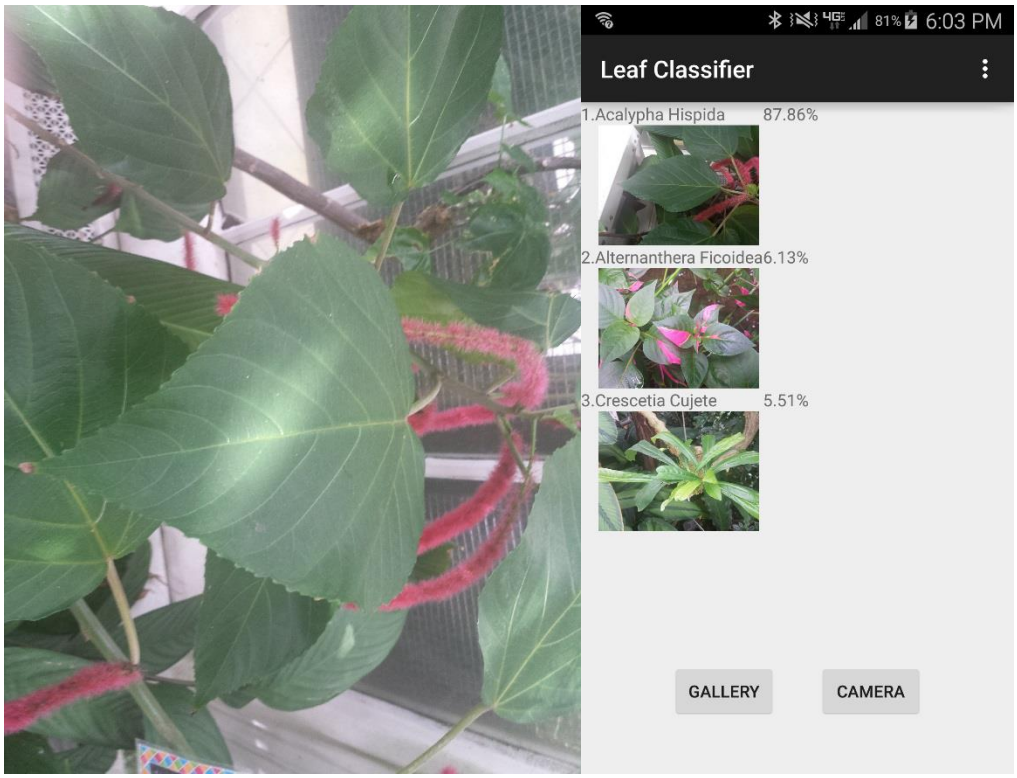

Figure 5.1: Results of Acalypha Hispida

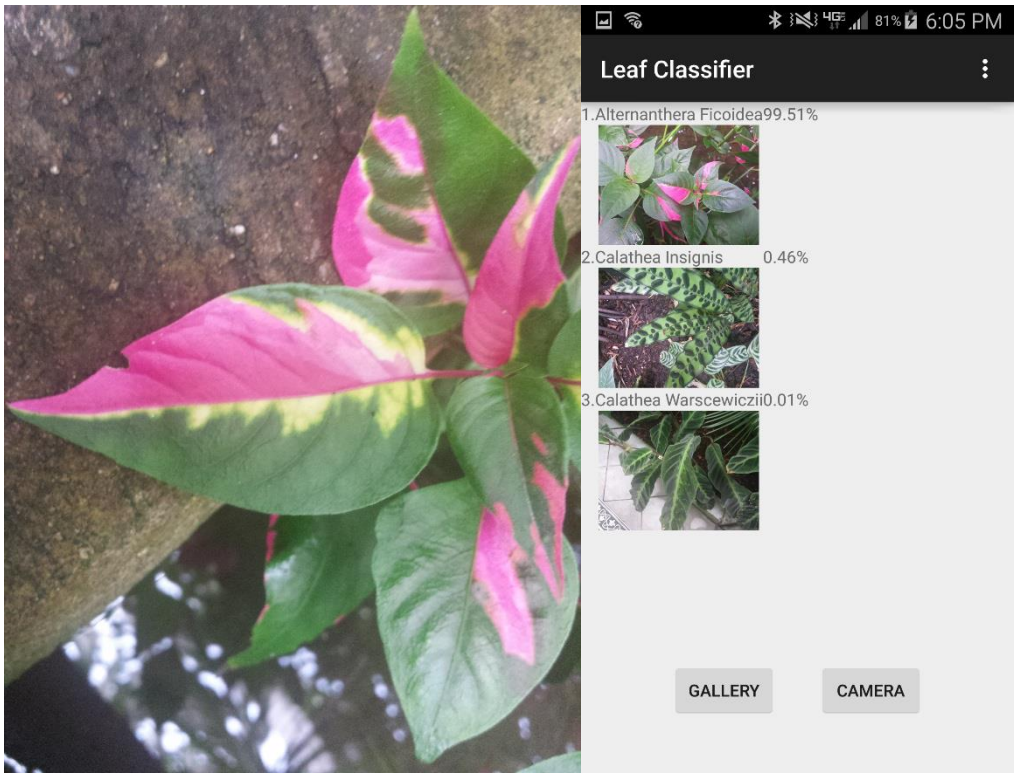

Figure 5.2: Results of Alternanthera Ficoidea

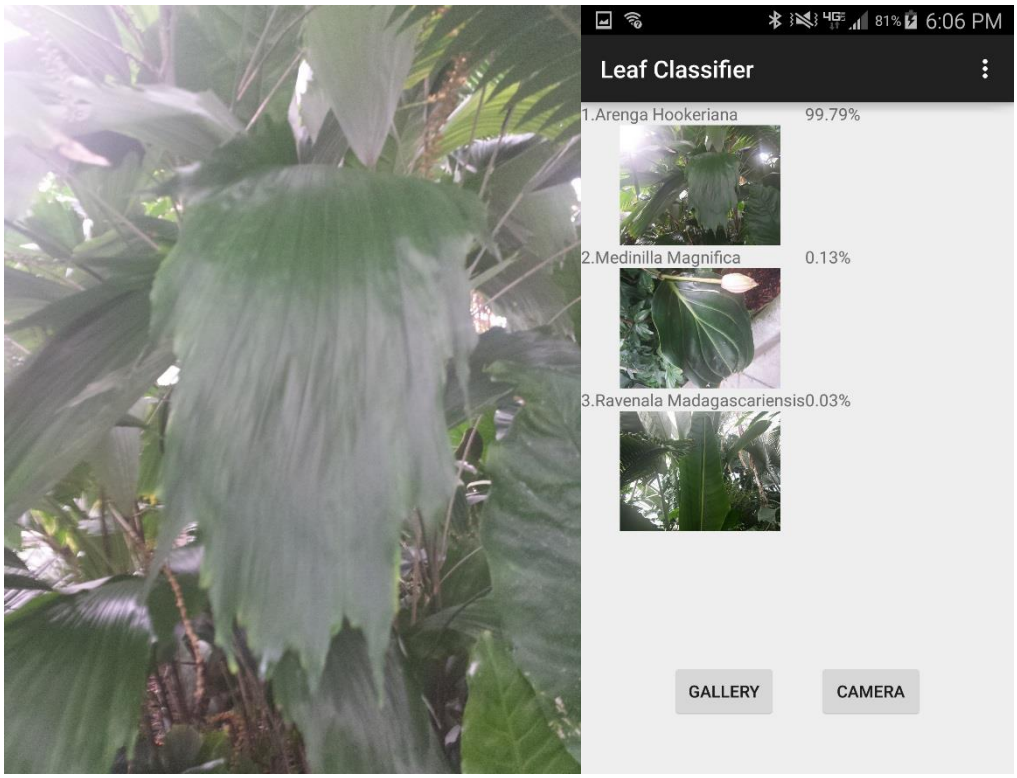

Figure 5.3: Results of Arenga Hookeriana

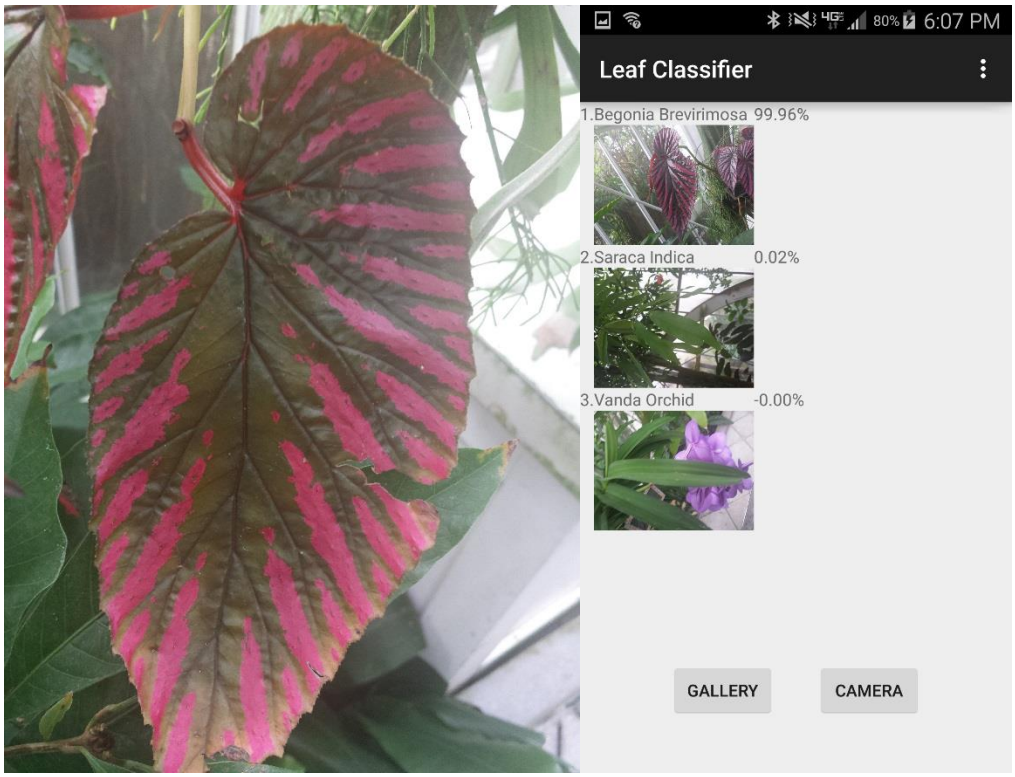

Figure 5.4: Results of Begonia Brevirimosa

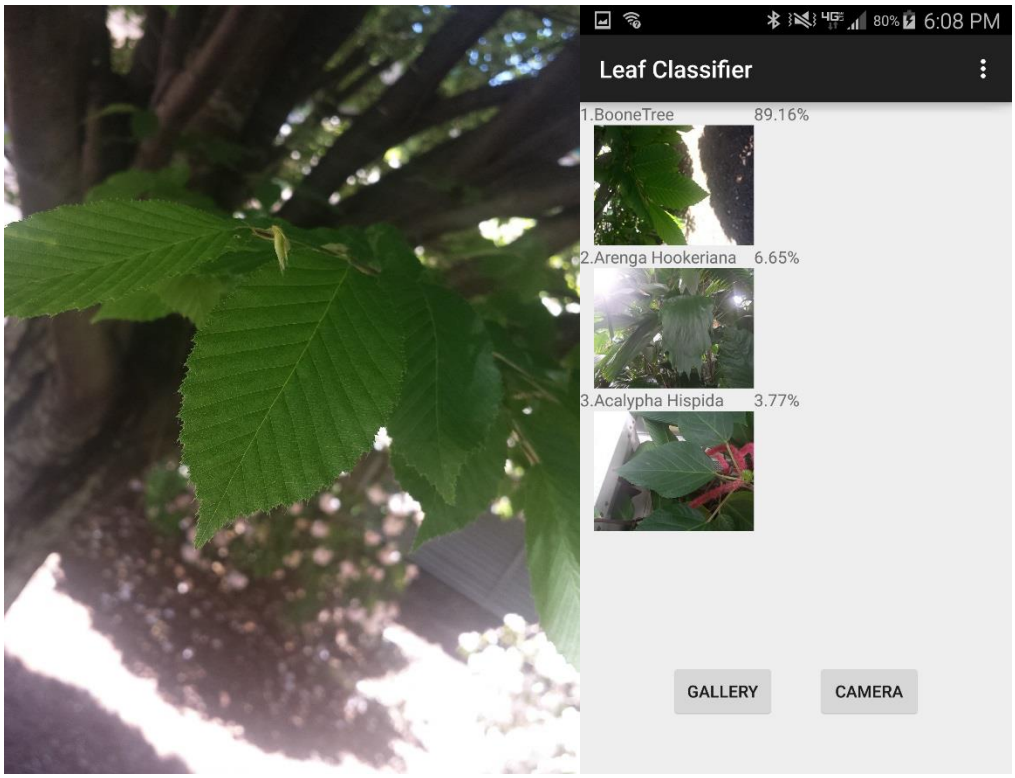

Figure 5.5: Results of unidentified tree in Boone

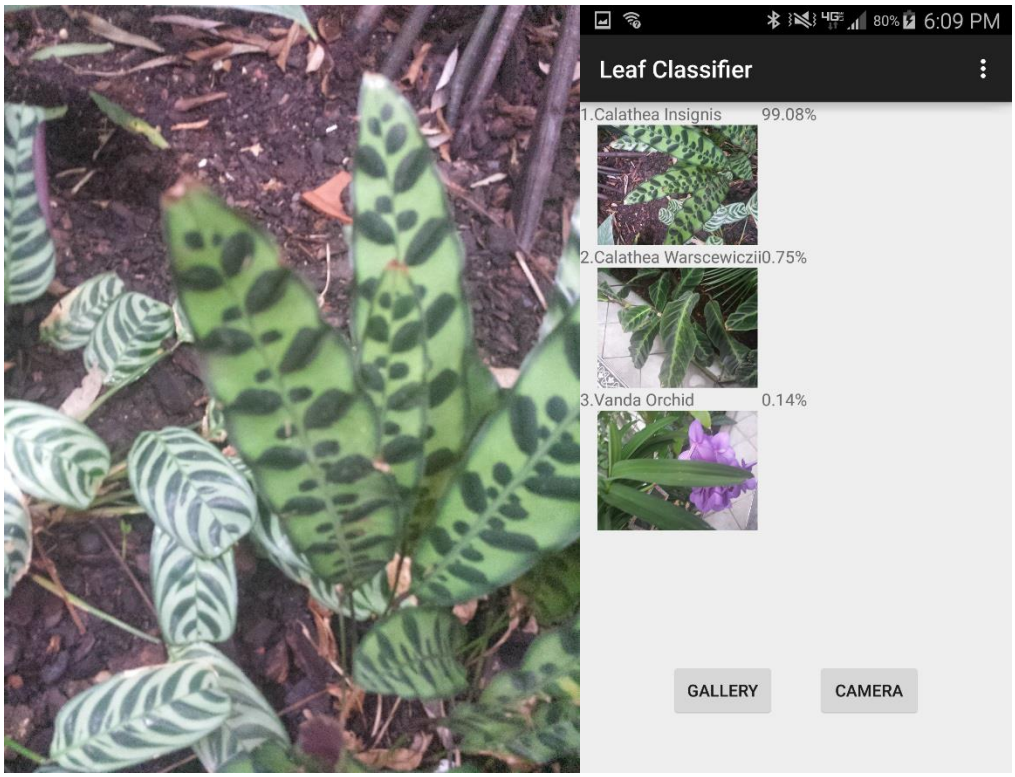

Figure 5.6: Results of Calathea Insignis

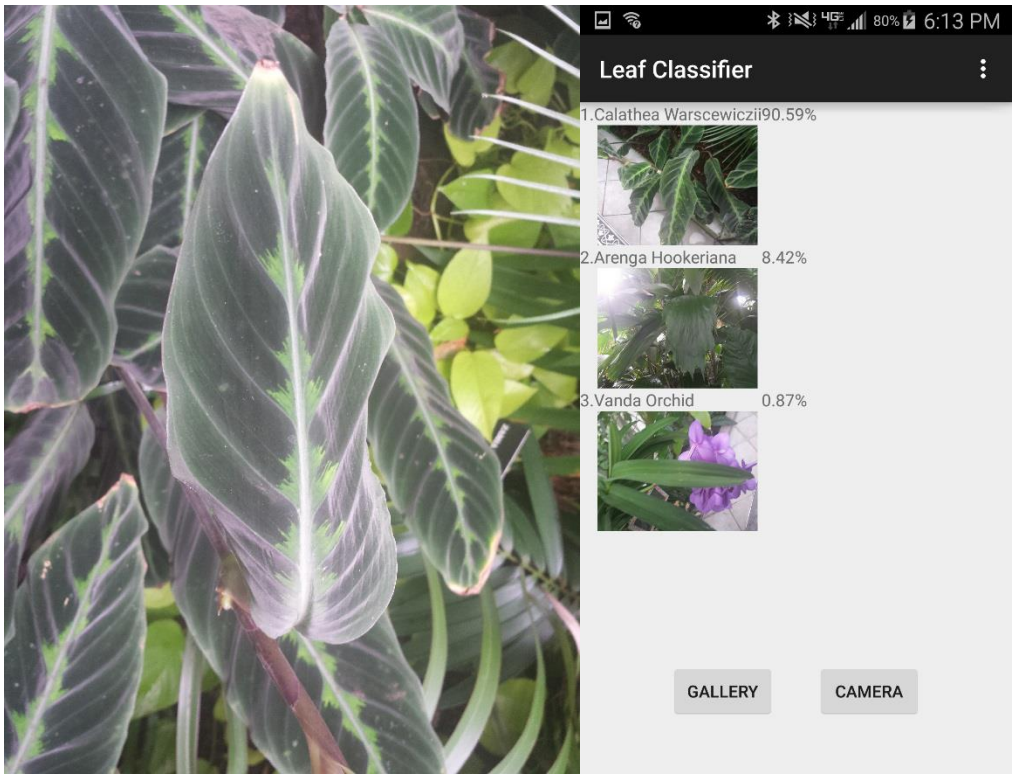

Figure 5.7: Results of Calathea Warscewiczii

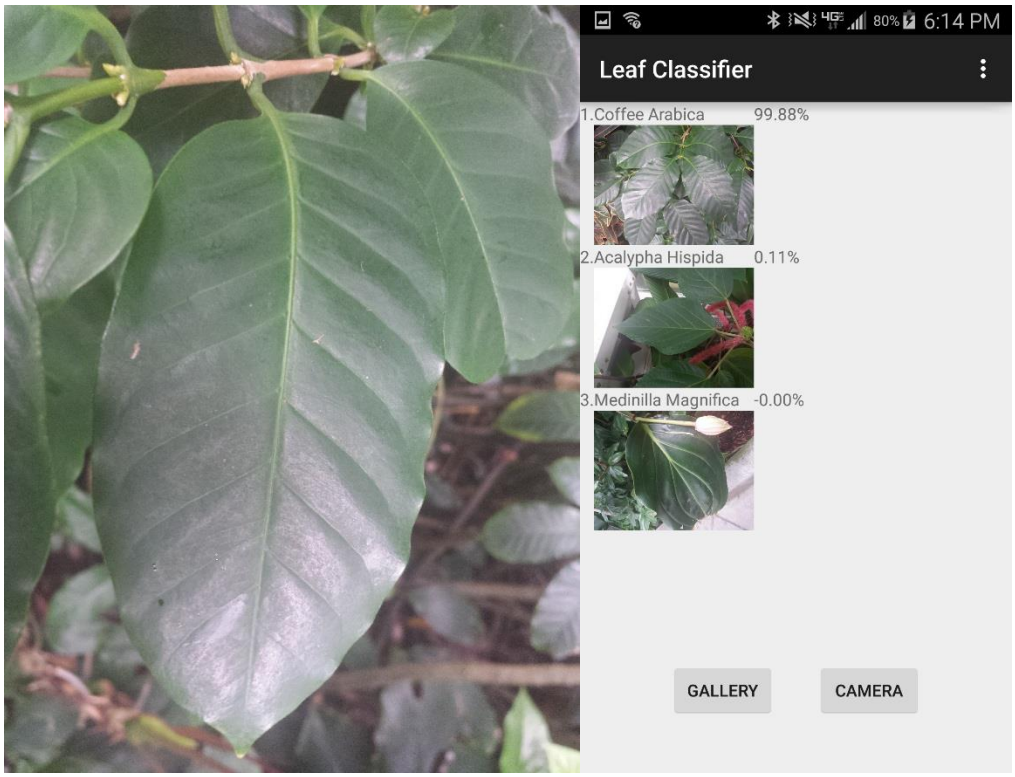

Figure 5.8: Results of Coffee Arabica

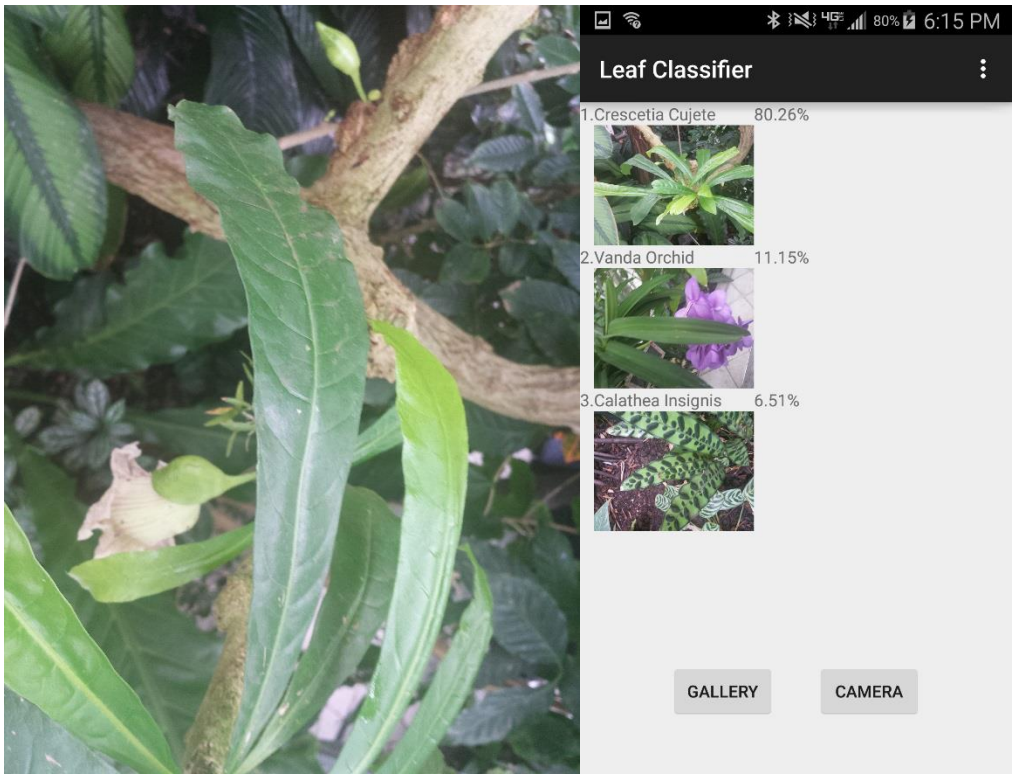

Figure 5.9: Results of Crescetia Cujete

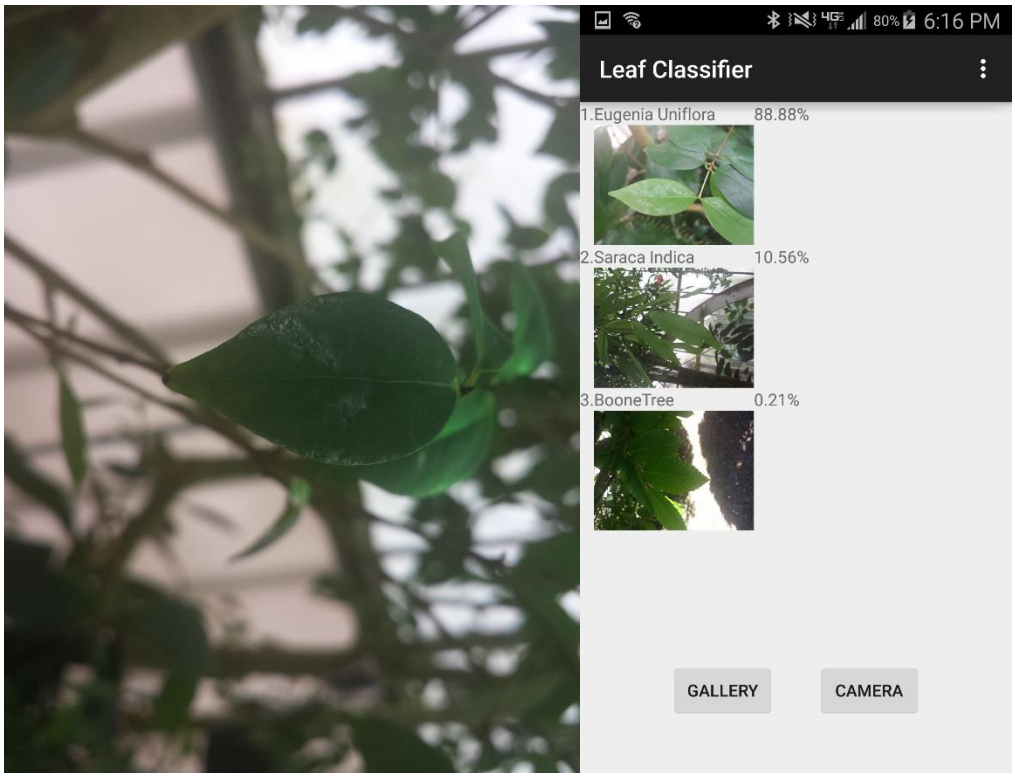

Figure 5.10: Results of Eugenia Uniflora

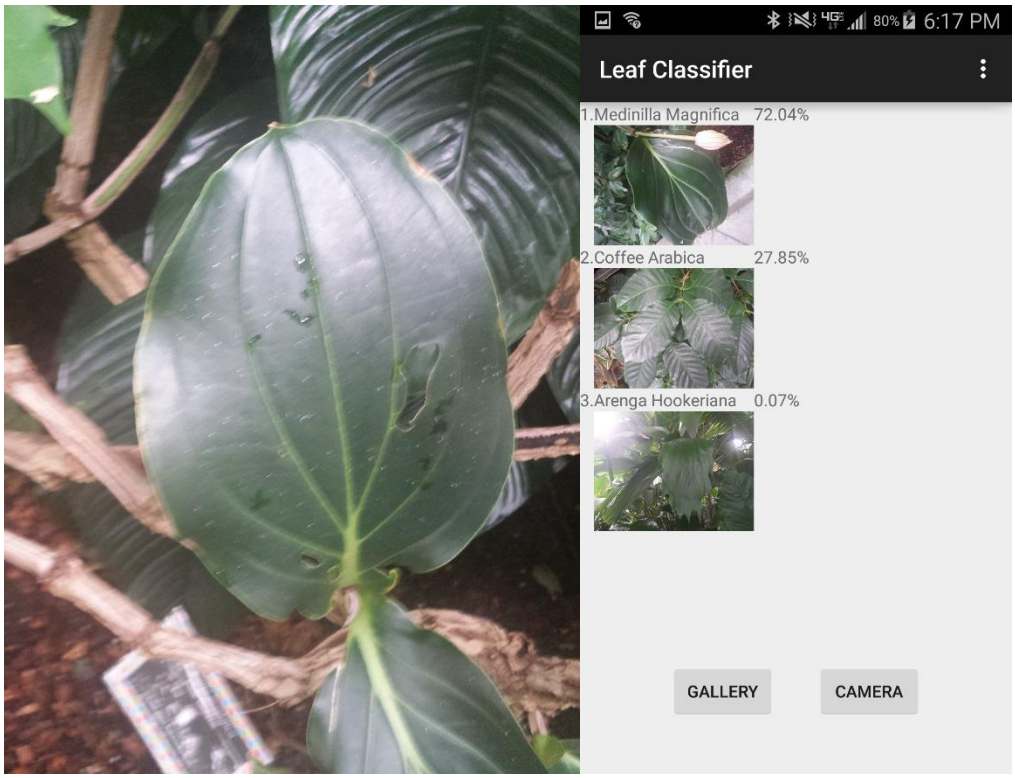

Figure 5.11: Results of Medinilla Magnifica

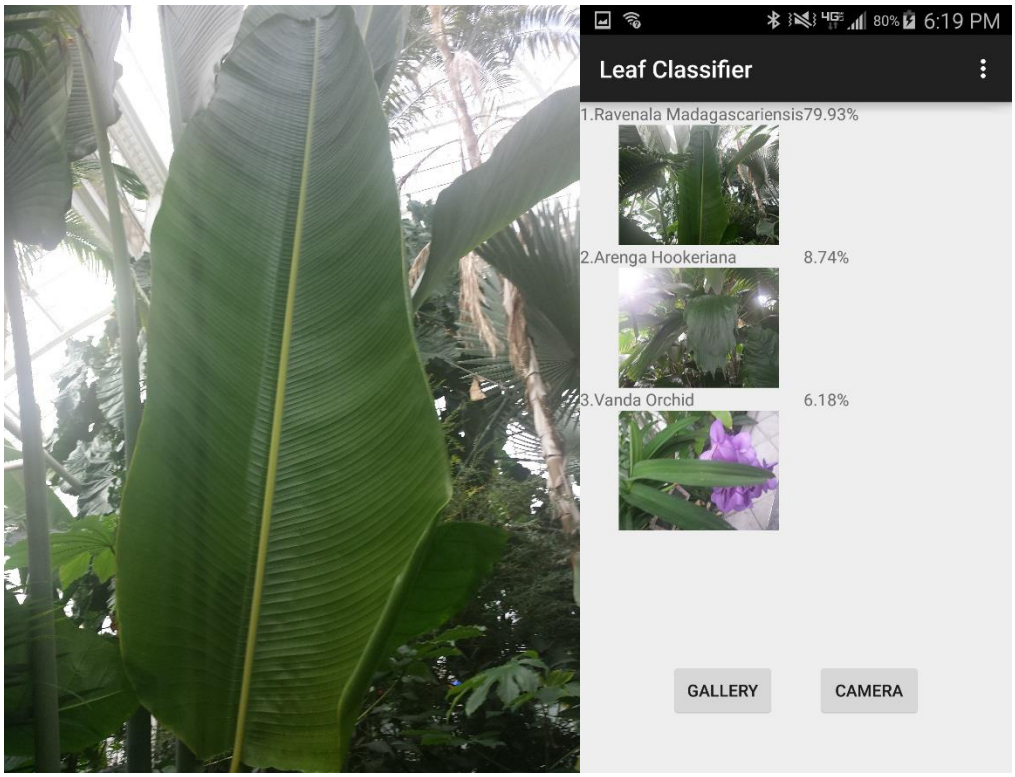

Figure 5.12: Results of Ravenala Madagascariensis

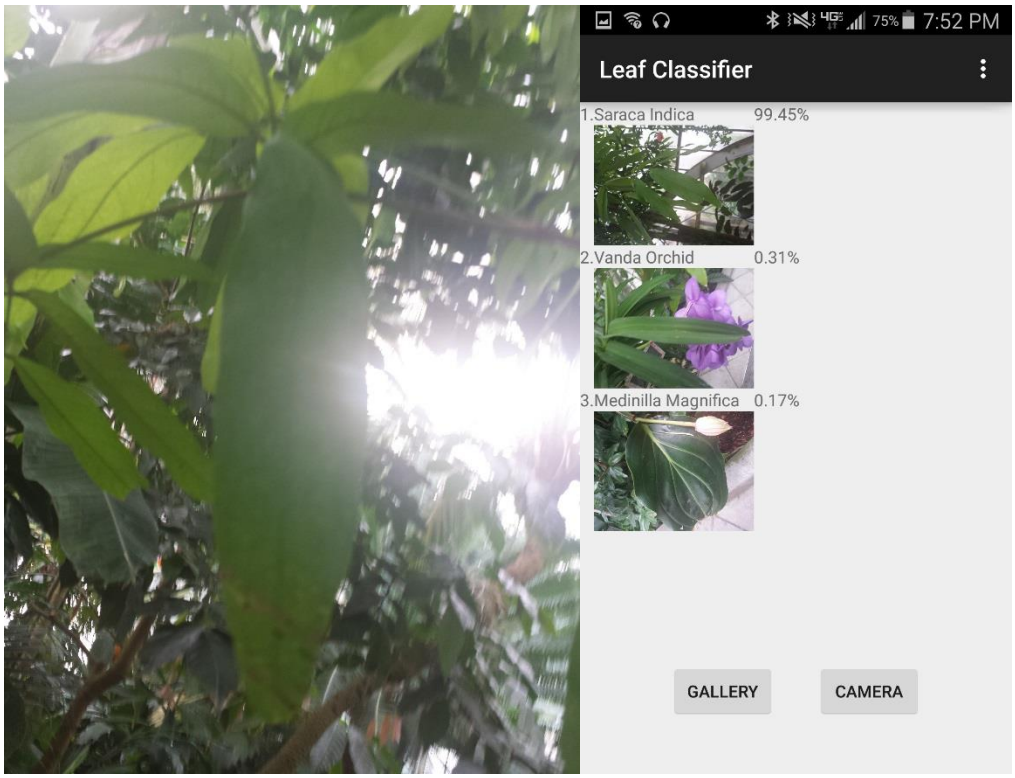

Figure 5.13: Results of Saraca Indica

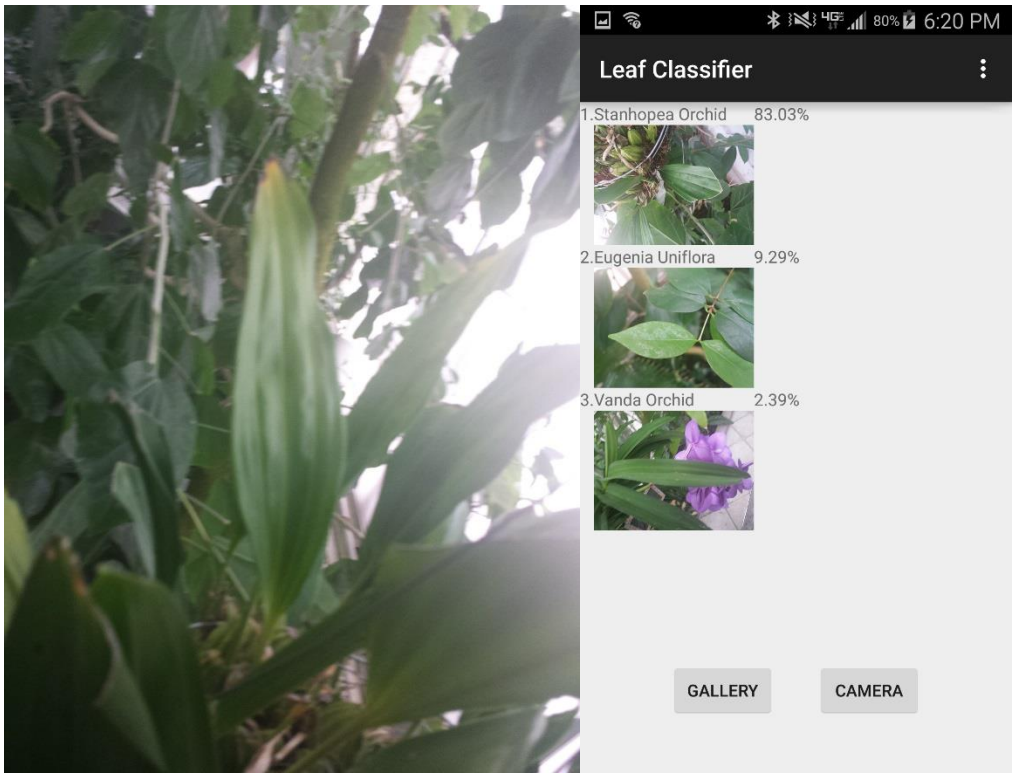

Figure 5.14: Results of Stanhopea Orchid

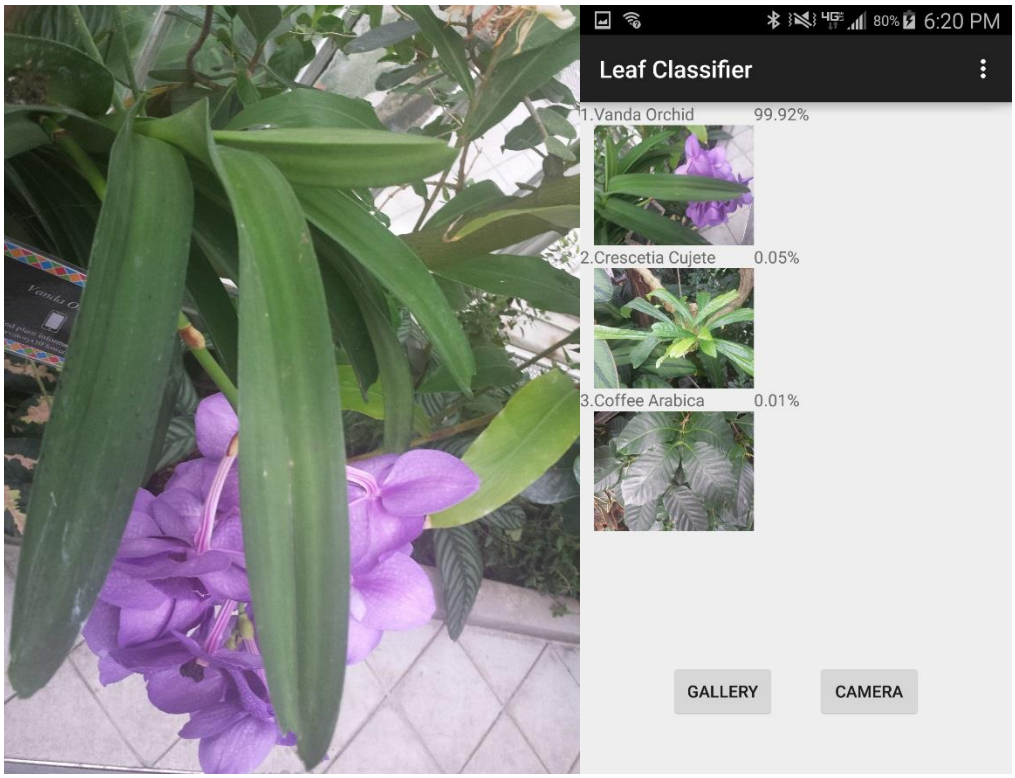

Figure 5.15: Results of Vanda Orchid

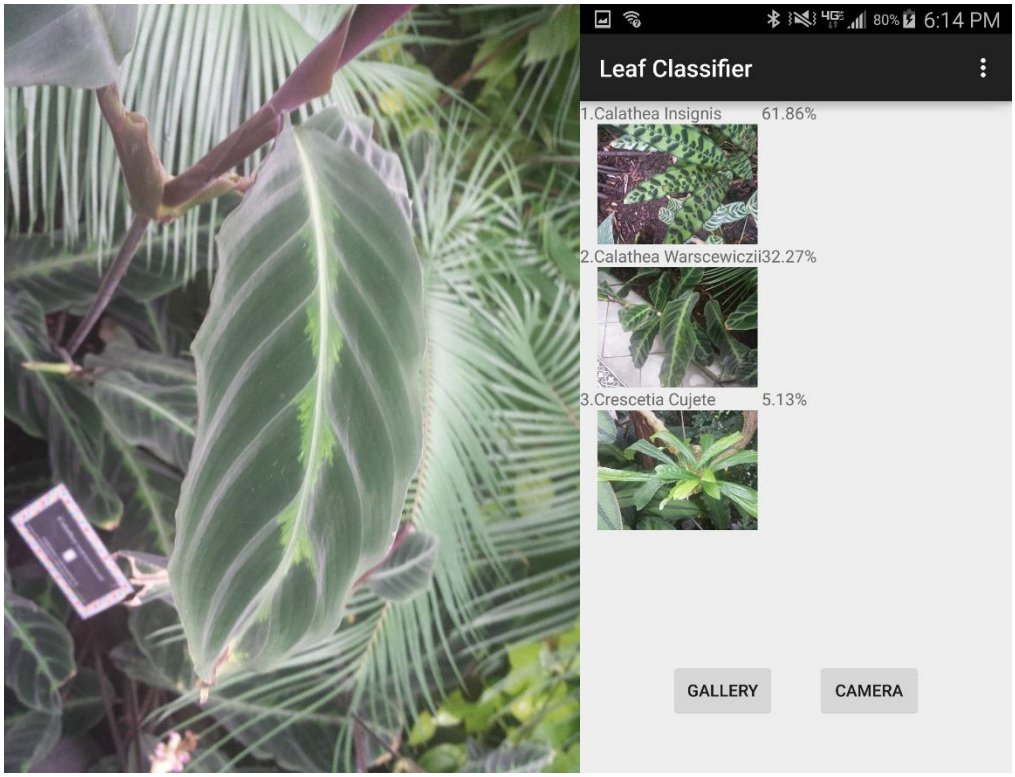

Figure 5.16: Results of Calathea Insignis

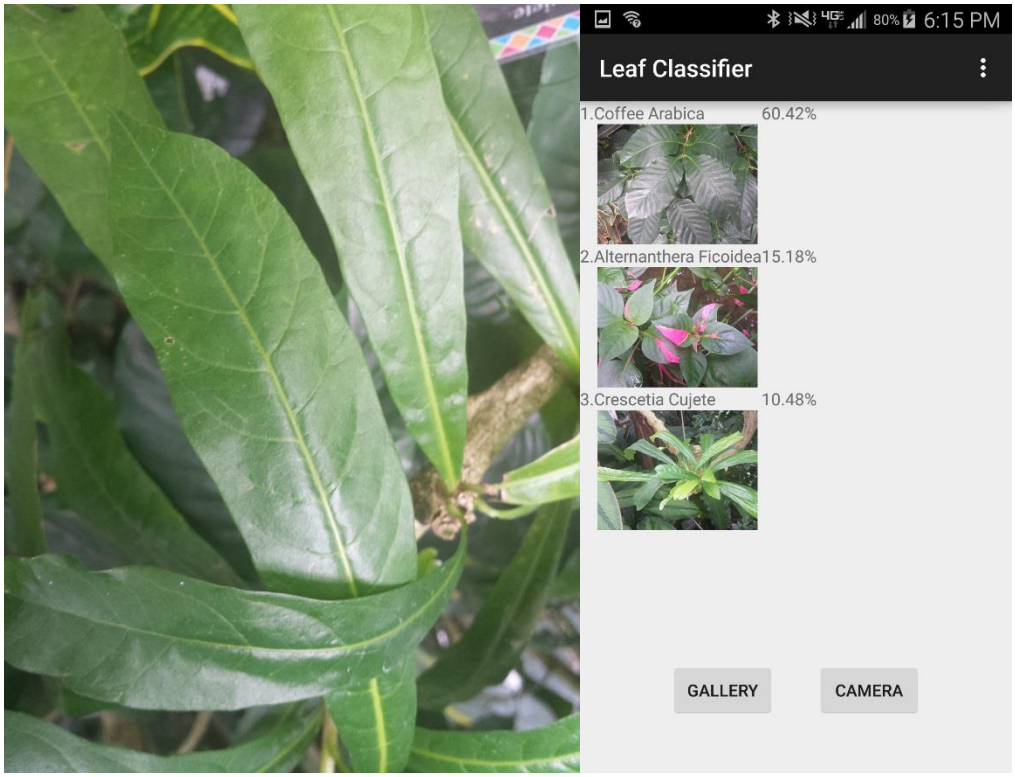

Figure 5.17: Results of Crescetia Cujete

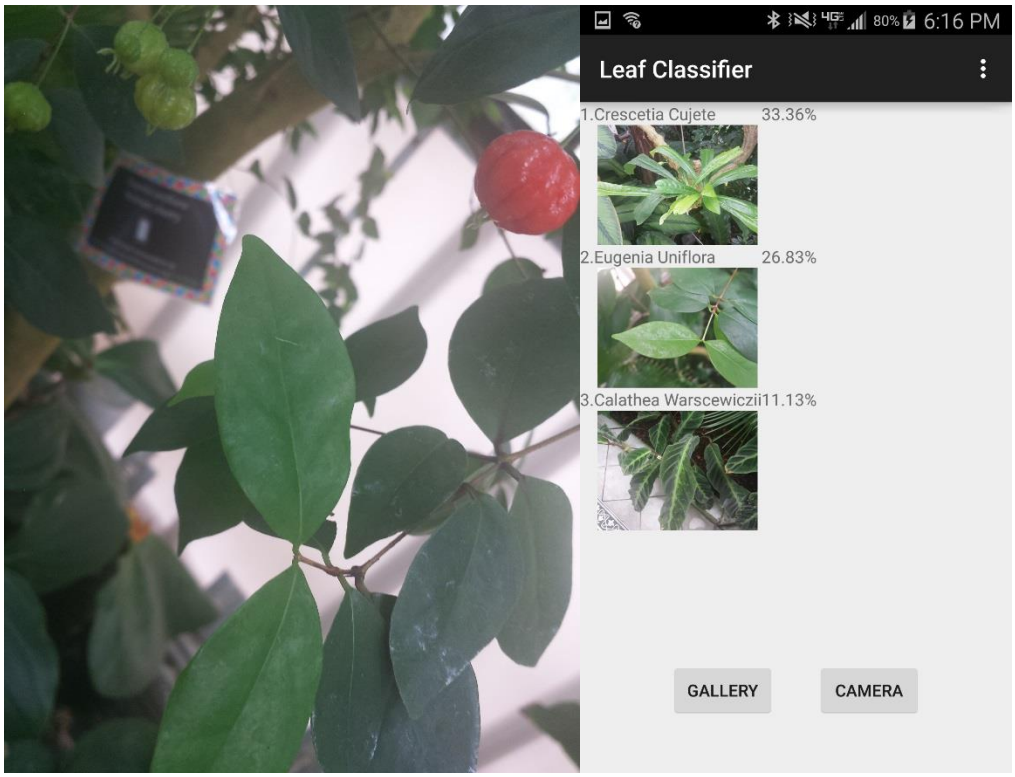

Figure 5.18: Results of Eugenia Uniflora

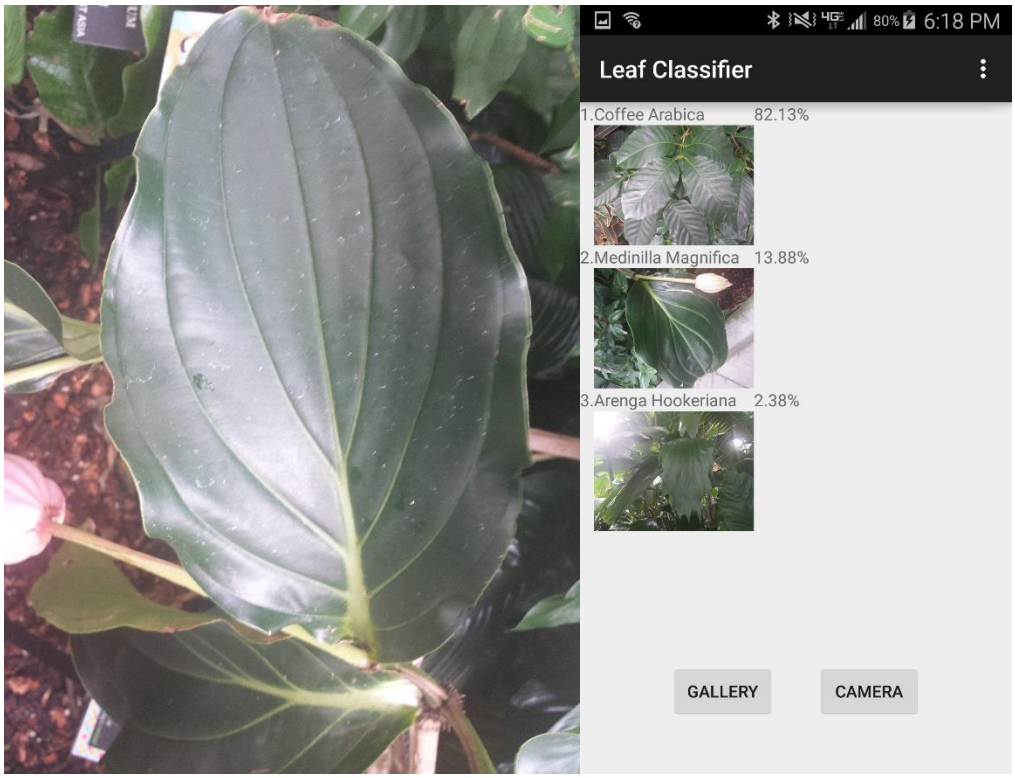

Figure 5.19: Results of Medinilla Magnifica

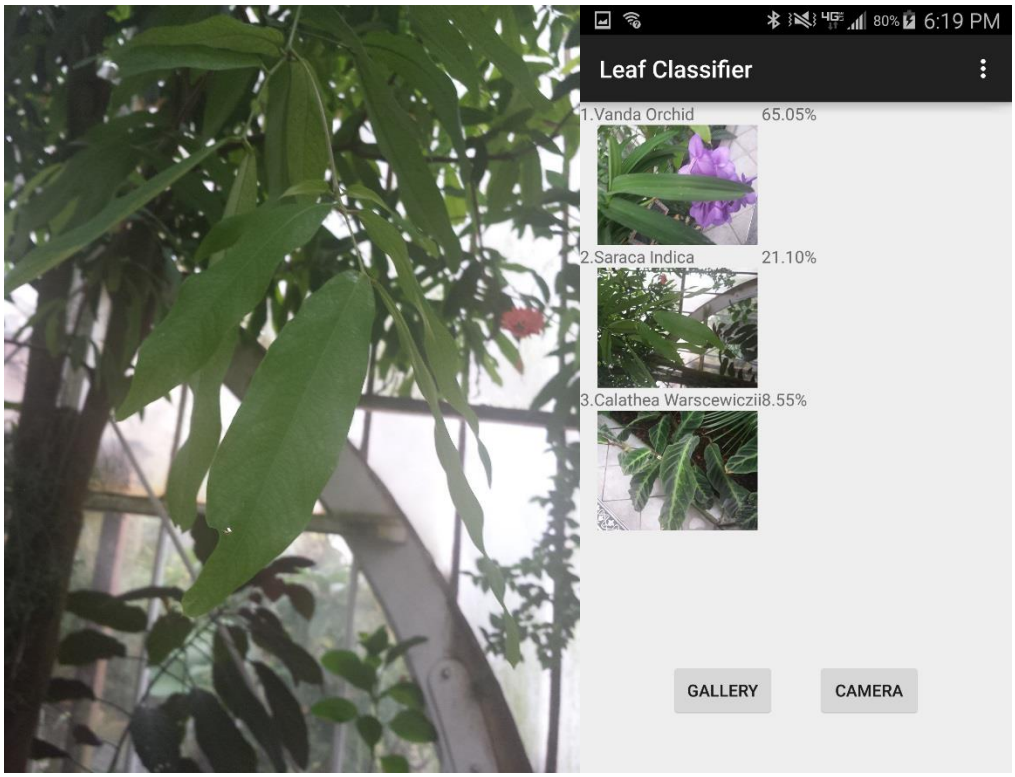

Figure 5.20: Results of Saraca Indica

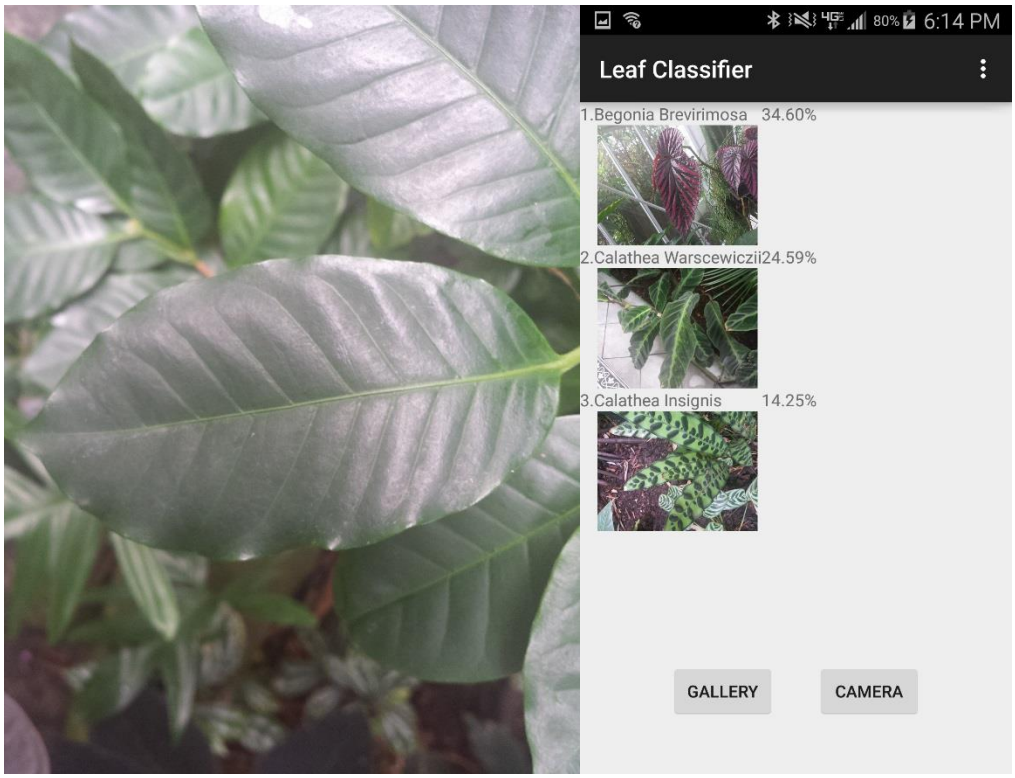

Figure 5.21: Results of Coffee Arabica

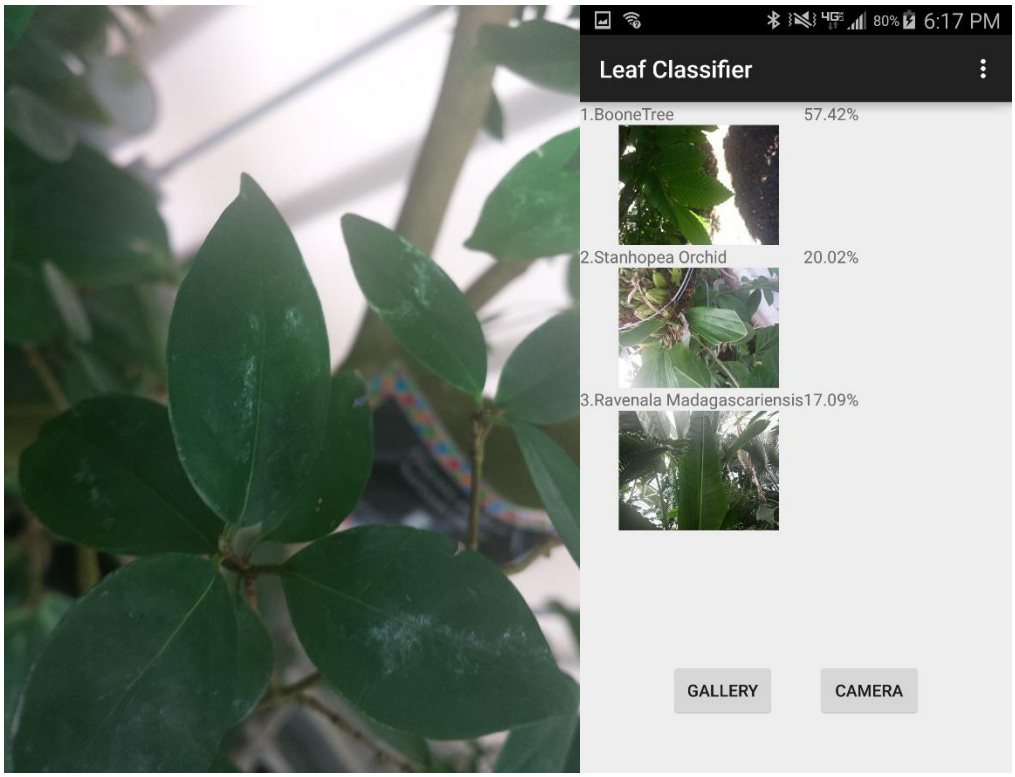

Figure 5.22: Results of Eugenia Uniflora

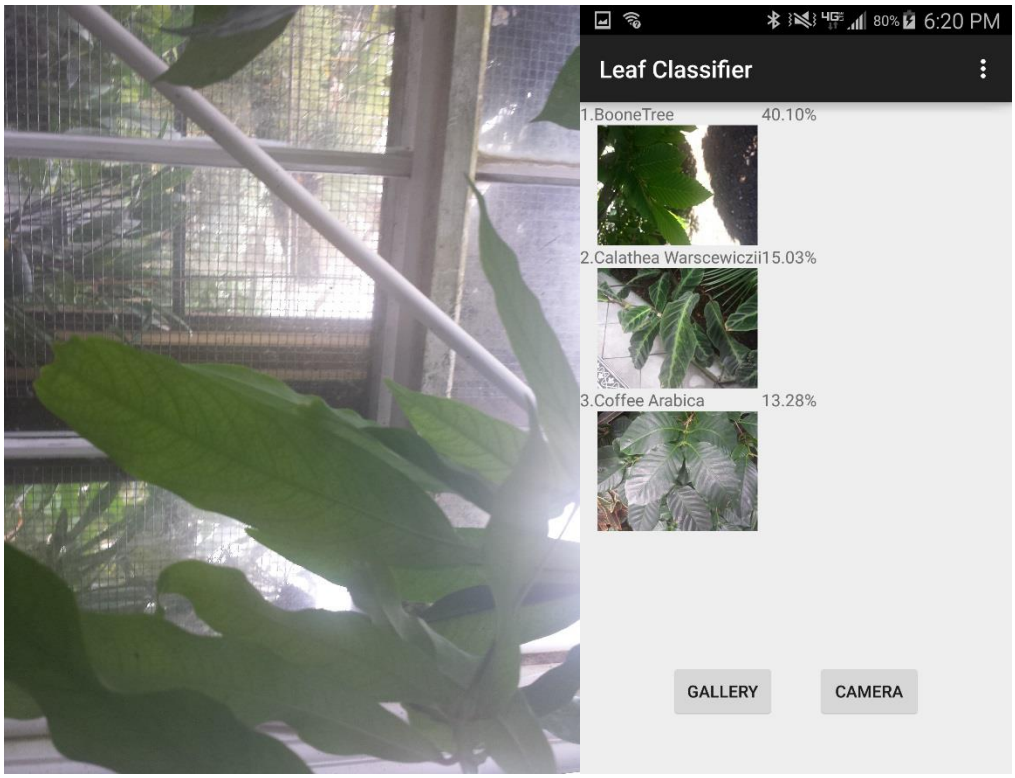

Figure 5.23: Results of Saraca Indica

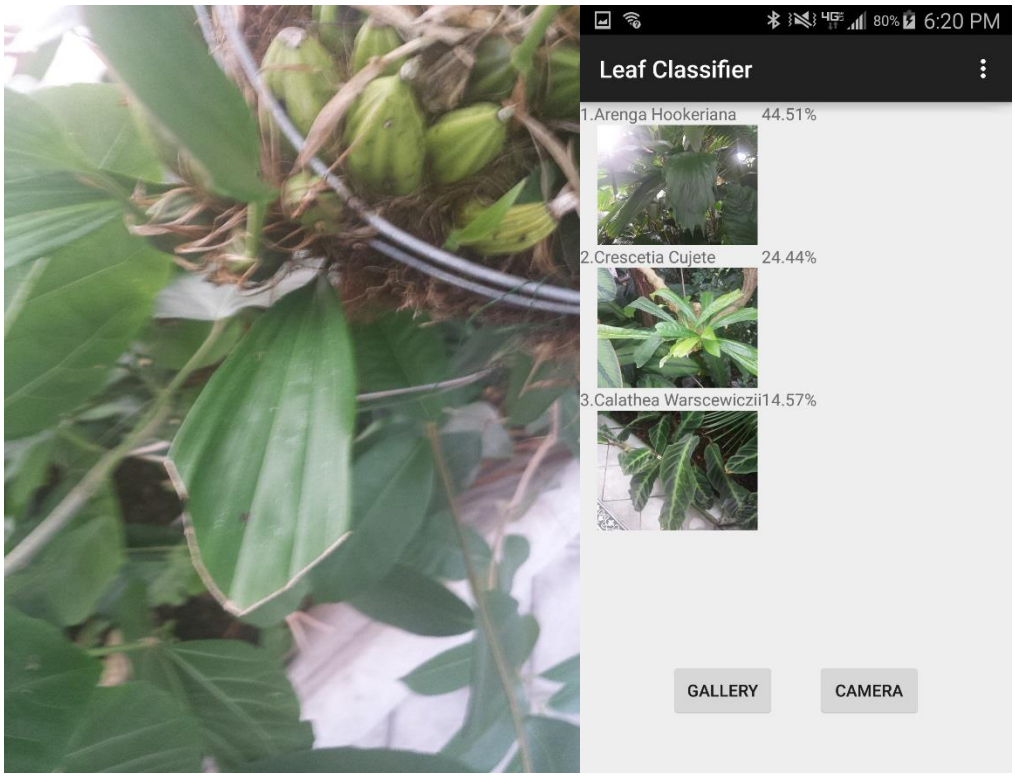

Figure 5.24: Results of Stanhopea Orchid

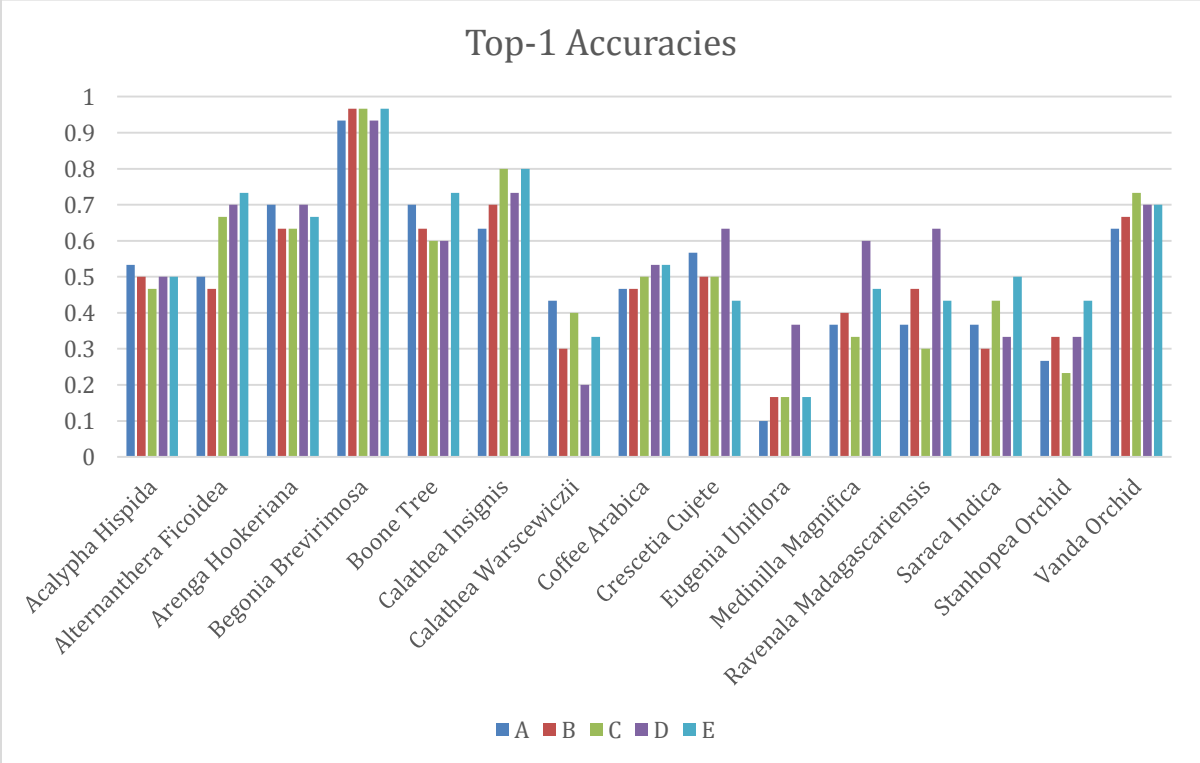

Figure 5.25: Top-1 accuracies for each class

Table 5-1: Top-1 Accuracies for each class

	A	B	C	D	E
Acalypha Hispida	0.533333	0.5	0.466667	0.5	0.5
Alternanthera Ficoidea	0.5	0.466667	0.666667	0.7	0.733333
Arenga Hookeriana	0.7	0.633333	0.633333	0.7	0.666667
Begonia Brevirimosa	0.933333	0.966667	0.966667	0.933333	0.966667
Boone Tree	0.7	0.633333	0.6	0.6	0.733333
Calathea Insignis	0.633333	0.7	0.8	0.733333	0.8
Calathea Warscewiczii	0.433333	0.3	0.4	0.2	0.333333
Coffee Arabica	0.466667	0.466667	0.5	0.533333	0.533333
Crescetia Cujete	0.566667	0.5	0.5	0.633333	0.433333
Eugenia Uniflora	0.1	0.166667	0.166667	0.366667	0.166667
Medinilla Magnifica	0.366667	0.4	0.333333	0.6	0.466667
Ravenala Madagascariensis	0.366667	0.466667	0.3	0.633333	0.433333
Saraca Indica	0.366667	0.3	0.433333	0.333333	0.5
Stanhopea Orchid	0.266667	0.333333	0.233333	0.333333	0.433333
Vanda Orchid	0.633333	0.666667	0.733333	0.7	0.7

5.3 Overall Neural Network Accuracies

Figure 5.26 shows how the five networks differ in top-3 test accuracy over the 100 training iterations. This accuracy is an average for the entire five-fold cross validation process. Top-3 accuracy is reported because the mobile application displays the top-3 results with pictures, so getting a high top-3 accuracy will allow the user to see the appropriate classification often. Figures 5.27-5.31 show the train and test accuracy of the individual networks to compare how well the networks fit the data.

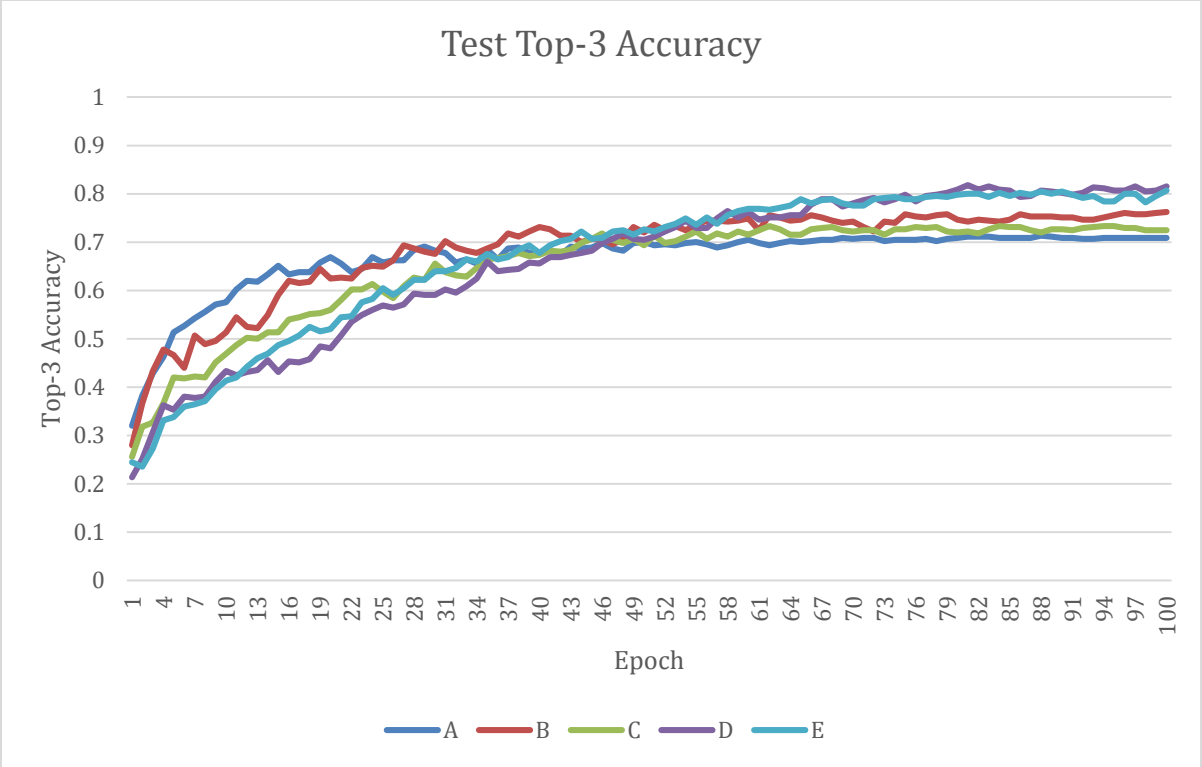

Figure 5.26: Test accuracy of neural network configurations

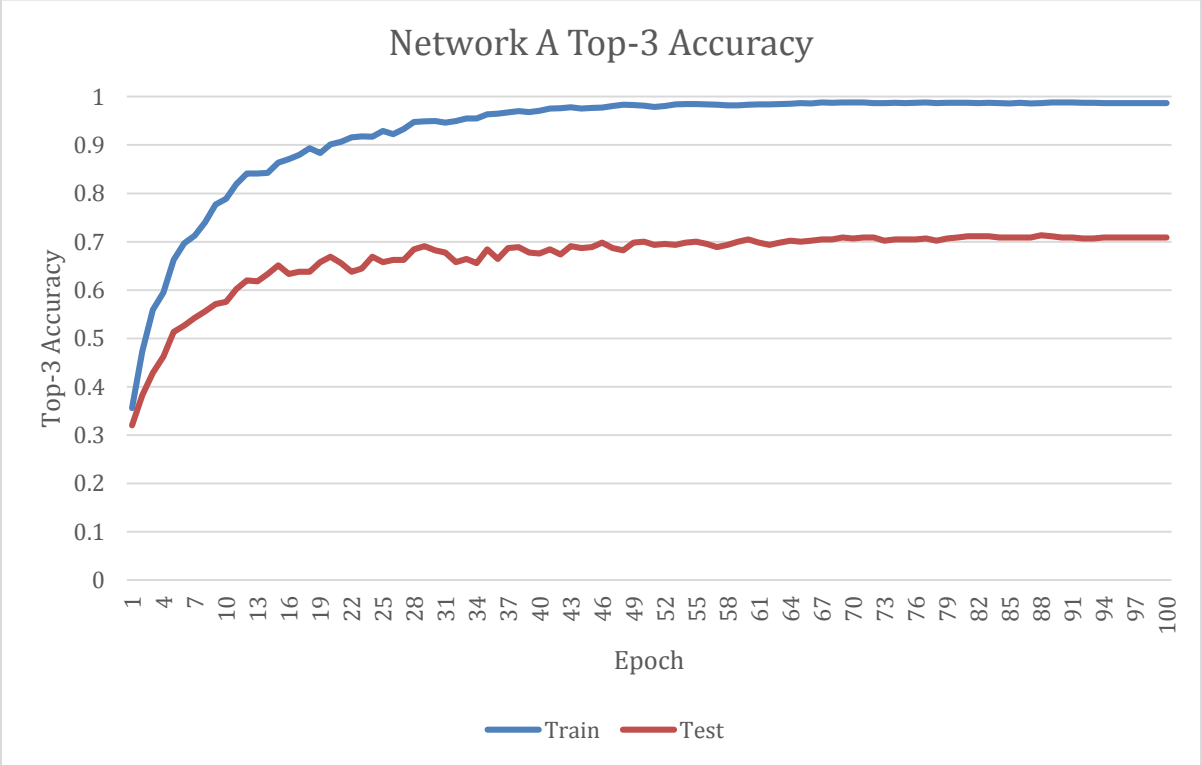

Figure 5.27: Train and test accuracies of Network A

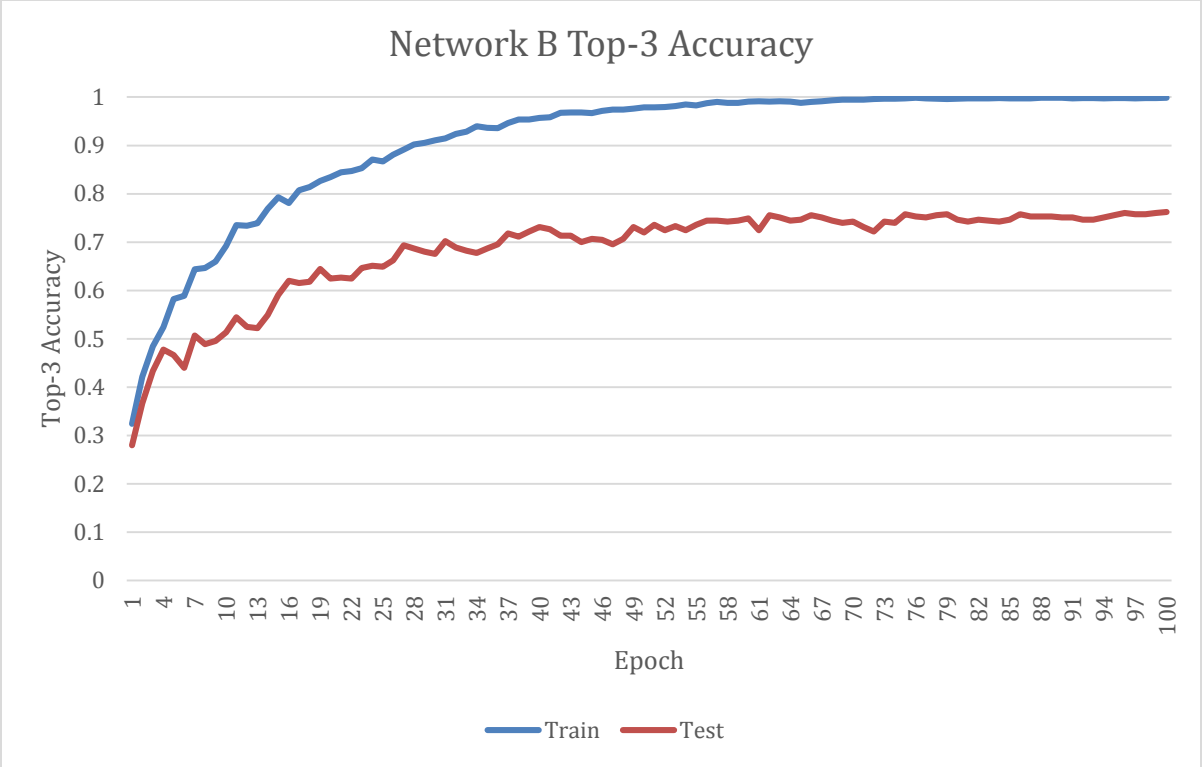

Figure 5.28: Train and test accuracies Network B

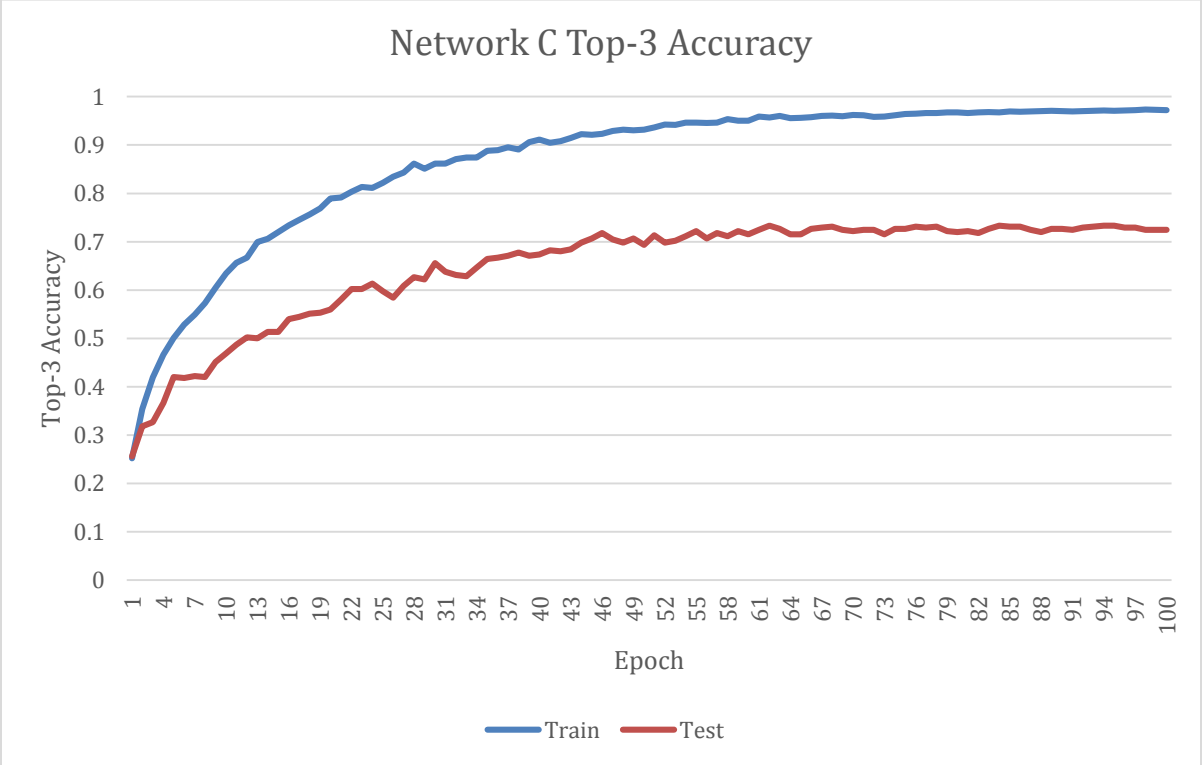

Figure 5.29: Train and test accuracies of Network C

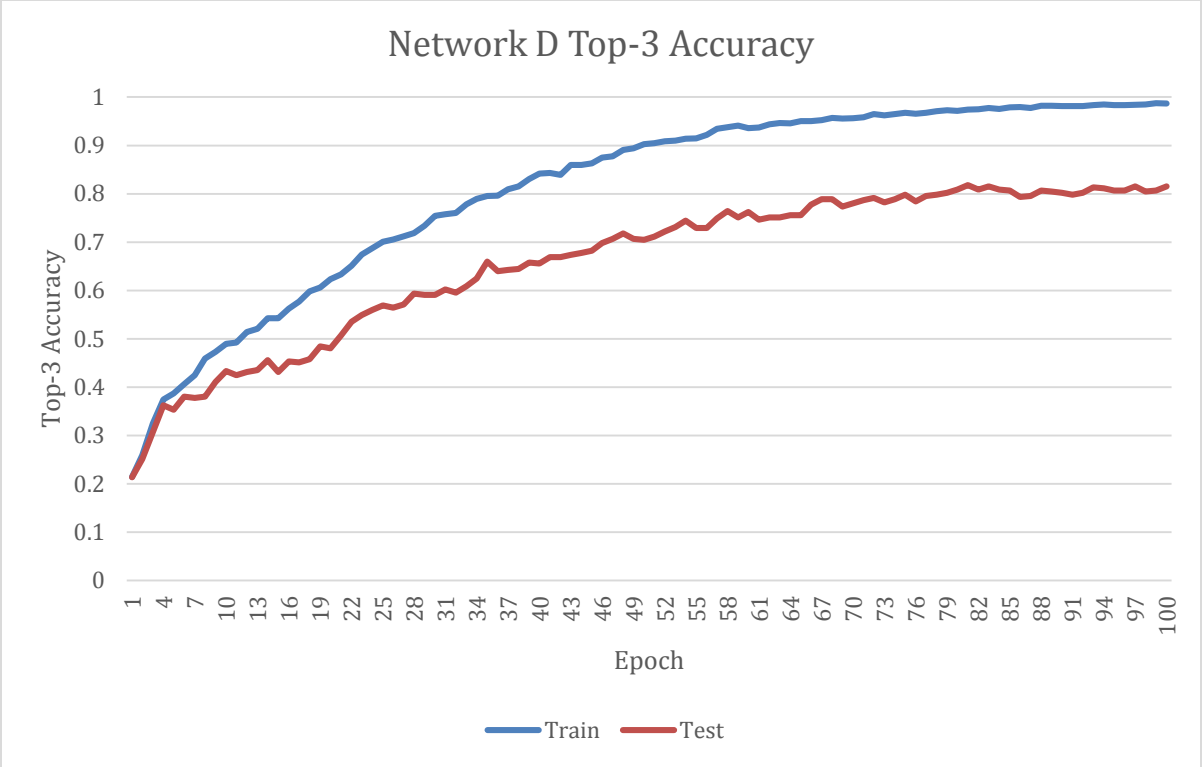

Figure 5.30: Train and test accuracies of Network D

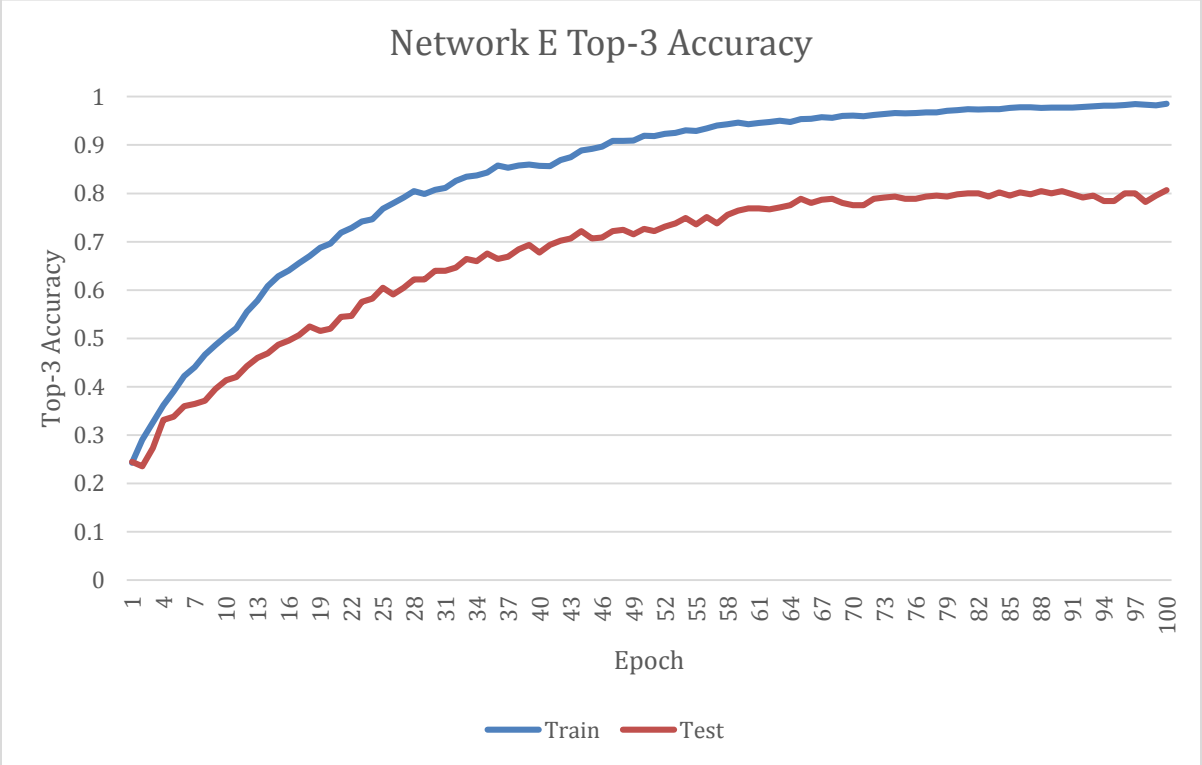

Figure 5.31: Train and test accuracies of Network E

The accuracies for the top three results of the networks can be seen in Table 5-2.

Table 5-2: Top-3 accuracies

	1	2	3	Total Top-3
A	0.504444	0.117778	0.086667	0.708889
B	0.5	0.171111	0.091111	0.762222
C	0.515556	0.126667	0.082222	0.724444
D	0.566667	0.168889	0.08	0.815556
E	0.56	0.168889	0.077778	0.806667

5.4 Mobile Application Load and Computation Times

A final result for this mobile application is the load and computation time of the application. A slow mobile application is frustrating to the user and will not be used. Table 5-3 shows the average load and computation times of the networks over 5 uses.

Table 5-3: Load and Computation Time of Networks

	Load	Compute
A	21.2796	2.4598
B	21.4642	2.5634
C	5.681	2.509
D	5.5378	2.583
E	5.1254	2.7492

Chapter 6 Conclusions and Future Work

6.1 Conclusions

Section 6.1.1 discusses the accuracies with respect to the plant species. Section 6.1.2 discusses and compares the accuracies with respect to the different neural network configurations. Finally, Section 6.1.2 discusses the load and computation times of the mobile application.

6.1.1 Leaf Species Accuracies

There was a large amount of variance in accuracies of individual classes. Begonia Brevirimosa was recognized correctly 95.3% of the time while Eugenia Uniflora was only recognized correctly 19.3% of the time. The high accuracy of Begonia Brevirimosa's leaves were likely caused by a combination of the uniqueness of its leaves and that the images in the dataset were all in focus with similar lighting. A comparison of three of the images of Begonia Brevirimosa's leaves can be seen in Figure 6.1.

Figure 6.1: Begonia Brevirimosa

Eugenia Uniflora had the least top-1 accuracy of any of the classes. One reason for this is that there is a lot of variance in its leaves in general – there are significant color and even shape differences. Another reason is that the lighting of the photos taken differed, and not all of the pictures were in focus. A comparison of three of the images of Eugenia Uniflora's leaves can be seen in Figure 6.2.

Figure 6.2: Eugenia Uniflora

It can be seen in Figures 5.16-5.24 that the percent likelihoods for the incorrect classifications of leaves were significantly lower in general than those of the correct classifications in Figures 5.1-5.15. The majority of the correct classifications in Figures 5.1-5.15 had percent likelihoods greater than 90%. The incorrect classifications were generally around 60% or less. This shows that the neural network simply does not recognize the leaves being misclassified and could be significantly improved with a much larger dataset that covered more variation in the leaves.

6.1.2 Neural Network Accuracies

In general, deeper networks tended to have less overfitting of the images than shallower ones. Networks D and E, the two deepest networks, had the highest top-3 test

accuracies of any of the networks. Network A, the shallowest network, had the lowest top-3 test accuracy of the networks. Deeper networks allow for more complex features to be searched for in the images, so the deeper networks can learn better overall representations of the classes.

In addition, having two fully connected layers before the softmax classifier performed significantly better than only having one fully connected layer. This is likely due to the dropout on the fully connected layers. Dropout in fully connected layers tends to outperform dropout on convolutional layers.

Lastly, having the extra convolution layer in E did not seem to make a significant difference. In fact, it slightly reduced the network's accuracy from network D. The reasons for this are unclear. It could be that additional iterations of training would make up for the slight difference. It could also be that with an image as small as the one being used, D is as deep of a network as is useful.

6.1.3 Mobile Application Load and Computation Times

As can be seen in Table 5-3, the load time of the networks had much more variance than the compute times. The difference in compute times from network A to E was less than 0.3 seconds. In addition, each network's compute times are completely reasonable for a mobile classification application. All of the times for calculation were less than three seconds.

The load times seem to be the major factor in the feasibility of this mobile application. Networks A and B had completely unreasonable load times – both over 21 seconds. This seems to be caused by the size of the weight matrix of the first fully connected layer. Since convolution layers are sparsely connected, they require far fewer weights to

process the same size input. The input to A and B's first fully connected layer was 39x29x16, where C and D had input sizes of 14x19x16. This required 3,619,200 doubles for the weight matrix of the first fully connected layer of A and B, while C and D only had 851,200 doubles for that layer. Each convolution layer only required 144 doubles for their weight matrix. Networks C, D, and E all had reasonable load times of just over 5 seconds.

It appears that additional convolution and pooling layers will reduce the load time of the network by reducing the number of connections of the fully connected layers. However, the convolution operation is expensive, so having more convolution layers will increase the calculation time.

6.2 Future Work

In this research, a mobile leaf classification application was designed using different depths of convolutional neural networks. This application was trained to distinguish between 15 species of leaves, and a top three accuracy of 81.6% was obtained.

The best improvement to this research would be to get a significantly larger dataset. Getting other species of leaves would allow the application to be more useful in that it could recognize more plants. Getting more images of the same leaves would give more variance in the training set, causing the application to learn better representations of the classes and get more accuracy.

In addition, since this research has diverged from needing an image of the leaf segmented from background, it would be possible to transition this from a leaf classification application to a plant classifier. Since the application would most likely be used to distinguish between species of plant by taking a picture of the leaf, the application may be able to learn a representation of the entire plant better than just a close up picture of the leaf.

It would also be possible to train the network to learn representations of pictures of both the entire plant and just the leaf for more options in using the application.

Better accuracy may also be obtained by modifying the network to average results over several images. Similar to how dropout trains several different networks to give improved accuracy, computation over several images should increase the accuracy of the classifier in practice.

In addition, several other configurations could be considered for the application. As of now, the network starts with an 80x60 image, but a larger image would provide more information and potentially give better results at the cost of a longer execution time. The number and size of features for the convolution layers, the pooling dimension, and the output sizes of the fully connected layers could also be modified to examine more networks. Finally, with larger image sizes, deeper networks could be examined as well. With these deeper networks, if the load time gets to be too much for a mobile application, a client-server architecture could be examined to be used when the device has an internet connection. This would allow for much more complex networks, while still maintaining the option of using a less complex network for classification if there is no internet connection.

Bibliography

[1] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez and J. V. B. Soares, Leafsnap: A Computer Vision System for Automatic Plant Species Identification. In *12th European Conference on Computer Vision*, Florence, Italy, 2012.

[2] K. He, X. Zhang, S. Ren and J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR,* vol. abs/1502.01852, 2015.

[3] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors. *CoRR,* vol. abs/1207.0580, 2012.

[4] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks. *NIPS*, 2012.

[5] H.-N. Qi and J.-G. Yang, Saw tooth feature extraction of leaf edge based on support vector machine. *International Conference on Machine Learning and Cybernetics,* vol. 5, pp. 3039-3044, 2003.

[6] Y. Li, Q. Zhu, Y. Cao and C. Wang, A Leaf Veing Extraction Method Based on Snakes Technique. In *International Conference on Neural Networks and Brain*, Beijing, 2005.

[7] H. Fu and Z. Chi, Combined thresholding and neural network approach for vein pattern extraction from leaf images. *IEEE Proceedings. Vision, Image and Signal Processing,* vol. 153, no. 6, pp. 881-892, 2006.

[8] Y. Nam, E. Hwang and K. Byeon, ELIS: An Efficient Leaf Image Retrieval System. In *Third International Conference on Advances in Pattern Recognition*, Bath, UK, 2005.

[9] X. Gu, J.-X. Du and X.-F. Wang, Leaf Recognition Based on the Combination of Wavelet Transform and Gaussian Interpolation. In *Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I*, Hefei, China, 2005.

[10] J.-X. Du, X.-F. Wang and G.-J. Zhang, Leaf shape based plant species recognition. *Applied Mathematics and Computation,* vol. 185, no. 2, pp. 883-893, 2007.

[11] B. Heymans, A neural network for Opuntia leaf-form recognition. *International Joint Conference on Neural Networks,* vol. 3, pp. 2116-2121, 1991.

[12] D. Warren, Automated leaf shape description for variety testing in chrysanthemums. In *Sixth Interational Converence on Image Processing and Its Applications*, 1997.

[13] T. Saitoh and T. Kaneko, Automatic recognition of wild flowers. *International Conference on Pattern Recognition,* vol. 2, pp. 507-510, 2000.

[14] M. Zhenjiang, M.-H. Gandelin and Y. Baozong, An OOPR-based rose variety recognition system. *Engineering Applications of Artificial Intelligence,* vol. 19, no. 5, pp. 78-101, 2006.

[15] B. Shrestha, *Classification of plants using images of their leaves,* Boone, NC: Appalachian State University, 2010.

[16] H. Lee, R. Grosse, R. Ranganath and A. Y. Ng, Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks. *Communications of the ACM,* vol. 54, pp. 95-103, 2011.

[17] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean and A. Y. Ng, Building High-level Features Using Large Scale Unsupervised Learning. In *Proceedings of teh Twenty-Ninth International Conference on Machine Learning*, 2012.

[18] K. Mehrotra, S. Jetley, A. Deshmukh and S. Belhe, Unconstrained handwritten Devangari character recognition using convolutional neural networks. In *Proceedings of the 4th International Workshop on Multilingual OCR,* New York, NY, USA, 2013.

[19] P. Y. Simard, D. Steinkraus and J. C. Platt, Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Proceedings of the Seventh International Conference on Document Analaysis and Recognition*, Edinburgh, UK, 2003.

[20] R. M. Bell and Y. Koren, Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter,* vol. 9, no. 2, pp. 75-79, 2007.

[21] X. Glorot, A. Bordes and Y. Bengio, Deep Sparse Rectifier Neural Networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011.

[22] X. Pitkow and M. Meister, Decorrelation and Efficient Coding by Retinal Ganglion Cells. *Nature Neuroscience,* pp. 628-635, 2012.

[23] A. J. Bell and T. J. Sejnowski, The "Independent Components" of Natural Scenes are Edge Filters. *Vision Res.,* vol. 37, no. 23, pp. 3327-3338, 1997.

[24] A. Ng, Ngiam Jiquan, C. Y. Foo, Y. Mai, C. Suen, A. Coates, A. Maas, A. Hannun, B. Huval, T. Wang and S. Tandon, UFLDL Tutorial. [Online]. Available: http://deeplearning.stanford.edu/tutorial/. [Accessed 19 April 2015].

[25] B. T. Polyak, Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 6, pp. 1-17, 1964.

[26] I. Sutskever, J. Martens, G. Dahl and G. Hinton, On the Importance of initialization and momentum in deep learning. *ICML*, pp. 1139-1147, 2013.

[27] X. Glorot, A. Bordes and Y. Bengio, Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011.

[28] L. Breiman, Random Forests. *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

Appendix

All of the programs developed for this thesis are included on the enclosed DVD. Following is a list of all of the files on the DVD:

1. LeafApp: Project containing the mobile application developed for this thesis
2. Leaves: Directory containing the dataset collected for this thesis
3. NeuralNetwork: Neural network library developed for this thesis
4. CompiledResults.xlsx: Graphs and data points collected from the training and testing of the different neural networks
5. ParseRes.java: Parses the output of the training and testing of a neural network and stores the data in excel files.
6. ResultA.txt: The output of training and testing of neural network configuration A
7. ResultB.txt: The output of training and testing of neural network configuration B
8. ResultC.txt: The output of training and testing of neural network configuration C
9. ResultD.txt: The output of training and testing of neural network configuration D
10. ResultE.txt: The output of training and testing of neural network configuration E

Vita

Tim Jassmann was born in 1991 in Houston, Texas. After just a few years living in Lake Jackson, Texas, he moved to grow up in Raleigh, North Carolina. After high school, he thought he wanted to major in music, but after not getting accepted into any of the programs that interested him he decided to try out Computer Science. His interest in the field grew so much that after he graduated with his Bachelor of Science from Appalachian State University, he decided to continue to pursue his Master of Science from ASU as well. During his Masters, he also worked as a research assistant to Dr. Rahman Tashakkori. In July 2015, he started working for Amazon's Digital Music Web Player team as a Software Development Engineer.