

31/9/2019



UNIVERSITY OF
KWAZULU-NATALTM
INYUVESI
YAKWAZULU-NATALI

Discipline of Electrical, Electronic and Computer
Engineering

Leaf Features Based Plant Identification using Deep Learning

Phase 3 Report: System Design and Initial testing



Shane Dewar - 214502730

SUPERVISOR: DR. J. TAPAMO

Abstract

The following report documents the design and implementation of an automated plant species identification software using a deep learning Convolution Neural Network (CNN).

Plant identification proves to be a very complex and challenging task as there are an estimated 391 000 different species of plants, with many of them having very similar leaf shapes and patterns. It has however, been discovered that each species of tree's leaves are unique; like a human fingerprint; and can therefore be used to conclusively categorize plants by their leaves.

Following the initial research, A Wide Residual Neural Network (ResNet) was determined as the most effective solution. This ResNet CNN will perform as the backbone plant identification algorithm. Testing shows that the implemented solution can classify 76 different species of plants with an accuracy of 98.8%. Data augmentation and transfer learning played a key role in successfully training the network on the small dataset. The Project's progression from problem identification, through to system design and testing are detailed in this report. The feasibility study along with final tests performed in this report indicate that the proposed solution effectively meets the project aim.

Table of Contents

Abstract.....	1
1. Introduction	5
2. Literature Review	5
2.1 Plant identification methods	5
2.2 Neural Networks	5
2.3 Convolutional Neural Networks (CNN`s)	6
2.3.1 Convolutional Layer	7
2.3.2 Pooling Layer	7
2.3.3 Activation Functions	8
2.3.4 Dropout Regularization.....	9
2.4 CNN Architectures.....	9
2.4.1 AlexNet (2012)	9
2.4.2 GoogLeNet (2014)	10
2.4.3 ResNet (2015).....	10
2.5 Related Plant identification work	11
3. Problem Statement	12
3.1 System Specifications.....	12
3.1.1 Functional Specifications	12
3.1.2 Non-Functional Specifications	12
3.1.3 Constraints	12
3.2 Pre-requisites	13
3.2.1 Knowledge.....	13
3.2.2 Hardware	13
3.2.3 Software	13
4. Solution Description.....	14
4.1 System block diagram	14
4.2 CNN Structure	14
4.2.1 Spacial Dimensions	16
4.2.2 Activation maps	16
4.2.3 Shortcut Convolution	16
4.2.4 Number of convolutions	16
4.2.5 Full Network Design	16
4.3 Image Pre-processing.....	16
4.4 Use-Case Diagram	17
4.5 performance Metrics	17

4.5.1 The Loss Function.....	17
4.5.2 Accuracy Curve.....	18
4.5.3 Training Time	19
4.5.4 System Memory and Execution Time	19
4.6 Added Functionality	19
4.6.1 Plant Information Database	19
4.6.2 Mobile Application.....	19
5. Implementation	20
5.1 Neural Network.....	20
5.2 User Interface (UI).....	21
5.3 Mobile Application.....	22
5.3.1 Server Implementation	23
5.3.2 Mobile App implementation.....	23
5.4 Plant Classification Function	24
5.5 Plant Dataset.....	25
5.6 Plant Database	26
5.7 Transfer Learning	26
6. System Testing	27
6.2 The Wide-ResNet Tests.....	27
6.2.1 The Loss function	27
6.2.2 Accuracy Graph	28
6.2.3 Confusion Matrix.....	29
6.2.3 Network Accuracy vs. Number of classes	30
6.2.4 Training Time	31
6.2.5 Model Size.....	31
6.3 Practical Test	32
7. Feasibility Analysis	34
7.1 Market Feasibility.....	34
7.2 Budget Feasibility.....	34
7.3 Technological Feasibility	34
7.4 Time allocation.....	34
7.4.1 Gantt Chart.....	35
8.Future Work.....	35
9.Conclusion.....	36
References	37
Appendix	40

Appendix A – ResNet Full Block Diagram.....	40
Appendix B – User Interface	41
Appendix C	42
Appendix C1 – class accuracy.....	42
Appendix C2 – training values.....	42
Appendix C3 – Model Summary	43

1. Introduction

The aim of this project; to develop an automated plant identification system; arises from the fact that manual identification of plants is extremely time consuming and is prone to errors. Plant identification proves to be a very complex and challenging task as there are an estimated 391 000 different species of plants (Dasgupta, 2016) with many of them having very similar leaf shapes and patterns. Botanists (Those who study plants) spend years mastering the challenge and the main goal of the system is to assist them with easier and more efficient classification of plants. Other possible applications of the system are edible plant identification which can be used in survival situations to determine which plants are safe to eat (Wibowo, 2018) (Sujaro, 2014), as well as in medicinal plant research applications (Harsani, 2017).

Convolutional neural networks are currently the state-of-the-art deep learning method, having won all majors image classification challenges in the past 7 years (Das, 2017). A well developed and trained CNN will allow highly accurate and efficient classification of plants which is a requirement for this system. Techniques such as image pre-processing, dropout and transfer learning will be used to further enhance the accuracy of the network by preventing overfitting of training data.

2. Literature Review

This section examines the pre-requisite theoretical knowledge and materials required to complete this project, as well as investigating related work that has been attained in this field. This research is necessary in order to formulate the best possible implementation solution.

2.1 Plant identification methods

There are currently many methods used to classify plants, with the most prominent being: manual human identification, plant cell biology, phytochemistry (Wang, 2014) and Neural network image processing. Human identification is a slow process and is error prone unless conducted by an individual that has trained many years in the field. Biological methods of identification are extremely accurate (effectively 100% accuracy) (Newmaster, 2006) however once again, it is a very slow process.

Neural Networks provide the perfect compromise, offering high speed classification, while maintaining very accurate results. (Wick, 2017) Achieved plant “recognition rates above 99%” with a 9-layer CNN.

2.2 Neural Networks

A Neural Network is a deep learning algorithm derived analogously from the human brain, which consists of many connected “neurons”, each of which contain learnable biases and weights that determine its activation. Each neuron activation corresponds to different characteristics of the input image. The Mathematical formula to calculate each neuron’s weight is:

$$S_n = \text{Max}(0, \sum_{i=0}^n W_n * a_i + B)$$

where:

W_n = The n th neuron’s weight value

$a_i = \text{The } i\text{th input into the neuron}$

$B = \text{The neuron's Bias value}$

Figure 1 below shows a basic depiction of the link between a human brain neuron and the mathematical model of a NN “neuron”

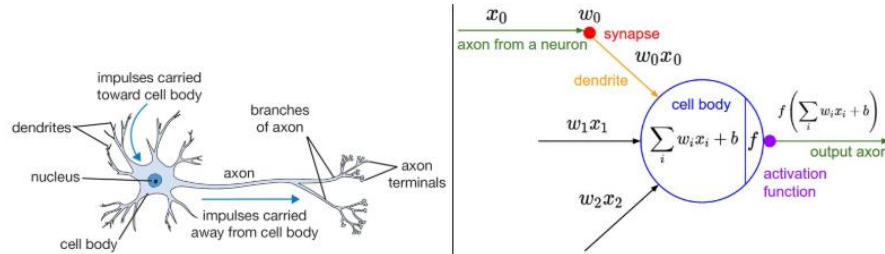


Figure 1: Brain neuron vs. Mathematical model of a neuron (Karpathy, 2017)

Remarkably, after being trained on a known set of images, the network can accurately classify testing images that haven't been seen before. Figure 2 below shows the structure of a very simple 2-layer NN.

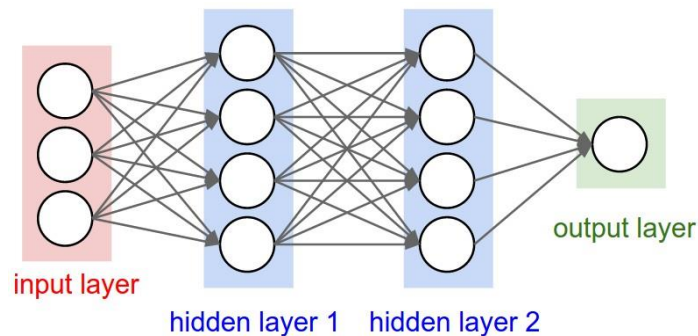


Figure 2: 2-layer NN structure (Karpathy, 2017)

2.3 Convolutional Neural Networks (CNN's)

A CNN is a special kind of neural network that is commonly used in image classification problems. CNN's contain special layers called convolutional layers that apply various filters to the input image in order to extract features and patterns within the image. By feeding forward the features through many convolutional layers, the network can identify and categorize very intricate images. Convolutional networks have the advantage over general NN in that they are able to “successfully capture the spacial and temporal dependencies of the image” (Das, 2017)

A typical CNN Structure will contain repeated layers of convolution, activation functions and a pooling layer. Finally, a fully connected layer performs the final classification. An example 2-conv layer network is shown in figure 3 below:

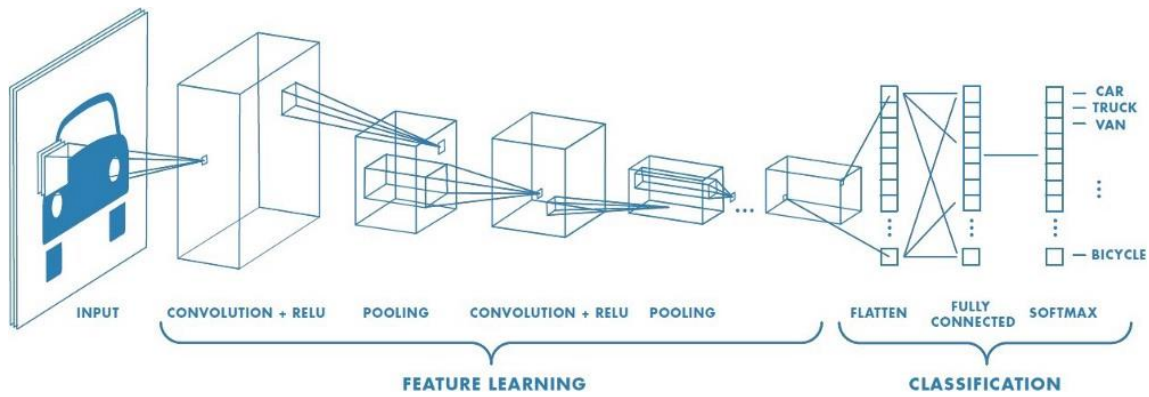


Figure 3: Basic CNN Structure (Das, 2017)

The image above simply depicts a basic CNN structure. There is no set structure for the best CNN and a lot of research is currently being developed in the field. Modern benchmark architectures are far more complex. An example of this is ResNet by (He et al, 2015) which is a 152-layer deep network that won the ILSVRC 2015 ImageNet challenge.

2.3.1 Convolutional Layer

The Convolutional layer is the part of the network that is responsible for extracting features from the input image. It contains several filters each of which have learnable parameters. The size of the filters is predetermined and generally consist of 1×1 , 2×2 or 3×3 size matrices. Each filter is convolved over the input image by performing a matrix multiplication at every point and shifting the filter by a pre-set number called the **stride**. This produces an activation map of size: $\frac{n-m}{Stride} + 1$

Where n = input image size, m = Filter Size, and assuming n and m are square matrices.

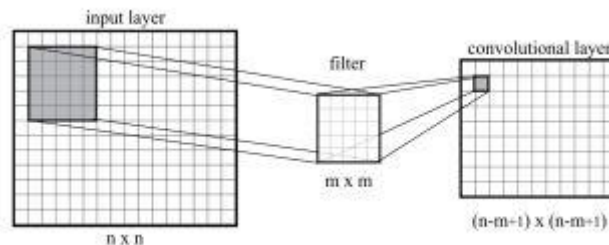


Figure 4: Convolution process (Wang, 2014)

It can be noted from figure 4 above that the Activation map has smaller dimensions than the input image, therefore zero padding of the input image is often incorporated to preserve spacial dimensions.

2.3.2 Pooling Layer

The pooling layer is responsible for reducing spacial dimensions by down sampling the Activation maps. This in turn reduces computation power required in the subsequent layers. Another advantage of pooling is that it extracts dominant features and eliminates possible noise components, thus increasing the images noise suppression and rotational/positional invariance.

There are 2 common pooling methods, namely: Max Pooling and Average Pooling. Whereby the Maximum or Average value respectively in an N*N sample of the image (generally 2*2 or 3*3) extracted. Figure 5 illustrates the difference between Max and Average Pooling.

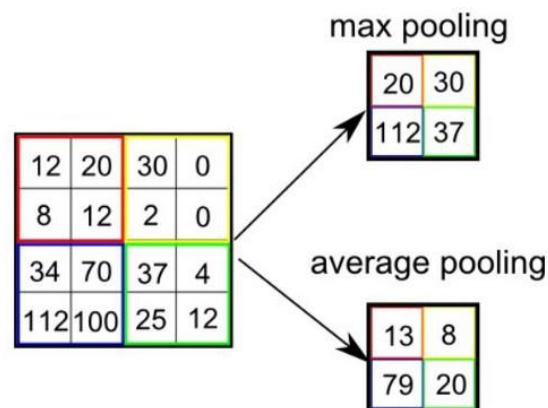


Figure 5: Pooling functions (Saha, 2018)

Studies show that Max Pooling performs far better than average pooling in most cases (Saha, 2018) and hence will be used in the development of this system

2.3.3 Activation Functions

Activation functions are non-linear functions which are incorporated after every Conv/Pooling layer in order to normalize the Output Maps. This prevents extremely large or small values being obtained. Common activation functions used are Sigmoid, tanh and Rectified Linear Unit (ReLU)

Sigmoid: $y = (1 + e^{-x})^{-1}$

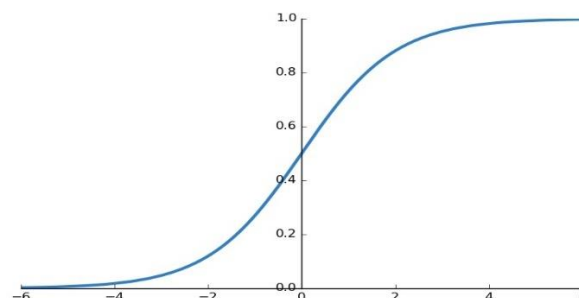


Figure 6: Sigmoid Function

Sigmoid functions suffer from the vanishing gradient problem which occurs when small or large inputs are saturated at 0 or 1 respectively. This causes the gradient at subsequent layers to approach zero and “vanish”. Vanishing gradients create slow optimisation during the network training (Wang, 2014).

ReLU: $y = \text{Max}(0, x)$

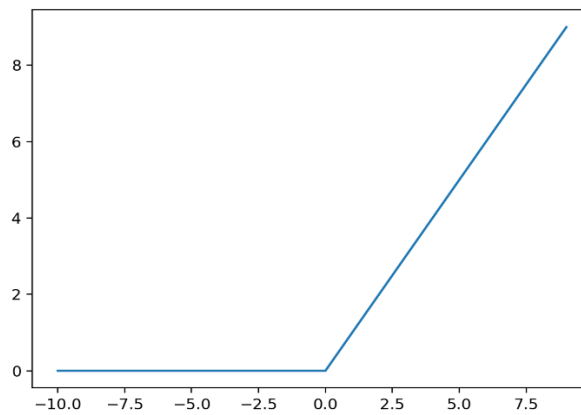


Figure 7: ReLU function

It can be seen from figure 7 that ReLU functions eliminate the vanishing gradient problem at high input values and thus act as a considerably better activation function in practice than sigmoid.

2.3.4 Dropout Regularization

Dropout is a Neural network optimization technique that prevents the network from overfitting on training data. During training, random neurons are dropped (their activations set to zero) and hence those neurons do not contribute during feed forward or backpropagation. The effect of dropout is essentially training the network on various architectures in parallel that are a subset of the full architecture implemented (Brownlee, 2018). Typical dropout probabilities are between 0 - 0.5.

2.4 CNN Architectures

There are an infinite number of ways a CNN network can be designed and there is currently a lot of research being conducted in this field. The following three architectures are variations of the CNN network that have performed the best in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

2.4.1 AlexNet (2012)

AlexNet is an 8-layer deep (5 Convolutional layers) that contains just over 60M parameters. This was the first major deep CNN network that was used in image classification. The network won ILSVRC'12 with a top-5 error of 16% (Das, 2017) (Karim, 2018). The large number of parameters mean that this network requires significant resources and training can be slow.

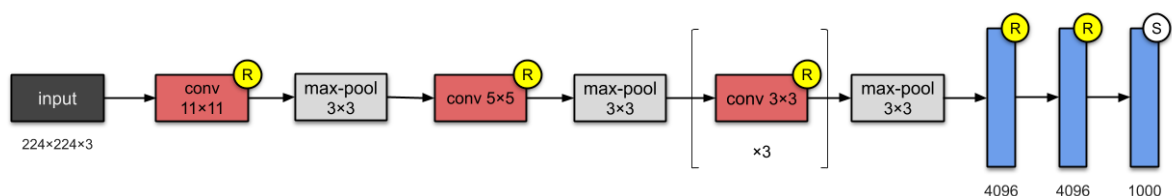


Figure 8: AlexNet Block Diagram (Karim, 2018)

2.4.2 GoogLeNet (2014)

This is a 22-layer network containing 4M parameters. The system is primarily made up of convolution inception modules (depicted in figure 9). The inception modules allow the network to considerably reduce the number of parameters whilst improving classification accuracy. “GoogLeNet achieved a top 5 error rate of 6.67% in the ILSVRC`14” (Das, 2017). The main feature of this network is its relatively low number of parameters. This allows the network to be trained in considerably less time and less resource requirements.

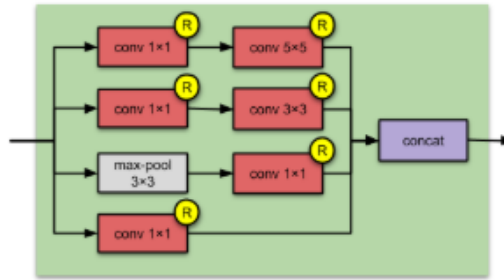


Figure 9: Inception Module (Karim, 2018)

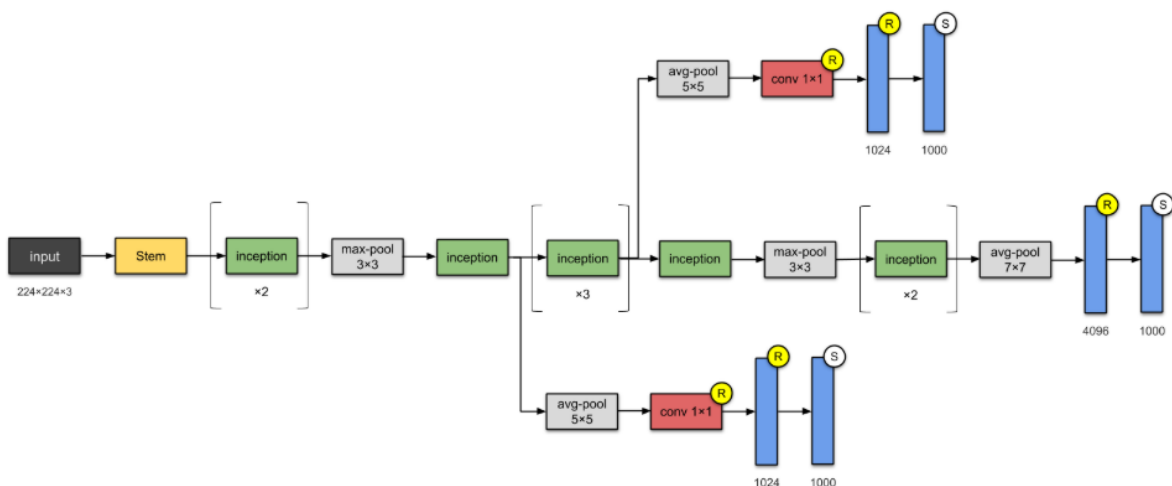


Figure 10: GoogLeNet Block diagram (Karim, 2018)

2.4.3 ResNet (2015)

The most significant hindrance when developing very deep CNN networks is the vanishing gradient problem. This occurs when the loss function's gradient approaches zero and hence causes slow optimisation during the network training (Wang, 2014). The more layers there are in a network, the bigger the vanishing gradient problem becomes.

ResNet solves this problem with the implementation of Residual blocks (Wang, 2015). Every 2 (or 3) convolutions, an identity shortcut skips these layers and is added to the output. Many residual blocks are grouped together to form the entire network. This architecture allows extremely deep (up to 1000) layer networks to be developed that converge effectively (He et al, 2015).

The structure of a resnet block as well as the full ResNet Architecture are displayed in figures 11 and 12 respectively:

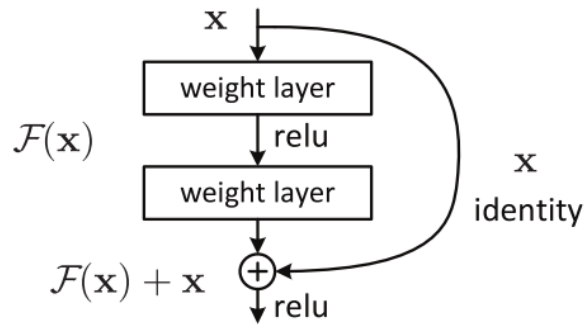


Figure 11: Residual Block (Morris, 2018)

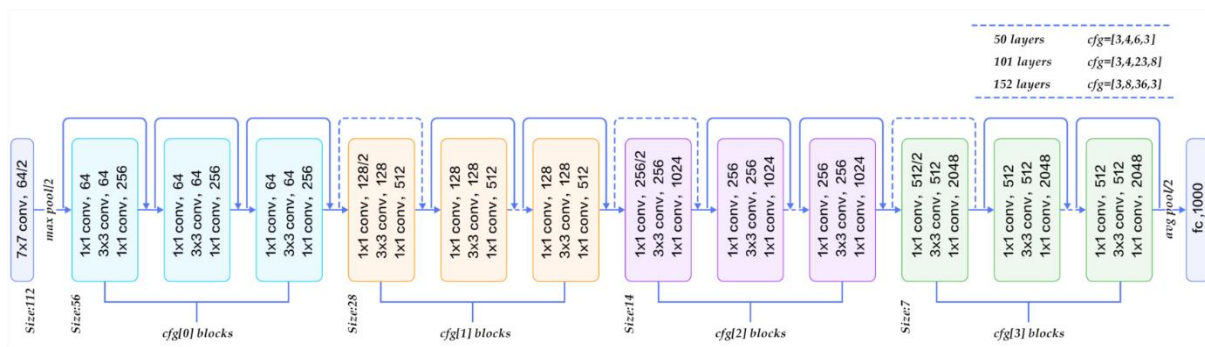


Figure 12: ResNet Structure (Das, 2017)

A 152 layer ResNet network won ILSVRC`15 with an error rate of less than 4%. This is better than human level performance (Das, 2017).

An advancement of this architecture is the wide ResNet which implements less layers, but the number of convolutional filters is increased (i.e. The depth of the activation maps).

(Zagoruyko & Komodakis, 2017) demonstrated that wide resnets are more accurate and converge faster than their deep counterparts.

2.5 Related Plant identification work

There have been many research papers and attempts to develop automated plant identification models in recent years. Before the wide-spread development of CNN's, most of the research was in feature extraction of the leaf images such as the shape, colour histogram, texture and vein outlines (Jeon & Rhee, 2017). The highest accuracy that was achieved from machine learning using these features was 90% (Du, Wang, & Zhang, 2007).

The implementation of CNN's, however brought vast improvements as it eliminated the need for the time-consuming feature extraction (Hedjazi, 2017). The most common and biggest challenge in leaf identification using CNN is the lack of a large public dataset of leaf images. The two most popular datasets are Flavia (Wu, 2007) and Foliage (Kadir & Nugroho, 2011) which have 60 and 100 images per class respectively. This places a large importance on image augmentation to prevent the network from overfitting.

By implementing variations of these augmentation techniques, (Wick & Puppe, 2017) achieved 99% recognition accuracy on the above-mentioned datasets using a nine-layer Conv network.

(Jeon & Rhee, 2017) Achieved a test accuracy of 99.8% using a modified GoogLeNet CNN model.

The most popular leaf identification software that is commercially available is Leafsnap (Kumar & Biswas, A Computer Vision System for Automatic Plant Species Identification, 2012). The software is available as a mobile application and can identify 185 different species of trees.

The final implementation of this project will be trained and tested on the Flavia dataset to compare the results obtained against those achieved by others in the related work.

3. Problem Statement

A System is to be developed that will automate the process of plant identification, to assist botanists with plant species recognition. The software will identify plants by analysing pictures of their leaves and returning the result to the user. The core classification algorithm is to be implemented using a deep learning network that will produce the most effective solution in terms of accuracy and efficiency. The deep learning algorithm will allow the system to develop and improve by way of additional plant species being loaded into the software.

3.1 System Specifications

3.1.1 Functional Specifications

- Develop a Convolutional Neural Network as the core classification algorithm
- Design the CNN hyperparameters (learning rate, Architecture, Activation functions etc.) to ensure the network converges.
- The system must allow the user to upload an image of a leaf and return the classification result back to the user.
- The system should automatically pre-process the input image before classification (resizing, data augmentation etc.)

3.1.2 Non-Functional Specifications

- Robustness – The system should function without fail under different circumstances
- Maintainability – The software should be maintainable after deployment.
- User Friendliness – The User Interface should be clean, appealing and easy to navigate.
- Accuracy – The system must have an acceptably high identification accuracy. The initial aim is to achieve 95% or higher accuracy.

3.1.3 Constraints

- Time – The system must be developed and fully functional within the time allocated.
- Budget – The System must be developed within the allocated R1700 budget.
- Hardware – The PC running the software has limited memory and processing capabilities.
- The accuracy of the system will be affected by the conditions under which the leaf pictures are captured. The constraint variables which affect this are: Lighting, weather (wet or dry), age of the plant, leaf damage.
- To improve identification accuracy, leaf images should be captured in front of a white background.

- User's Technical ability – target market for the application is botanists who may not have a high technical ability. The application must therefore be easily operated without requiring any knowledge of the backend processes (such as neural networks).

3.2 Pre-requisites

The technical requirements and pre-requisite information needed to develop and implement the required solution are detailed below.

3.2.1 Knowledge

A detailed knowledge of neural networks; in particular; CNN's is required, as well as understanding the biological differences in plant leaves to effectively categorize them. The theory behind this was researched and highlighted in section 2.

The system will be developed in python programming language; therefore, a good understanding of python is required to implement the software. Python was chosen as it is currently the most popular language in machine learning and data science (Voskoglou, 2017). The libraries are well developed and documented, allowing for the most efficient implementation of the solution.

Completion of this project requires an in-depth understanding of mathematical concepts, particularly Calculus. This knowledge was acquired in 1st, 2nd and 3rd year maths courses. Software Engineering, Design and analysis of Algorithms, and Artificial Intelligence courses provide the knowledge and experience required to develop the software.

3.2.2 Hardware

A PC with relatively high processing power and memory is required to run the software (8GB RAM and a graphics card). Neural Networks can be highly complex and therefore require more resources than most software. A PC with higher processing power will allow faster training of the network and a deeper network to be developed.

3.2.3 Software

The software used to complete this project is:

- Python
- Pytorch framework
- MATLAB
- PyCharm IDE
- Kivy Python library (User Interface development)
- Apache
- MYSQL and Apache (Added feature)
- Android Studio and Flask (Added feature)

All the above Software is open source and free to download.

Pytorch: Pytorch was chosen as the framework to develop the NN. The two most popular options are TensorFlow and Pytorch, but research indicates that Pytorch is the most suitable framework for this project, as it makes use of dynamic computational graphs as opposed to TensorFlow's static graphs. This is particularly important as the chosen Resnet architecture is dynamic in its design.

Pytorch is therefore more dynamic in its implementation and more suitable for research purposes (Jain, 2018).

4. Solution Description

Following the thorough research performed in section 2, the proposed solution is to develop a 28-layer wide ResNet Convolutional Neural Network. The design takes inspiration from the work of (Zagoruyko & Komodakis, 2017). Dropout Regularization and batch normalization is implemented to improve accuracy and prevent overfitting.

A Simple, yet appealing GUI is developed to improve the software user friendliness. When a leaf image is uploaded, image pre-processing techniques (discussed in section 4.3 below) are applied automatically to the input to improve classification accuracy.

An Android Application is developed to allow user to identify plants “on-the-go”. The application uploads the leaf image to a server, which executes the same classification script as the PC app. The result is then returned to the mobile app.

4.1 System block diagram

The diagram below depicts the overall structure of the proposed system.

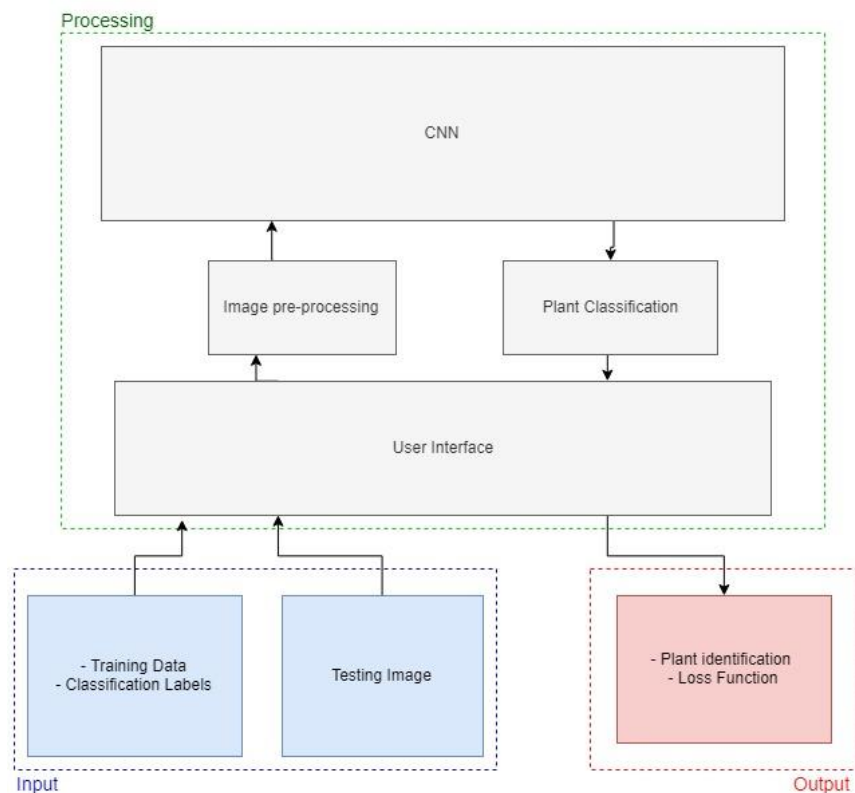


Figure 13: System Block Diagram

4.2 CNN Structure

The implemented neural network that will provide the core image classification is a wide-Resnet CNN (researched in section 2.4.3). The network will consist of 3 groups, with each group containing 3

residual blocks (figure 14 below). Each residual block contains a pair of convolution layers, as well as batch normalization and Relu (Rectifier Linear Units).

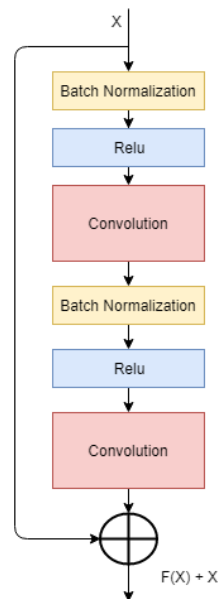


Figure 14: Residual block Structure

The network contains an initial convolution, after which the 3 residual groups. This is followed by an averaging pooling layer, a fully connected and finally the SoftMax categorisation layer. Figures 15 and 16 illustrate the basic structure of the network in table and block diagram form respectively.

Group	Output Size (Width*Height)	Block Structure (Kernel size, depth, no. of blocks)
Conv 1	224*224	{3*3, 16}
Conv group 1	224*224	$\begin{Bmatrix} 3 * 3, 16 * 4 \\ 3 * 3, 16 * 4 \end{Bmatrix} * 4$
Conv group 2	112*112	$\begin{Bmatrix} 3 * 3, 32 * 4 \\ 3 * 3, 32 * 4 \end{Bmatrix} * 4$
Conv group 3	56*56	$\begin{Bmatrix} 3 * 3, 64 * 4 \\ 3 * 3, 64 * 4 \end{Bmatrix} * 4$
Avg Pool	1*1	{8*8}

Figure 15: ResNet Structure Table

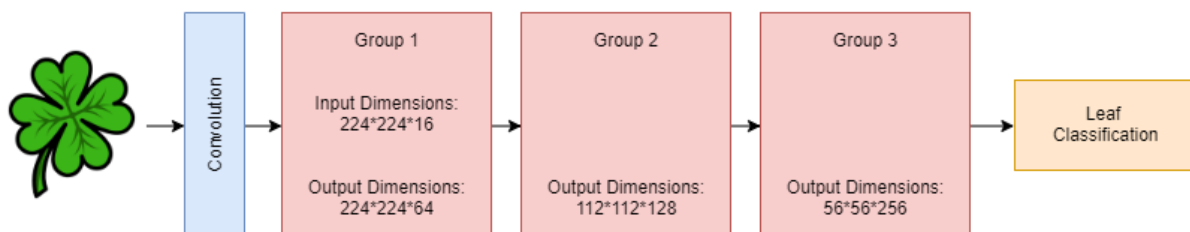


Figure 16: Basic Resnet structure

4.2.1 Spacial Dimensions

Each group halves the spacial dimensions of the activation maps. The reason for this is to reduce the number of parameters in the network and is performed in the first block within the group. This is achieved by setting stride = 2 and padding = 1, using the formula:

$$\text{output size} = \frac{\text{input size} + \text{padding} - \text{filter size}}{\text{stride}} + 1 = \frac{\text{input size} + 1 - 3}{2} + 1 = \frac{\text{input size}}{2}$$

4.2.2 Activation maps

The number of activation channels (the depth of the output maps) are multiplied by 4 in the first group. This is known as the widening factor. After which, each group doubles the number of output activation channels. Once again this is performed in the first block of each group.

4.2.3 Shortcut Convolution

The dimensions of the shortcut connection in the first block of each group will not match the output dimensions due to doubling the output depth in the convolution. To allow the skip connection to be added to the block output, a 3*3 convolution matching the output depth is performed on the skip connection.

4.2.4 Number of convolutions

The total number of convolutions in the network are:

$$\begin{aligned} &1 \text{ initial conv} + 3 \text{ groups} * 3 \text{ blocks per group} * 2 \text{ Convs per block} + 3 \text{ skip Convs} \\ &= 22 \text{ layer deep network} \end{aligned}$$

4.2.5 Full Network Design

A detailed diagram of the entire network is illustrated in Appendix A (figure 46).

4.3 Image Pre-processing

All Images loaded into the system are initially resized to fit the required dimensions of the input layer structure (224*224 pixels). The images will have three depth layers, representing the RGB pixel values.

As the plant leaf datasets are relatively small, the following techniques will be applied to training images in order to increase the networks testing accuracy and prevent overfitting of training data:

- Rotation - The image will be rotated by a random angle up to 30 degrees.
- Flipping – The image will be randomly flipped vertically and horizontally.
- Brightness – The brightness of the image will be randomly scaled by 0.5
- Contrast – The contrast of the image will be randomly scaled by 0.5
- Normalize: The Images are normalized according to mean and standard deviation, to improve training.

Figure 17 below shows the sample code that performs these data augmentations.

```

self.data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(30),
        transforms.ColorJitter(brightness=0.5, contrast=0.5),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),

```

Figure 17: data Augmentation code

4.4 Use-Case Diagram

The Use-Case Diagram describes the Actors (users) and their relationship with the Software interface.

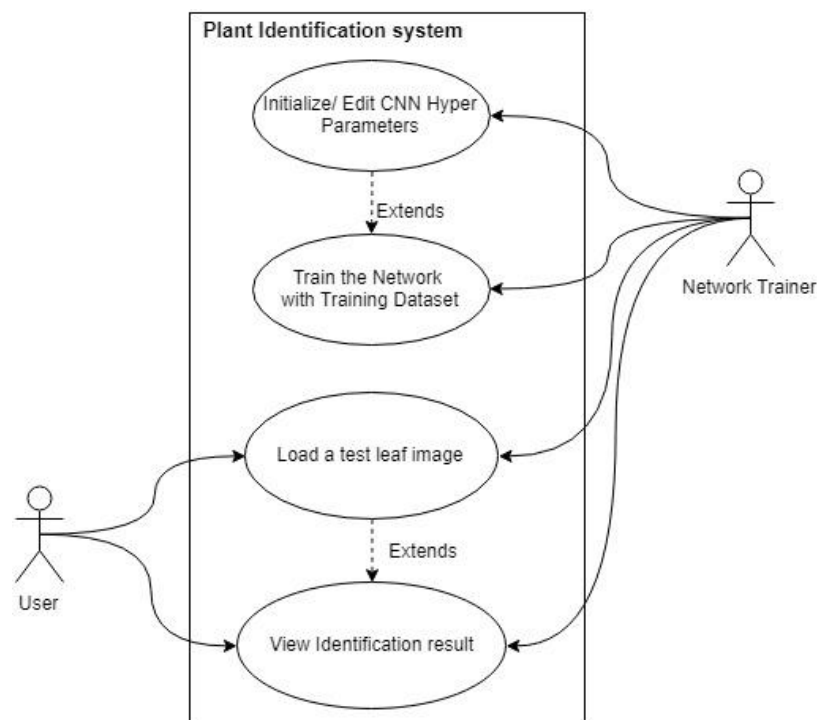


Figure 18: Plant Identification Use Case Diagram

4.5 performance Metrics

The development of system performance metrics is an extremely important part of the design process. These metrics will be used to measure the success of the system throughout development and finally completion. Performance metrics also validate the performance of the system against other similar systems that have been developed, and this will ultimately determine whether the final solution is a success or failure. Detailed below are the metrics that this system will be assessed on.

4.5.1 The Loss Function

The Loss function is a very important metric used in Deep learning systems. It is used as a measure of the inconsistency between the network's predicted value and the actual label. The aim is to attain as small a loss value as possible when performing validation tests.

There are many loss functions that can be implemented, but the 3 most notable functions are:

4.5.1.1 Mean Square Error (MSE) loss function

$$\text{loss}(x, y) = (x - y)^2$$

MSE loss is a simple, effective function but often performs poorly in high dimensional problems (Jha, 2019) as is the case with deep neural networks.

4.5.1.2 Smooth L1 Loss (Huber function)

$$\text{loss}(x, y) = \begin{cases} 0.5(x - y)^2, & \text{if } |x - y| < 1 \\ |x - y| - 0.5, & \text{otherwise} \end{cases}$$

The Huber loss is a very well-rounded loss function and generalizes well with most problems

4.5.1.3 Cross-Entropy Loss

$$\text{loss}(x, y) = - \sum x * \log(y)$$

Cross Entropy determines the probability distributions of the data and is most effective in classification problems (Jha, 2019). The cross-entropy loss will be the most effective loss metric for this problem and hence will be the function that is used to evaluate this system.

4.5.1.4 Loss Curve

The loss curve is a plot of the loss function at set intervals during training and validation. The shape of the curve provides an indication of how well the network is training, and hence if the hyper parameters such as architecture and learning rate have been chosen effectively. The loss curve can also give an indication if the network is beginning to overfit. Figures 18 and 19 below obtained from (Brownlee, 2019) give an illustration of a network with a good loss curve and one which is overfitting.

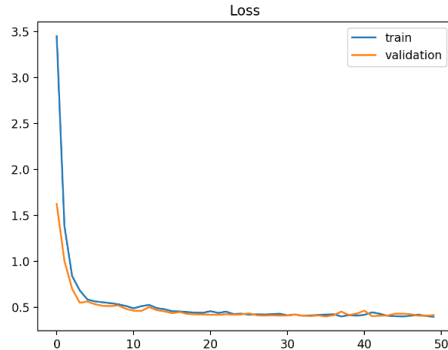


Figure 19: Good loss curve

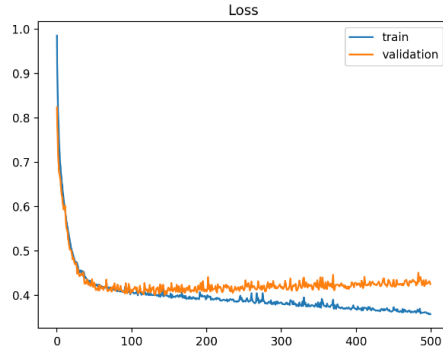


Figure 11: Loss curve indicating overfitting

4.5.2 Accuracy Curve

The Accuracy curve plots the average number of correct classifications at set intervals during training and validation. This curve is similar to the loss curve in that it gives an indication of how well the network is training and will be used in conjunction with the loss curve as a measure of how accurate the CNN software is performing. Figure 20 below shows the expected accuracy curves obtained when there is overfitting and none.

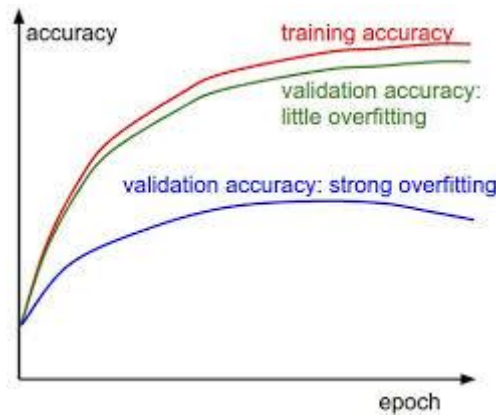


Figure 20: Expected accuracy graph (Karpathy, 2017)

4.5.3 Training Time

Another important metric is the time taken to train the network before it converges to a minimum loss or maximum accuracy. This value is highly dependent on the network architecture, the Learning rate that has been pre-set as well as the hardware that is running the software.

If the network is taking extremely long to converge during training or is converging very quickly and plateauing at a high loss, then this is an indication of the learning rate being set too small or high respectively.

4.5.4 System Memory and Execution Time

Finally, the program memory and average program execution time will be measured for the final software implementation. These metrics are a user satisfaction measure, which is ultimately the most important aspect, as the system is being developed for them.

A large program memory means that most users will not have enough space to install the program and long program execution time means the software will be slow to respond and hence not user friendly.

4.6 Added Functionality

The following are two extra specifications that are added to the system.

4.6.1 Plant Information Database

A database is set up within the system that will display important information about the plant whenever the system identifies a leaf image that the user has uploaded. Examples of this information is the plant's scientific name, if it bears fruit or flowers, if the plant is poisonous or if the plant has any medicinal uses.

4.6.2 Mobile Application

The development of a mobile application would allow users to identify plants easily and on the go. Mobile applications lack processing power and therefore it was determined that uploading the image to a server to classify the image would be the most efficient solution.

5. Implementation

5.1 Neural Network

The network is implemented using Pytorch. The code is developed in a modular approach with function re-use, efficiency and maintainability at the forefront of the design process. A generalized Residual block function is developed and repeatedly called in the Wide-Resnet class to form groups. Multiple groups are created and sequentially executed to form the entire network. Figures 21 and 22 below illustrate the basic block code and part of the Wide-Resnet class.

```
class BasicBlock(nn.Module):
    # input,output channels , stride , dropout
    def __init__(self, inf, outf, stride, drop):
        super().__init__()
        self.bn1 = nn.BatchNorm2d(inf)
        self.conv1 = nn.Conv2d(inf, outf, kernel_size=3, padding=1,
                               stride=stride, bias=False)

        # Dropout Regularization
        self.drop = nn.Dropout(drop, inplace=True)
        self.bn2 = nn.BatchNorm2d(outf)
        self.conv2 = nn.Conv2d(outf, outf, kernel_size=3, padding=1,
                               stride=1, bias=False)

        if inf == outf:
            self.shortcut = lambda x: x
        else:
            # if input and output channels don't match, we cant add x + F(x)
            # therefore, we apply convolution to x to double the channels
            self.shortcut = nn.Sequential(
                nn.BatchNorm2d(inf), nn.ReLU(inplace=True),
                nn.Conv2d(inf, outf, 3, padding=1, stride=stride, bias=False))

    def forward(self, x):
        x2 = self.conv1(F.relu(self.bn1(x)))
        x2 = self.drop(x2)
        x2 = self.conv2(F.relu(self.bn2(x2)))
        r = self.shortcut(x)
        return x2.add_(r)
```

Figure 21: Residual Block Code

```
class WideResNet(nn.Module):
    def __init__(self, n_grps, N, k=1, drop=0.3, first_width=16):
        super().__init__()
        layers = [nn.Conv2d(3, first_width, kernel_size=3, padding=1, bias=False)]
        # Double feature depth at each group, after the first
        widths = [first_width]
        for grp in range(n_grps):
            widths.append(first_width*(2**grp)*k)
        for grp in range(n_grps):
            layers += self.make_group(N, widths[grp], widths[grp+1],
                                     (1 if grp == 0 else 2), drop)
        layers += [nn.BatchNorm2d(widths[-1]), nn.ReLU(inplace=True),
                  nn.AdaptiveAvgPool2d(1), Flatten(),
                  nn.Linear(widths[-1], 10)]
        self.features = nn.Sequential(*layers)

    def make_group(self, N, inf, outf, stride, drop):
        group = list()
        for i in range(N):
            blk = BasicBlock(inf=(inf if i == 0 else outf), outf=outf,
                             stride=(stride if i == 0 else 1), drop=drop)
            group.append(blk)
        return group

    def forward(self, x):
        return self.features(x)
```

Figure 22: Wide ResNet Code

The code shown in the figures above, demonstrates the Blueprint implementation of the modules. This allows seamless updating of network parameters and the network architecture. The software will be maintainable in the future, as well as structured for efficient debugging should bugs arise.

5.2 User Interface (UI)

The target market for this system will be botanists whose main objective is to quickly and efficiently identify plants using the software. These end users may not have an in-depth technical understanding and hence a clean, professional and appealing UI was developed for their ease of use. Kivy python library is used to develop the GUI as it is cross-platform and offers the best tools to develop the envisioned application design. Figures 23 and 24 illustrate examples of the implemented UI. Other UI screenshots can be found in appendix B.

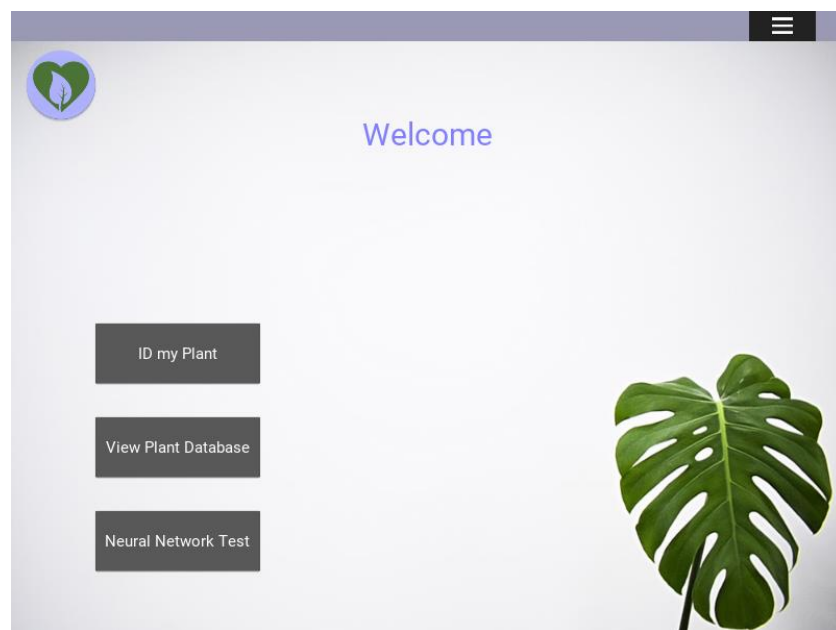


Figure 23: Home Page UI

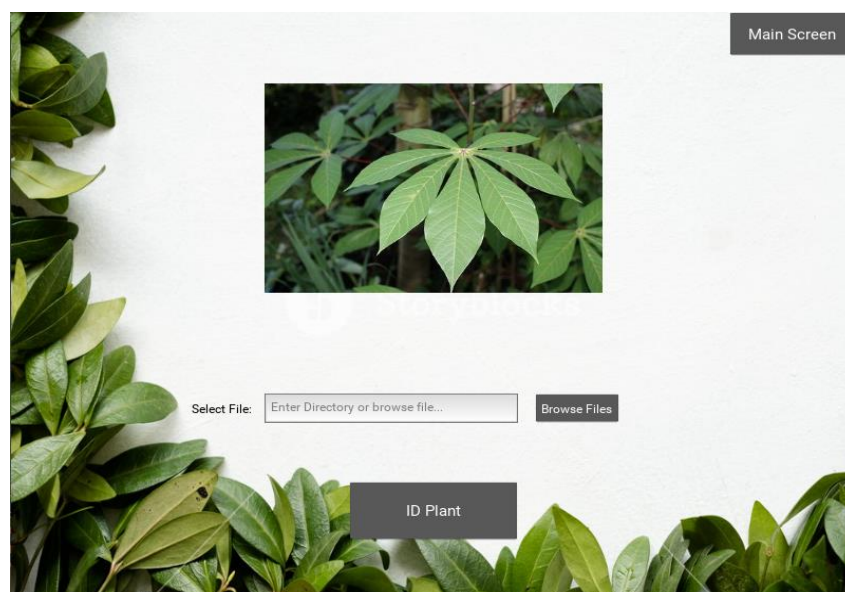


Figure 24: ID plant UI

5.3 Mobile Application

The Mobile App is developed to serve as an extension of the PC app. Users can use their phone to take pictures of plants and receive an identification wherever they may be. The UI of the mobile app is designed to follow a similar theme as that of the PC application. The figures below give an illustration of the mobile app UI.



Figure 26: Home Screen UI

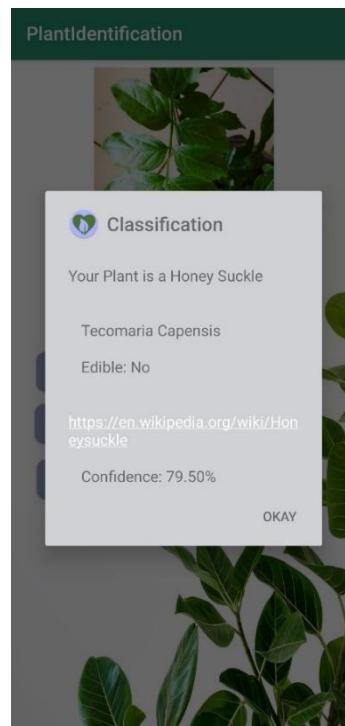


Figure 25: Classification Result

When the user presses the “ID my plant” button, the image is encoded into a string array and then sent as a JSON request to a Flask server. The Flask server then executes the same neural network classification code that PC app uses, and returns the result as a JSON response to the mobile app. The process is shown in the diagram below:

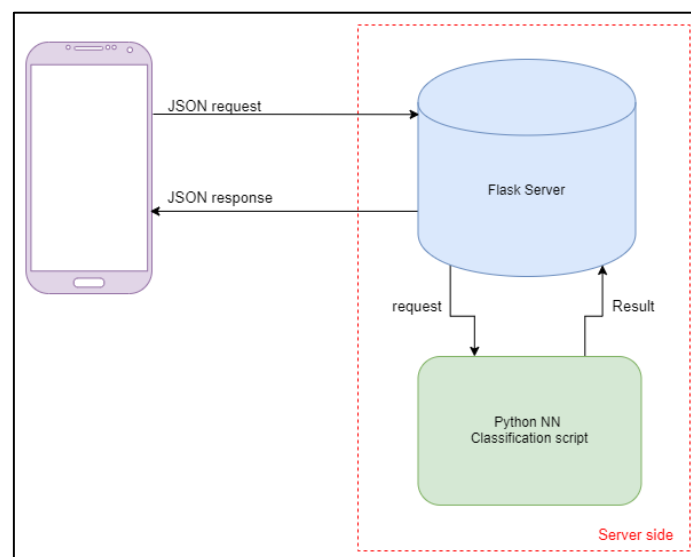


Figure 27: Android App server communication

There are two major advantages in this design. Firstly, the mobile application is very lightweight as all the large, resource intensive processes are executing on the server side. The second advantage is that user images are uploaded to the server whenever they request a plant identification, and this assists with the dataset problem as users will essentially be collecting images for us. These images are saved in a folder on the server PC where they can be sorted and used to retrain the network for improving accuracy and expanding the number of species.

5.3.1 Server Implementation

The server is implemented using flask framework, as it is very efficient for setting up small web-server applications. While running, the Server waits for any http POST requests on port 5000. When a request is received, it extracts the image string from the JSON request. The string is then decoded into an image and the ClassifyImage() python script is executed. When the script completes, the image is saved and named as the plant species, plus the current date. This is to assist with sorting the uploaded images.

Finally, the classification result is returned to the mobile app as a JSON response. The Server code is shown below:

```
def handle_request():
    # Decode the image string
    imageString = base64.b64decode(request.form['image'])

    # Save the image with the current date as it's name
    filepath = "C:/Users/shane/Pictures/MobileUploads/"
    filename = str(datetime.datetime.now().date()) + '_' + str(datetime.datetime.now().time().replace(':', '_'))
    filepathname = filepath + filename + '.jpg'
    print(filepathname)
    with open(filepathname, 'wb') as f:
        f.write(imageString)

    # execute the classification script
    classify, name, sci_name, edible, url, probability = test.my_image(filepathname)

    # Append result to the image name
    os.rename(filepathname, (filepath + name + '_' + filename + '.jpg').replace(' ', '_'))

    return jsonify({"response": name, "sci": sci_name, "edible": edible, "url": url, "probability": probability})
```

Figure 28: Flask server

5.3.2 Mobile App implementation

The implementation of the mobile app is very simple. The Image is initially encoded into a byte string (Figure 29 below).

```
private String imageToString(Bitmap bitmap){
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.JPEG, 100, byteArrayOutputStream);
    byte[] imgBytes = byteArrayOutputStream.toByteArray();
    return Base64.encodeToString(imgBytes, Base64.DEFAULT);
}
```

Figure 29: Convert image to byte string

After this a JSON request is sent to the server using it's IP address and port 5000. Finally, the response is captured, and the results are unpacked into variables (Figure 30).


```

private void uploadImage() {
    StringRequest stringRequest = new StringRequest(Request.Method.POST, UploadUrl,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String s) {
                System.out.println(s);
                pBar.setVisibility(View.GONE);
                try {
                    JSONObject jsonObject = new JSONObject(s);
                    String Response = jsonObject.getString("response");
                    String sci = jsonObject.getString("sci");
                    String edible = jsonObject.getString("edible");
                    String url = jsonObject.getString("url");
                    String probability = jsonObject.getString("probability");
                    SpannableString MyPlant = new SpannableString("\nYour Plant is a " + Response +
                        "\n\n\n\t\t" + sci + "\n\n\n\t\tEdible: " + edible + "\n\n\n\t\t" + url +
                        "\n\n\n\t\tConfidence: " + probability + "%");
                    Linkify.addLinks(MyPlant, Linkify.ALL);
                }
            }
        })
    ;
}

```

Figure 30: JSON request

5.4 Plant Classification Function

The Function that identifies a plant is the same one used for both the PC and mobile apps. It accepts the image directory as a parameter and returns the following 5 variables:

Plant name, scientific name, if its edible, web-link and the classification confidence. The function code is shown in the figure below:

```

class ClassifyImage:
    def __init__(self):
        obj = WideResnet()
        self.net = obj.MyWideResnet()
        self.net.load_state_dict(torch.load(Constants.MODEL_STORE_PATH + 'MY_DS_TransferLearn76ckpt'))
        self.net.eval()

    def my_image(self, direct):
        classification, name, sci_name, edible, url, probability = "", "", "", "", "", ""
        try:
            real = Image.open(direct)
            x = transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor(),
                                   transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])
            inp = x(real).unsqueeze_(0)
            outputs = self.net(inp)
            _, predicted = torch.max(outputs, 1)

            # get Confidence/probability
            sm = torch.nn.Softmax(dim=1)
            prob = sm(outputs)
            probability = Decimal(prob[0][predicted.item()].item() * 100)
            probability = Decimal(probability.quantize(Decimal('.01'), rounding=ROUND_HALF_DOWN))
            probability = str(probability)

            # Fetch DB values for plant
            mycursor, mydb = connection()
            print(predicted.item())
            sql = "SELECT * FROM 'PlantSpecies' WHERE 'PlantID' = %s"
            mycursor.execute(sql, (predicted.item() + 1,))
            myresult = mycursor.fetchall()
            for row in myresult:
                classification = row[0]
                name = row[1]
                sci_name = row[2]
                edible = row[3]
                url = row[4]

        except Exception as e:
            print("Failed to classify image")
            print(e)
        return classification, name, sci_name, edible, url, probability

```

Figure 31: plant classification function

The image is resized, normalized and then passed through the network, which returns the predicted class. This output is then passed through a SoftMax layer which returns the result as a probability distribution. This probability signifies the confidence of the classification.

A connection to the plant database is then made and data pertaining to the specific plant is retrieved. These values, along with the probability are returned to the application.

5.5 Plant Dataset

One of the biggest challenges for this project was organising an appropriate dataset. There are very few publicly available leaf datasets, and most of the available datasets have a very limited number of samples or are not properly labelled and therefore cannot be used to train the network. The dataset used for this project is a collection of 3 smaller datasets namely:

Flavia (Wu, 2007) : 32 Species - approximately 60 images per class.

Swedish Tree Dataset (Söderkvist, 2001): 15 Species – 75 images per class.

Leafsnap Dataset (Kumar, 2012): 29 Species – approximately 100 images per class.

The Final dataset for used for this project contains 76 different species of plants. Sample images of the different plants are shown below:

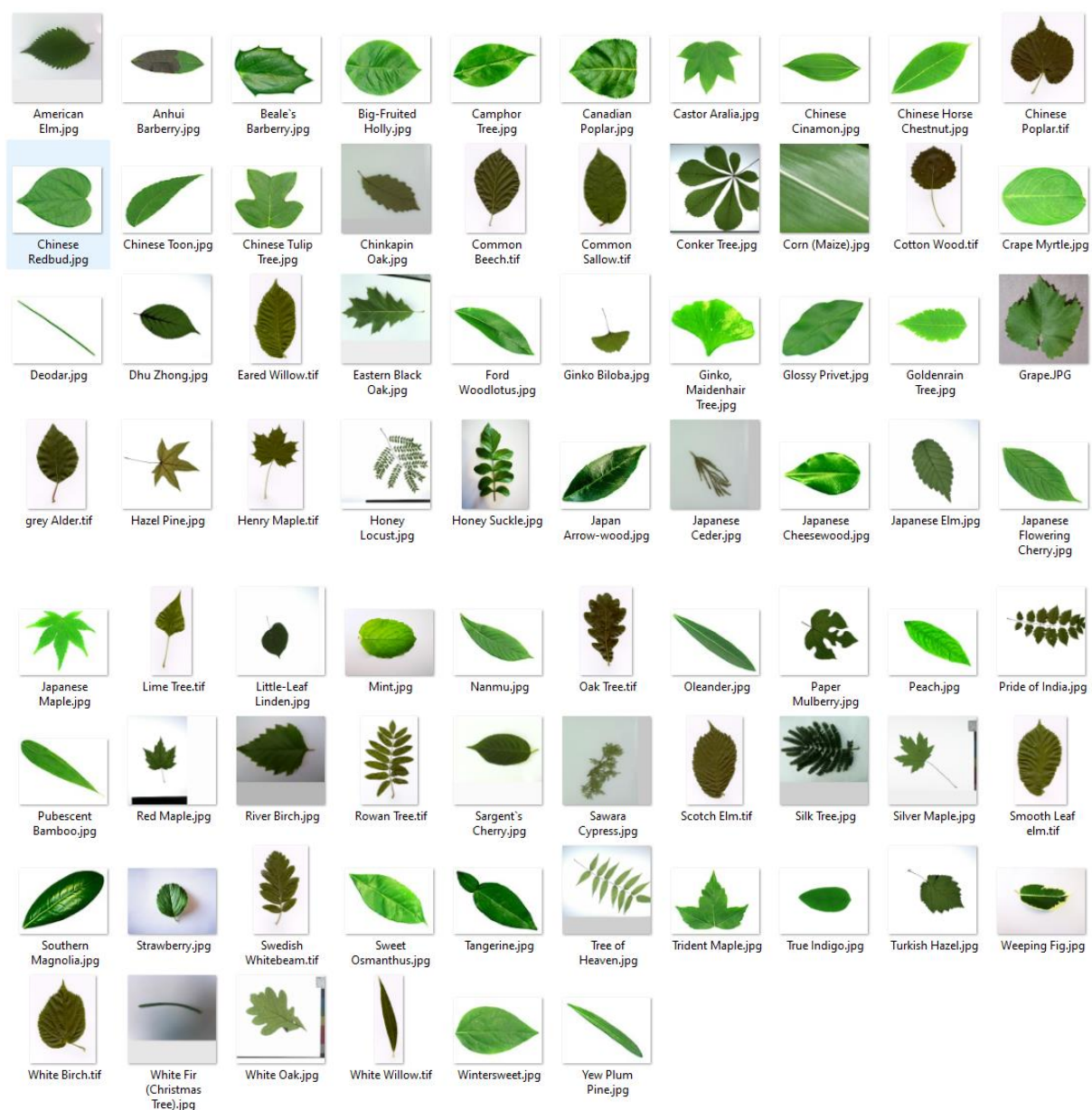


Figure 32: Sample Dataset Images

5.6 Plant Database

A MySQL database was set up and populated with information about each of the 76 plant species. The database contains the plants scientific name, if it is edible, as well as a web-link to find further information on the plant. The database is hosted on an Apache webserver so that it can be accessed remotely by the PC and mobile applications. The figure below shows a sample of the information stored in the table.

PlantID	Name	Scientific Name	Edible	URL
1	Pubescent Bamboo	Phyllostachys edulis (Carr.) Houz.	Yes - Shoots	https://en.wikipedia.org/wiki/Phyllostachys_edulis
2	Chinese Horse Chestnut	Aesculus chinensis	Yes - Acorn	https://en.wikipedia.org/wiki/Aesculus
3	Anhui Barberry	Berberis anhweiensis Ahrendt	No	https://en.wikipedia.org/wiki/Berberis
4	Chinese Redbud	Cercis chinensis	No	https://en.wikipedia.org/wiki/Cercis_chinensis
5	True Indigo	Indigofera tinctoria L.	No	https://en.wikipedia.org/wiki/Indigofera_tinctoria
6	Japanese Maple	Acer Palmatum	No	https://en.wikipedia.org/wiki/Acer_palmatum
7	Nanmu	Phoebe nanmu (Oliv.) Gamble	No	https://en.wikipedia.org/wiki/Phoebe_nanmu
8	Castor Aralia	Kalopanax septemlobus (Thunb. ex A.Murr.) Koidz.	Yes - Fruit	https://en.wikipedia.org/wiki/Kalopanax
9	Chinese Cinamon	Cinnamomum japonicum Sieb.	Yes - Bark	https://en.wikipedia.org/wiki/Cinnamomum_cassia
10	Goldenrain Tree	Koelreuteria paniculata Laxm.	No	https://en.wikipedia.org/wiki/Koelreuteria_panicul...
11	Big-Fruited Holly	Ilex macrocarpa Oliv.	Yes - Fruit	https://en.wikipedia.org/wiki/Holly

Figure 33: Plant Database

The central database allows the plant information for all the applications to be managed and updated in one place. It has an added advantage that the applications require less memory and hence are more lightweight.

5.7 Transfer Learning

A supervised pretraining technique called transfer learning was used to improve training and assist with the prevention of overfitting. The concept is to initially train the network on a large dataset that differs substantially from the actual smaller dataset (Wick & Puppe, 2017). This allows the network to learn generalized feature extraction properties which assists when training on the final dataset. The trained weights of all the layers except for the last classification layer are used as initial conditions for training the network on the final dataset. The Caltech101 dataset (Lei, Fergus, & Perona, 2004) was used for this purpose. The figure below provides a sample of the classes found in the dataset.

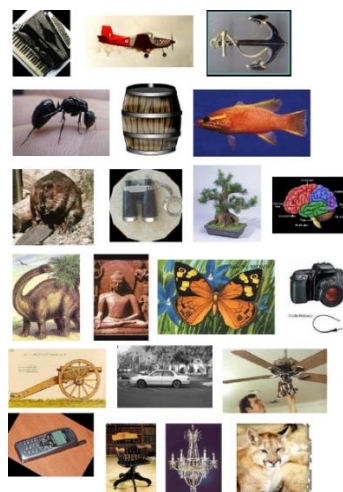


Figure 34: Sample Caltech101 images

The Network was trained for 25 epochs on this dataset which took a total of 8 hours 21 min. Section 6.2.1 shows the positive effect that transfer learning had on the training process.

6. System Testing

This section details the thorough tests that have been performed on the system and discussion of the results that have been obtained.

6.2 The Wide-ResNet Tests

The network was trained for 15 epochs, using batch sizes of 8. It was found that using smaller batch sizes is more appropriate for systems with smaller datasets as there is a slight noise incurrence that actually helps the network generalize better (Luschi & Masters, 2018). Stochastic Gradient Descent (SGD) optimizer was used with Nesterov momentum. Learning rate = 0.01 initially and was halved every 5 epochs. Cross Entropy was the loss function used.

6.2.1 The Loss function

In the following test, the validation loss was tracked throughout the training process and plotted as shown in figure 35. The test was performed under 3 conditions:

- The complete model with data augmentation and transfer learning
- Data augmentation and no transfer learning
- Transfer learning but no data augmentations

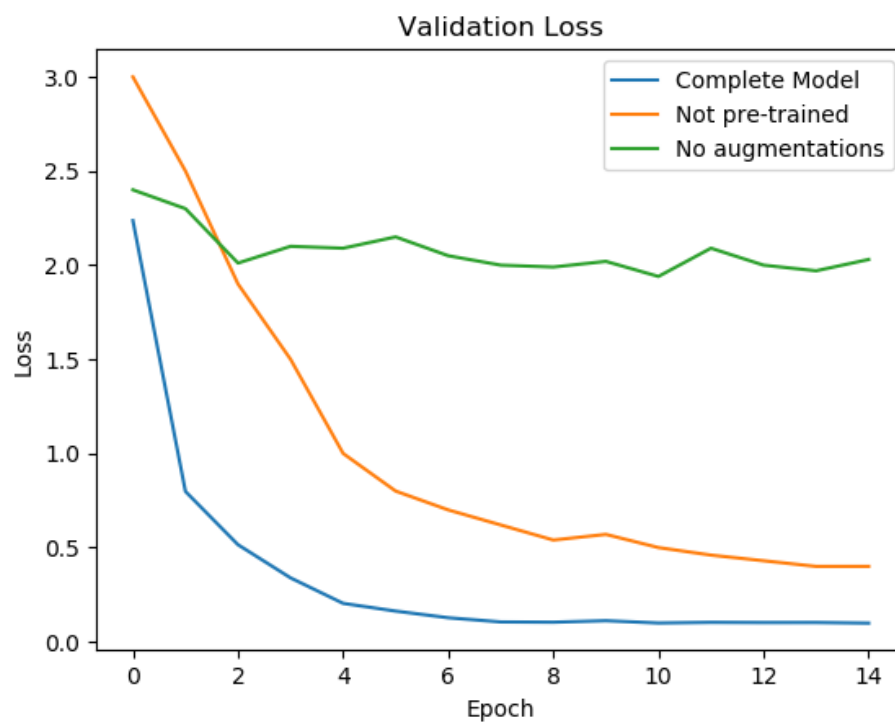


Figure 35: Training Loss

The complete model trained extremely well with a steep initial drop after which the loss value plateaued at around 0.04. The model without transfer learning performed relatively well too, although it can be noted that the model trained slower and converged to a higher loss value. This is due to the pre-trained model having already acquired generalized feature extraction capabilities from training on a larger dataset.

The model with no data augmentations, however, performed poorly and did not converge at all. As expected, the network overfitted to the small training data set and hence the validation loss did not improve. This test proves how vital the data augmentations are, when training a network with a small dataset.

6.2.2 Accuracy Graph

The training and validation accuracy of the complete model was tracked during the training process and the resulting graph is shown below.

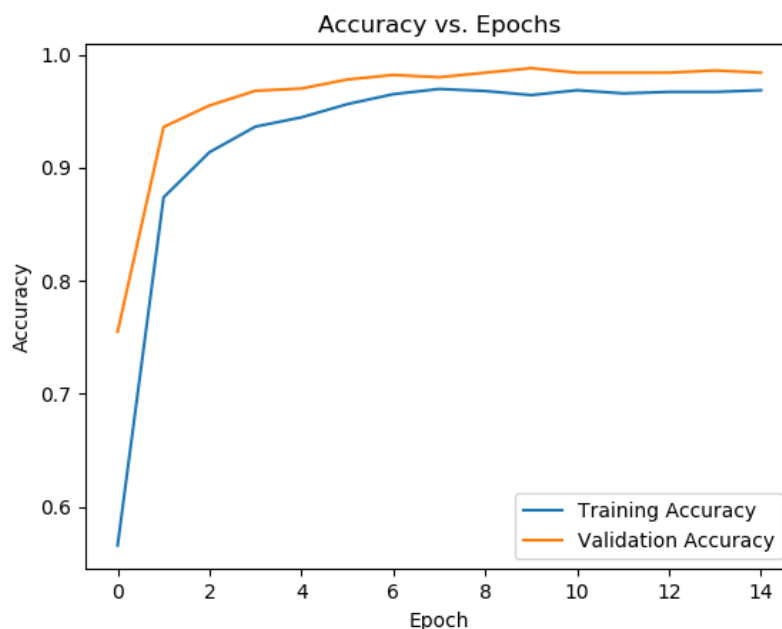


Figure 36: Training and validation accuracy

The validation accuracy converged at a maximum value of 0.988 after 9 epochs. The model parameters were saved at that point. This model is used for performing the systems image classification. The validation accuracy is slightly higher than training accuracy due to the large data augmentations and dropout that is applied during training.

The final validation accuracy of 98.8%, surpasses the goal of 95% and is competitive among the best performing papers that were researched in section 2.5. The classification accuracy of each class was also noted, with many plant species having 100% correct classifications. This is shown in figure 37 below and the full breakdown can be seen in appendix C1 and C2.

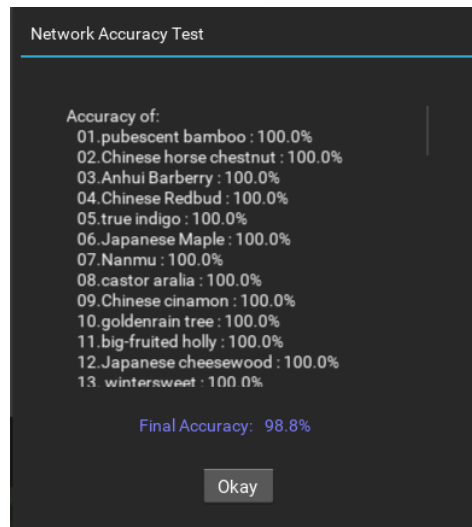


Figure 37: Final validation accuracy

6.2.3 Confusion Matrix

A confusion matrix for the network's validation classifications was built using the function shown in figure 38 below. This resulting matrix was plotted as a heat map which is shown in figure 39.

```
# Confusion Matrix
nb_classes = 76
confusion_matrix = torch.zeros(nb_classes, nb_classes)
with torch.no_grad():
    for i, (inputs, classes) in enumerate(dataloaders['val']):
        inputs = inputs.to(device)
        classes = classes.to(device)
        outputs = model_ft(inputs)
        _, preds = torch.max(outputs, 1)
        for t, p in zip(classes.view(-1), preds.view(-1)):
            confusion_matrix[t.long(), p.long()] += 1
print(confusion_matrix)
```

Figure 38: Confusion matrix code

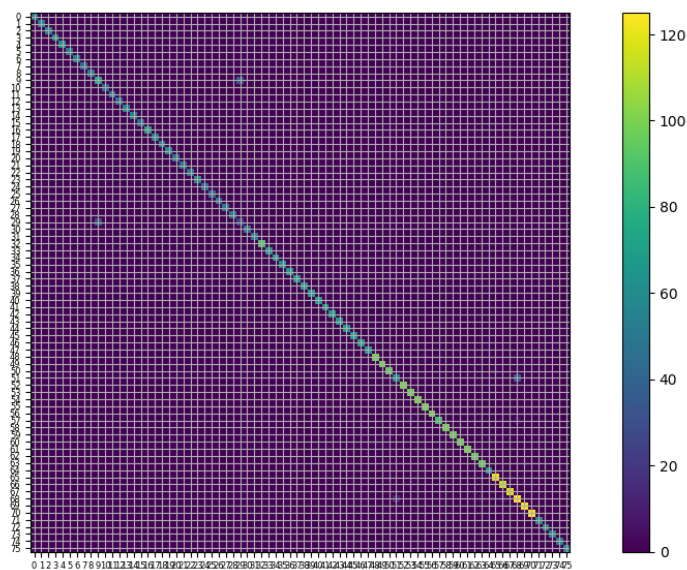


Figure 39: Confusion Matrix

The diagonal of the matrix illustrates the correct classifications. As expected, most of the classifications occur in this region. The lighter coloured dots that don't appear on the diagonal indicate where the misclassifications are occurring. It can be noted that the misclassifications occur mostly in 4 places.

Looking at one of these dots, we see it occurs between classes 52 and 69. An example leaf image of both classes are shown below:



Figure 41: class 52- *Quercus Meulenberg*



Figure 40: class 69 - *Prunus Sargenti*

It is now understandable that the network would make a misclassification. The colour and shape of these leaves are extremely similar. The confusion matrix provides a good indication of which plant species share very similar leaf features.

6.2.3 Network Accuracy vs. Number of classes

This test was performed by training the network on various sized datasets, ranging from 15 to all 76 classes. This provides an indication of how well the network scales and provides a direct comparison with the results obtained in related work. The graph below shows the results obtained:

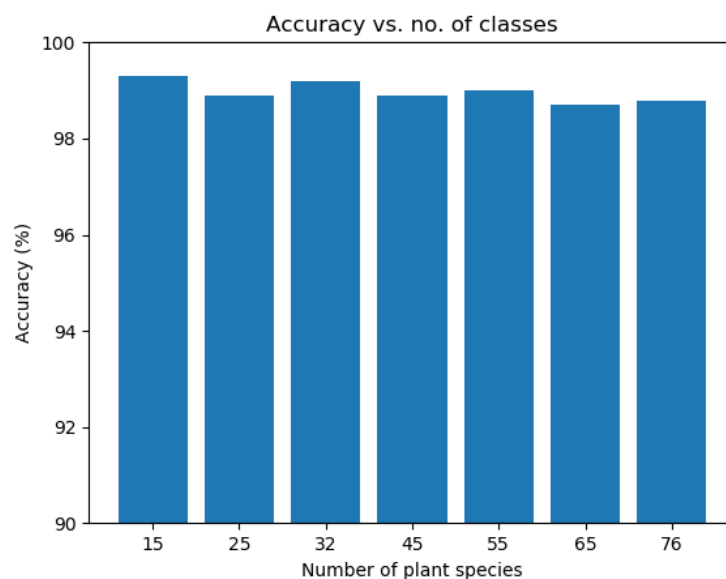


Figure 42: Accuracy vs number of species

The results show that the network scales well with the increase in number of classes (plant species).

The network achieved an accuracy of 99.2% when trained on the Flavia Dataset (32 species). This outperforms the 99.0% accuracy that (Wick & Puppe, 2017) achieved, but falls short of the 99.8% accuracy achieved by (Jeon & Rhee, 2017). They employed techniques such as averaging, where 10 networks would be trained from different initial values and the classification results averaged between the 10 networks. This, however, is not feasible for a practical implementation as performance would be 10x slower.

6.2.4 Training Time

The PC used to train the network had the following specs:

CPU: Intel i5-7200u @ 2.5GHz

RAM: 8GB

GPU: Nvidia GeForce 920MX – 2GB RAM

The Network took a total of 192min = 3 hours 12 min to complete 15 epochs of training. Training time could be reduced if a high-performance machine was used.

```
Epoch 14/14
-----
train Loss: 0.0988 Acc: 0.9685
val Loss: 0.0366 Acc: 0.9840

Training complete in 192m 46s
Best val Acc: 0.988000
```

Figure 43: Total Training time- 15 Epochs

6.2.5 Model Size

A NN library – torchSummary was used to display the parameters and size of the model. The Network contains a total of just over 11 Million parameters and has a total size of 106.14 MB.

These values are as expected and fall within the theoretical predicted size. The Aim of the initial network design was to remain under 15M parameters for system responsiveness.

```
=====
Total params: 11,215,500
Trainable params: 11,215,500
Non-trainable params: 0
-----
Input size (MB): 0.57
Forward/backward pass size (MB): 62.79
Params size (MB): 42.78
Estimated Total Size (MB): 106.14
=====
```

Figure 44: Model Parameters and Size

The Entire model summary can be found in Appendix C3



6.3 Practical Test

This test was conducted to determine the accuracy of the system in a practical application. 4 plants species were bought that are contained in the dataset. These plants are Mint, Strawberry, Honey Suckle and Weeping Fig. The plants are shown below:



Figure 45: Test Plants

The table below shows the results obtained when 7 images were taken and identified from the mobile app.

Image	Ground Truth	Prediction
	Mint	Mint
	Strawberry	Strawberry
	Honey Suckle	Honey Suckle



Weeping Fig

Weeping Fig

Weeping Fig

Weeping Fig

Strawberry

Strawberry

Honey Suckle

Honey Suckle

The Network correctly classified all 7 leaf pictures. The final 2 images of the Strawberry and Honey Suckle were taken in-front of a noisy background and the network was still able to give a correct classification. The network was trained purely on images with white background and yet can perform correct testing classifications without this constraint. This indicates that the network could be trained with noisy images and the white background constraint can be neglected.

7. Feasibility Analysis

7.1 Market Feasibility

This project includes potential for a wide scope of real-world implementations. The objective of the project, to assist botanists with plant identification is a real need currently. The software can be extended into a commercial product that can be used by anybody to identify plants. Further applications of the software include poisonous plant identification and medicinal plant identification.

7.2 Budget Feasibility

Conducting a financial feasibility study showed that the cost of developing this system is negligible. The System is entirely software based and the software that will be required to implement the solution (detailed in section 3.2.3) is all free to download and use. The only costs of the project will be electricity and Internet bandwidth used in the development of the system.

7.3 Technological Feasibility

Due to Neural Networks being very computationally expensive, a possible concern would be that the hardware cannot support this software. This, however, will not be a major constraint as most home PC these days contain enough memory and processing power to comfortably run the software. The Mobile application was designed to have all the processing performed on the server side; this means that almost any android smartphone can run the application.

7.4 Time allocation

Plant Identification System			Duration (Days)
START DATE	END DATE	DESCRIPTION	
08/07	22/07	Project Proposal and research	14
23/07	12/08	Initial Concept Design	20
13/08	10/09	Design concept Finalisation	28
11/09	10/10	System Development	30
11/10	02/11	System Testing and Evaluation	23
	04/11	Phase 3 report and presentation	

Table 1: Project Timeline

7.4.1 Gantt Chart

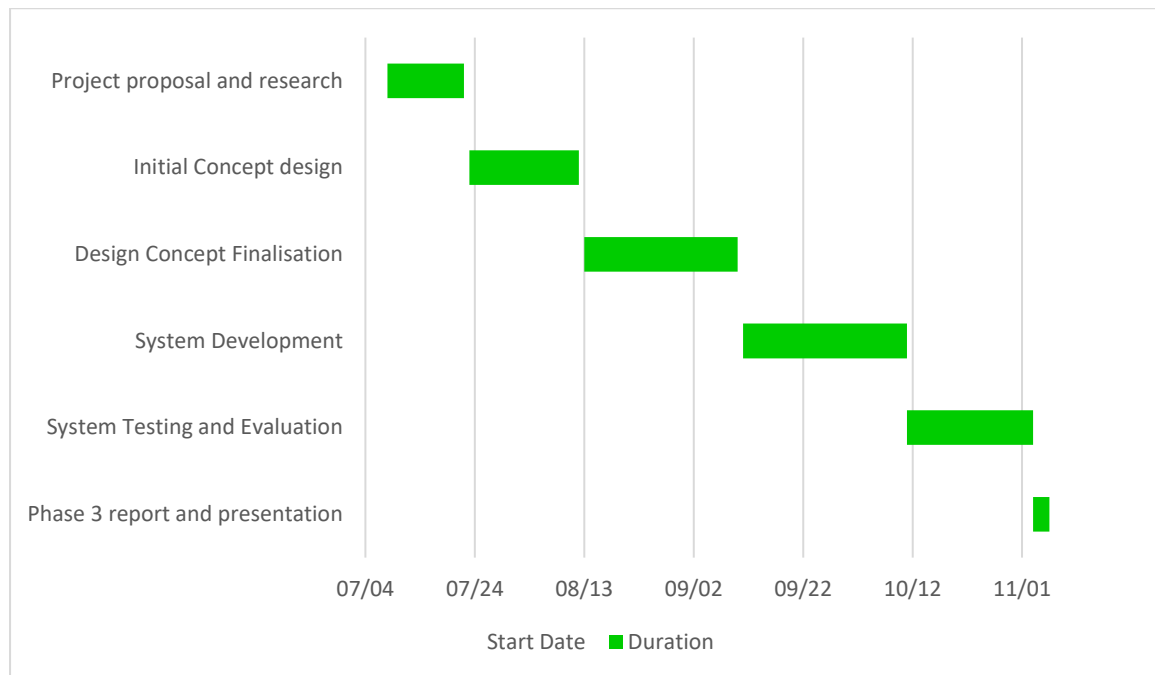


Figure 46: Gantt Chart

8.Future Work

The results obtained in section 6.2.3 indicate that this system would scale well when adding many more classes (plant species). A dataset containing a larger selection of plant species could be collected, followed by training the network on this new dataset. This could be achieved by working in collaboration with botanists; such as the South African botany department; to have a collection of plants indigenous to an area that the software can identify.

Another recommended areas for future research are:

Generalized plant identification - Expanding the software to classify plants, not only on leaves, but flowers, fruit and full plant pictures would provide a commercial application for the software. Plants with very similar leaf patterns could be identified by these other distinguishing features.

Image segmentation and background noise reduction – Incorporating an image pre-processing technique to extract only the foreground leaf image and remove all background noise, would remove the constraint of taking pictures in front of a solid background. This would make the software more convenient for commercial use.

9. Conclusion

This report documented the entire process of developing an automated leaf-based plant identification software. The research documented in phase 1 indicated that the most effective deep learning solution would be a Wide-Resnet convolutional neural network. The design methodology proposed in phase 2 was used to successfully implement the system. A mobile application was developed in addition to the PC app, which provides a convenient “on the go” method of identifying plants.

The system achieved a 98.8% testing accuracy and can classify 76 different plant species. These results are competitive with the best results obtained by others in the related field. The network showed that it scales well, and data augmentation was proven to be key in overcoming the challenge of a small dataset.

The research documented, as well as the thorough system testing indicate that this solution will solve the need for assisting botanists with the identification of plants. This, along with the thorough feasibility study concludes that the project aim was met within the required time allocation and set specifications.

References

- Brownlee, J. (2018). A Gentle introduction to dropout for Regularizing neural networks. Retrieved July 21, 2019
- Brownlee, J. (2019). How to use Learning Curves to Diagnose Machine Learning Model Performance. Retrieved July 21, 2019, from <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>
- Das, S. (2017). CNN Architectures: LeNet , AlexNet , VGG , GoogleNet, ResNet and more... *Medium Corporation*. Retrieved July 18, 2019, from <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- Das, S. (2017, November). CNN Architectures. Medium Corporation. Retrieved July 20, 2019, from <https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>
- Dasgupta, S. (2016). How many plant species are there in the world. *Mongabay*. Retrieved June 20, 2019, from <https://news.mongabay.com/2016/05/many-plants-world-scientists-may-now-answer/>
- Du, J.-X., Wang, X., & Zhang, G. (2007). Leaf shape based plant species recognition. *Applied Mathematics and Computation*, 185. Retrieved September 08, 2019
- Griess, T. (2013). Like fingerprints , a tree`s leaves are unique. *Kearneyhub*. Retrieved July 20, 2019, from https://www.kearneyhub.com/news/life/yard-and-garden/like-fingerprints-a-tree-s-leaves-are-unique/article_bdc0b618-69be-11e3-9b3b-001a4bcf887a.html
- Harsani, P. (2017). MEDICINAL PLANT SPECIES IDENTIFICATION SYSTEM USING TEXTURE ANALYSIS AND MEDIAN FILTER. *Kursor*. Retrieved September 2019, 03
- He, K. (2015). *Deep Residual Learning for Image recognition*. CA. Retrieved July 21, 2019
- Hedjazi, M. (2017). On identifying leaves: A comparison of CNN with classical ML methods. Retrieved September 08, 2019
- J.Norman, R. (1999). Feasibility Analysis and requirement determination. Retrieved July 15, 2019
- Jain, Y. (2018). Tensorflow or Pytorch? *Medium Corporation*. Retrieved September 06, 2019, from <https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4>
- Jeon, W.-S., & Rhee, S.-Y. (2017). Plant recognition using a Convolutional Neural Network. *International Journal of Fuzzy Logic and Intelligent Systems*. Retrieved September 08, 2019
- Jha, P. (2019). A brief overview of loss functions in pytorch. Retrieved July 17, 2019, from <https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7>
- Kadir, A., & Nugroho, L. (2011). Foliage Plant Retrieval Using Polar Fourier Transform, Color Moments and Vein Features. *Gadjah Mada University*. Retrieved September 08, 2019
- Karim, R. (2018). Illustrated: 10 CNN Architectures. *Medium Corporation*. Retrieved September 03, 2019, from <https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d>

- Karpathy. (2017). CS231n Convolutional Neural network for visual recognition. Retrieved July 18, 2019, from <http://cs231n.github.io/neural-networks-1/>
- Kumar, N., & Biswas, A. (2012). A Computer Vision System for Automatic Plant Species Identification. *Proceedings of the 12th European Conference on Computer Vision (ECCV)*. Retrieved September 08, 2019
- Kumar, N., Belheumer, P., & Arjit Biswas. (2012). *Leafsnap: A Computer Vision System for Automatic Plant Species Identification*. Proceedings of the 12th European Conference on Computer Vision (ECCV). Retrieved October 31, 2019, from <http://leafsnap.com/dataset/>
- Lei, F.-F., Fergus, R., & Perona, P. (2004). *Learning generative visual models from few training examples*. IEEE. CVPR 2004, Workshop on Generative-Model.
- Luschi, C., & Masters, D. (2018). Revisiting small batch training for deep neural networks. *GraphCore*. Retrieved September 06, 2019, from <https://www.graphcore.ai/posts/revisiting-small-batch-training-for-deep-neural-networks>
- Morris, B. (2018). Building a World-Class CIFAR-10 Model From Scratch. Retrieved September 03, 2019, from <https://brandonmorris.dev/2018/06/30/wide-resnet-pytorch/>
- Newmaster, S. (2006). DNA barcoding in land plants: evaluation of rbcl in multigene tiered approach. *Canadian Journal of Botany*, 1,2. Retrieved July 20, 2019, from <https://www.nrcresearchpress.com/doi/full/10.1139/b06-047>
- Saha, S. (2018). A comprehensive guide to Convolutional Neural Networks. *Towards Data Science*. Retrieved July 21, 2019, from <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- Söderkvist, O. (2001). *Computer vision classification of leaves from swedish trees*. Master Thesis, Linköping University. Retrieved October 2019, 2019, from <https://www.cvl.isy.liu.se/en/research/datasets/swedish-leaf/>
- Sujaro, W. (2014). Traditional knowledge of wild and semi-wild edible plants used in Bali (Indonesia) to maintain biological and cultural diversity. *Official Journal of the Societa Botanica Italiana*. Retrieved September 03, 2019
- Voskoglou, C. (2017). What is the best programming language for Machine Learning? *Medium Corporation*. Retrieved September 03, 2019, from <https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7>
- Wang, C. (n.d.). *Convolutional neural network for Image Classification*. John Hopkins university, Baltimore. Retrieved July 17, 2019
- Wang, C.-F. (2015). The Vanishing Gradient Problem. *Towards Data Science*. Retrieved September 03, 2019, from <https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>
- Wang, X., Liang, J., & Guo, F. (2014). Feature extraction algorithm based on dual scale decomposition and local binary descriptors for plant leaf recognition. *Science Direct*, 1. Retrieved July 20, 2019
- Wibowo, A. (2018). Classification algorithm for edible mushroom identification. 1. Retrieved September 2019, 03, from <https://ieeexplore.ieee.org/document/8350746>

- Wick, C., & Puppe, F. (2017). *Leaf Identification using a deep neural network*. University of Wurzburg, Wurzburg, Germany. Retrieved July 20, 2019
- Wu, S. G. (2007). A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network. *Cornell University*. Retrieved September 08, 2019
- Zagoruyko, S., & Komodakis, N. (2017). Wide Residual Networks. Retrieved September 03, 2019

Appendix

Appendix A – ResNet Full Block Diagram

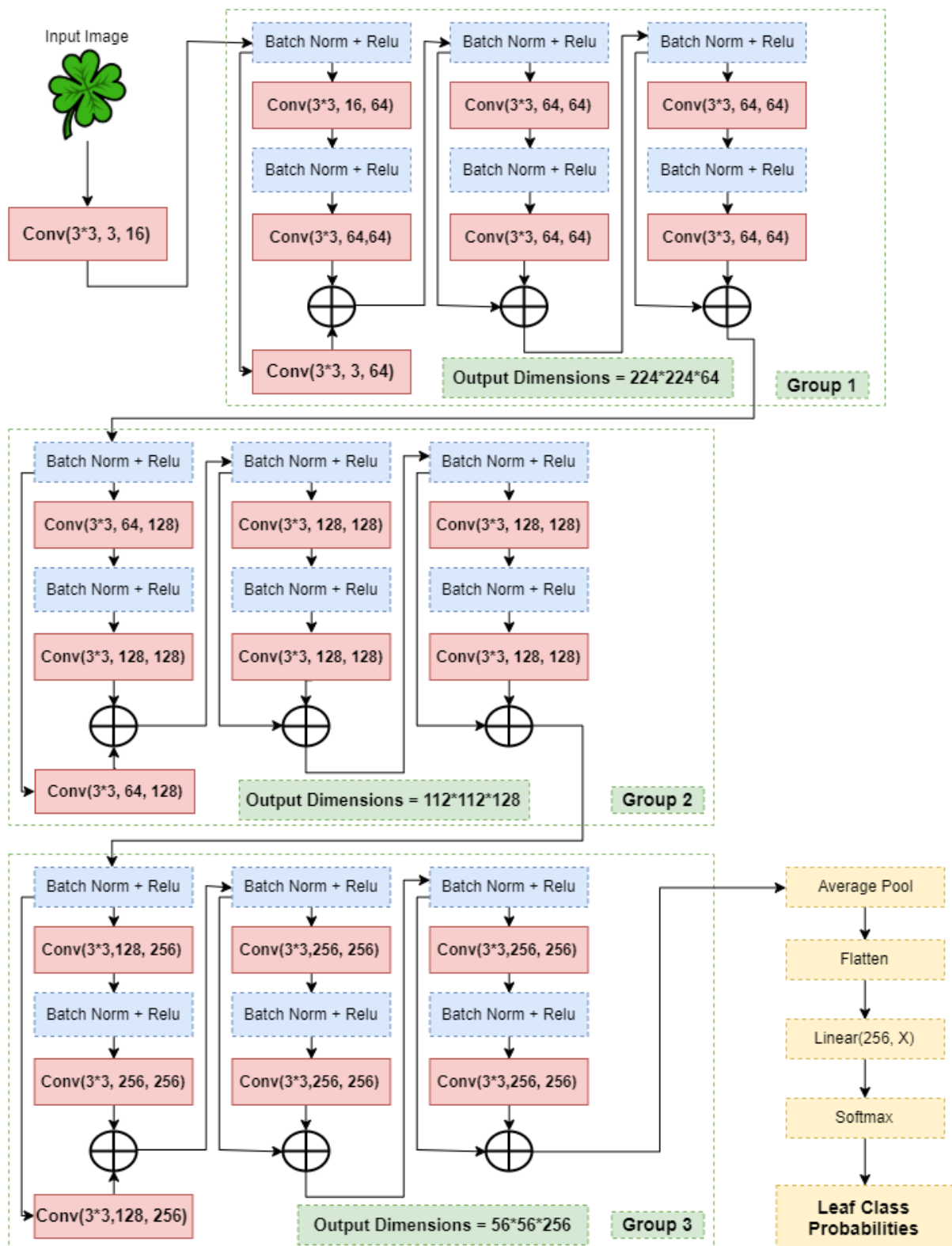


Figure 47: Full ResNet Block Diagram

Appendix B – User Interface

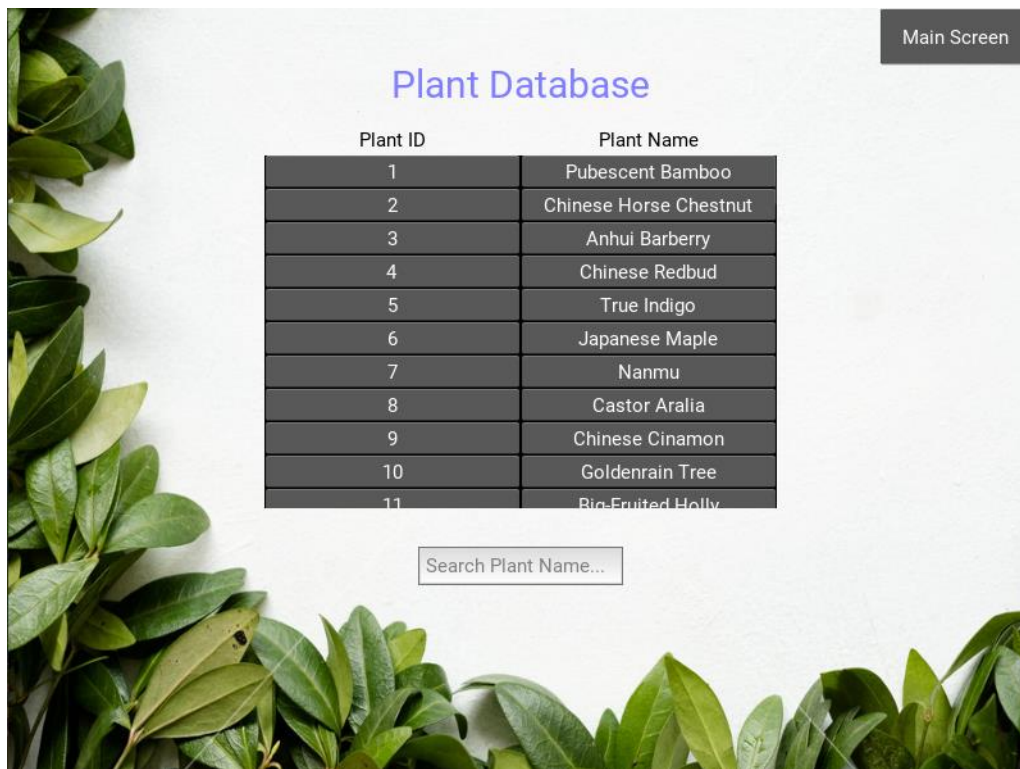


Figure 48: Database UI Screen

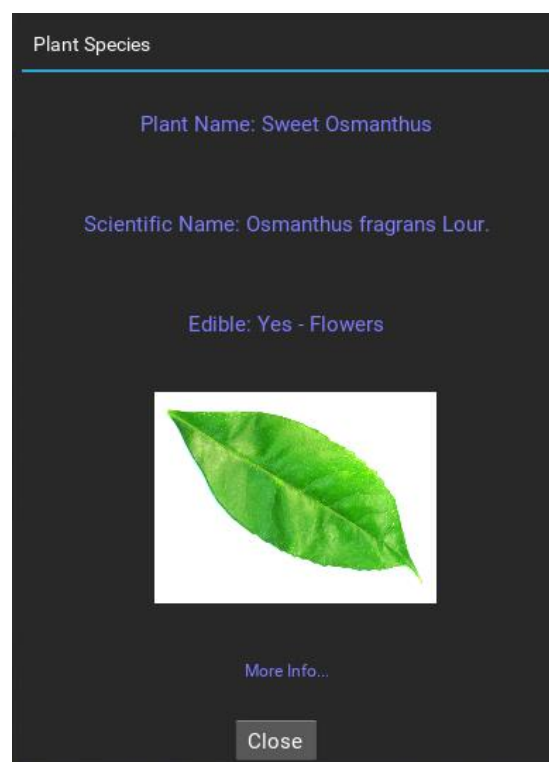


Figure 49: Classification result UI

Appendix C

Appendix C1 – class accuracy

```
Accuracy of 01.pubescent bamboo : 100 %
Accuracy of 02.Chinese horse chestnut : 100 %
Accuracy of 03.Anhui Barberry : 100 %
Accuracy of 04.Chinese Redbud : 100 %
Accuracy of 05.true indigo : 100 %
Accuracy of 06.Japanese Maple : 100 %
Accuracy of 07.Nanmu : 100 %
Accuracy of 08.castor aralia : 100 %
Accuracy of 09.Chinese cinamon : 100 %
Accuracy of 10.goldenrain tree : 100 %
Accuracy of 11.big-fruited holly : 100 %
Accuracy of 12.Japanese cheesewood : 100 %
Accuracy of 13. wintersweet : 100 %
Accuracy of 14.camphortree : 100 %
Accuracy of 15.Japan arrowwood : 100 %
Accuracy of 16.sweet osmanthus : 100 %
Accuracy of 17.deodar : 100 %
Accuracy of 18.ginko, maidenhair tree : 100 %
Accuracy of 19.Crape myrtle : 100 %
Accuracy of 20.Oleander : 100 %
Accuracy of 21.yew plum pine : 100 %
Accuracy of 22.Japanese flowering cherry : 100 %
Accuracy of 23.Glossy Privet : 100 %
Accuracy of 24.Chinese toon : 100 %
Accuracy of 25.Peach : 100 %
Accuracy of 26.Ford Woodlotus : 100 %
Accuracy of 27.trident maple : 100 %
Accuracy of 28.Beale's barberry : 100 %
Accuracy of 29.southern magnolia : 100 %
Accuracy of 30.Canadian poplar : 80 %
Accuracy of 31.Chinese tulip tree : 100 %
Accuracy of 32.tangerine : 100 %
Accuracy of 33. Honey Suckle : 100 %
Accuracy of 34. Ulmus Carpinifolia : 100 %
Accuracy of 35. Acer : 100 %
Accuracy of 36. Salix Aurita : 100 %
Accuracy of 37. Quercus : 100 %
Accuracy of 38. Alnus Incana : 100 %
Accuracy of 39. Betula Pubescens : 100 %
```

Figure 50:Class Accuracy 1

```
Accuracy of 40. Salix Alba 'Sericea' : 100 %
Accuracy of 41. Populus : 100 %
Accuracy of 42. Ulmus Glabra : 100 %
Accuracy of 43. Sorbus Aucuparia : 100 %
Accuracy of 44. Salix Sinerea : 100 %
Accuracy of 45. Populus : 100 %
Accuracy of 46. Tilia : 100 %
Accuracy of 47. Sorbus Intermedia : 100 %
Accuracy of 48. Fagus Silvatica : 100 %
Accuracy of 49. acer_rubrum : 100 %
Accuracy of 50. albitia julibrissin : 100 %
Accuracy of 51. chamaecyparis pisifera : 100 %
Accuracy of 52. quercus muehlenbergii : 77 %
Accuracy of 53. quercus velutina : 100 %
Accuracy of 54. tilia_cordata : 100 %
Accuracy of 55. abies_concolor : 100 %
Accuracy of 56. acer_saccharinum : 100 %
Accuracy of 57. aesculus_hippocastamon : 100 %
Accuracy of 58. broussonettia_papyrifera : 100 %
Accuracy of 59. ginkgo_biloba : 100 %
Accuracy of 60. gleditsia_triacanthos : 100 %
Accuracy of 61. liquidambar_styraciflua : 100 %
Accuracy of 62. quercus_alba : 100 %
Accuracy of 63. ailanthus_altissima : 90 %
Accuracy of 64. betula_nigra : 100 %
Accuracy of 65. corylus_colurna : 100 %
Accuracy of 66. cryptomeria_japonica : 100 %
Accuracy of 67. eucommia_ulmoides : 90 %
Accuracy of 68. koelreuteria_paniculata : 100 %
Accuracy of 69. prunus_sargentii : 91 %
Accuracy of 70. ulmus_american : 100 %
Accuracy of 71. zelkova_serrata : 100 %
Accuracy of 72. Mint : 100 %
Accuracy of 73. Strawberry : 100 %
Accuracy of 74. Grape : 100 %
Accuracy of 75. Ficus : 100 %
Accuracy of 76. Corn(Maize) : 100 %
Total correct percent: 98.8
```

Figure 51: Class Accuracy 2

Appendix C2 – training values

```
Epoch 0/14
-----
train Loss: 1.8368 Acc: 0.5658
val Loss: 0.2827 Acc: 0.9160

Epoch 1/14
-----
train Loss: 0.4981 Acc: 0.8737
val Loss: 0.1329 Acc: 0.9560

Epoch 2/14
-----
train Loss: 0.3152 Acc: 0.9137
val Loss: 0.1176 Acc: 0.9700

Epoch 3/14
-----
train Loss: 0.2390 Acc: 0.9362
val Loss: 0.0811 Acc: 0.9680

Epoch 4/14
-----
train Loss: 0.1937 Acc: 0.9445
val Loss: 0.1019 Acc: 0.9700

Epoch 5/14
-----
train Loss: 0.1426 Acc: 0.9562
val Loss: 0.0441 Acc: 0.9780

Epoch 6/14
-----
train Loss: 0.1173 Acc: 0.9650
val Loss: 0.0441 Acc: 0.9820

Epoch 7/14
-----
train Loss: 0.1051 Acc: 0.9696
val Loss: 0.0417 Acc: 0.9800
```

Figure 53: Network Training 1

```
Epoch 6/14
-----
train Loss: 0.1173 Acc: 0.9650
val Loss: 0.0441 Acc: 0.9820

Epoch 7/14
-----
train Loss: 0.1051 Acc: 0.9696
val Loss: 0.0417 Acc: 0.9800

Epoch 8/14
-----
train Loss: 0.1036 Acc: 0.9678
val Loss: 0.0375 Acc: 0.9840

Epoch 9/14
-----
train Loss: 0.1115 Acc: 0.9642
val Loss: 0.0307 Acc: 0.9880

Epoch 10/14
-----
train Loss: 0.0994 Acc: 0.9685
val Loss: 0.0311 Acc: 0.9840

Epoch 11/14
-----
train Loss: 0.1028 Acc: 0.9657
val Loss: 0.0333 Acc: 0.9840

Epoch 12/14
-----
train Loss: 0.1018 Acc: 0.9670
val Loss: 0.0325 Acc: 0.9840

Epoch 13/14
-----
train Loss: 0.1020 Acc: 0.9670
val Loss: 0.0336 Acc: 0.9860

Epoch 14/14
-----
train Loss: 0.0988 Acc: 0.9685
val Loss: 0.0366 Acc: 0.9840

Training complete in 192m 46s
Best val Acc: 0.988000
```

Figure 52: Network training 2

Appendix C3 – Model Summary

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
ReLU-3	[-1, 64, 112, 112]	0
MaxPool2d-4	[-1, 64, 56, 56]	0
Conv2d-5	[-1, 64, 56, 56]	36,864
BatchNorm2d-6	[-1, 64, 56, 56]	128
ReLU-7	[-1, 64, 56, 56]	0
Conv2d-8	[-1, 64, 56, 56]	36,864
BatchNorm2d-9	[-1, 64, 56, 56]	128
ReLU-10	[-1, 64, 56, 56]	0
BasicBlock-11	[-1, 64, 56, 56]	0
Conv2d-12	[-1, 64, 56, 56]	36,864
BatchNorm2d-13	[-1, 64, 56, 56]	128
ReLU-14	[-1, 64, 56, 56]	0
Conv2d-15	[-1, 64, 56, 56]	36,864
BatchNorm2d-16	[-1, 64, 56, 56]	128
ReLU-17	[-1, 64, 56, 56]	0
BasicBlock-18	[-1, 64, 56, 56]	0
Conv2d-19	[-1, 128, 28, 28]	73,728
BatchNorm2d-20	[-1, 128, 28, 28]	256
ReLU-21	[-1, 128, 28, 28]	0
Conv2d-22	[-1, 128, 28, 28]	147,456
BatchNorm2d-23	[-1, 128, 28, 28]	256
Conv2d-24	[-1, 128, 28, 28]	8,192
BatchNorm2d-25	[-1, 128, 28, 28]	256
ReLU-26	[-1, 128, 28, 28]	0
BasicBlock-27	[-1, 128, 28, 28]	0
Conv2d-28	[-1, 128, 28, 28]	147,456
BatchNorm2d-29	[-1, 128, 28, 28]	256
ReLU-30	[-1, 128, 28, 28]	0
Conv2d-31	[-1, 128, 28, 28]	147,456
BatchNorm2d-32	[-1, 128, 28, 28]	256
ReLU-33	[-1, 128, 28, 28]	0
BasicBlock-34	[-1, 128, 28, 28]	0
Conv2d-35	[-1, 256, 14, 14]	294,912
BatchNorm2d-36	[-1, 256, 14, 14]	512
ReLU-37	[-1, 256, 14, 14]	0
Conv2d-38	[-1, 256, 14, 14]	589,824

Figure 54: Network parameter summary 1

BatchNorm2d-39	[-1, 256, 14, 14]	512
Conv2d-40	[-1, 256, 14, 14]	32,768
BatchNorm2d-41	[-1, 256, 14, 14]	512
ReLU-42	[-1, 256, 14, 14]	0
BasicBlock-43	[-1, 256, 14, 14]	0
Conv2d-44	[-1, 256, 14, 14]	589,824
BatchNorm2d-45	[-1, 256, 14, 14]	512
ReLU-46	[-1, 256, 14, 14]	0
Conv2d-47	[-1, 256, 14, 14]	589,824
BatchNorm2d-48	[-1, 256, 14, 14]	512
ReLU-49	[-1, 256, 14, 14]	0
BasicBlock-50	[-1, 256, 14, 14]	0
Conv2d-51	[-1, 512, 7, 7]	1,179,648
BatchNorm2d-52	[-1, 512, 7, 7]	1,024
ReLU-53	[-1, 512, 7, 7]	0
Conv2d-54	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-55	[-1, 512, 7, 7]	1,024
Conv2d-56	[-1, 512, 7, 7]	131,072
BatchNorm2d-57	[-1, 512, 7, 7]	1,024
ReLU-58	[-1, 512, 7, 7]	0
BasicBlock-59	[-1, 512, 7, 7]	0
Conv2d-60	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-61	[-1, 512, 7, 7]	1,024
ReLU-62	[-1, 512, 7, 7]	0
Conv2d-63	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-64	[-1, 512, 7, 7]	1,024
ReLU-65	[-1, 512, 7, 7]	0
BasicBlock-66	[-1, 512, 7, 7]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Linear-68	[-1, 76]	38,988
=====		
Total params: 11,215,500		
Trainable params: 11,215,500		
Non-trainable params: 0		
=====		
Input size (MB): 0.57		
Forward/backward pass size (MB): 62.79		
Params size (MB): 42.78		
Estimated Total Size (MB): 106.14		

Figure 55: Network parameter summary 2