Intelligent Mobile Plant Disease Diagnostic System Using NASNet-Mobile Deep Learning

Adedamola O. Adedoja, Pius A. Owolawi, Temitope Mapayi, *Member, IAENG*, and Chunling Tu

Abstract-- **Plant diseases remain a threat to global food supply, as it causes unimaginable loss of food and revenue. Intelligent mobile plant disease diagnostic system has however become valuable due to its usefulness for the early diagnosis and detection of plant diseases using leave images even when there are no availability of competent and adequate experts in such locations. The objective of this paper is to develop an intelligent mobile plant disease diagnostic system that runs on a smartphone. The diagnostic system is based on NASNet-Mobile, a lightweight convolutional neural network (CNN) architecture using the images of the plant leaves for plant disease diagnosis. A mobile application is developed for both android and iOS smartphones to capture the plant leaf images. The system runs on a web service that gets diagnosis from the CNN model. The plant leave images captured using the developed mobile application are sent through the web service and the recognition of the plant disease is achieved using NASNet-Mobile CNN model. The proposed NASNet-Mobile CNN model plant disease diagnostic system achieved an accuracy rate of 99.31%.**

Index Terms-- **Agriculture, Deep Learning, Image Processing, Machine Learning, Plant Disease Identification**

I. Introduction

PLANT diseases are a huge constraint to the production of agricultural crops [1]. They decrease plant production, and as such pose a threat to the global food supply and the livelihood of the farmers cultivating the crops [1]. Every agricultural crop is vulnerable to diseases and these diseases can be caused by either bacterial, fungal, or viral pathogens [1]. Study has indicated that about 25% of world crop production is lost due to diseases year after year [2]. In places like Panama, the economic impact of loss from diseases has resulted in whole plantations being abandoned [3]. These losses are significant and they lead to financial and economic losses on the part of the farmers. Similarly, it could lead to famine that can subsequently lead to loss of life. Literature showed that potato blight in Ireland (1845-1846) and the Bengal famine epidemic (1943) are a few of the devastating socio-economic repercussions of plant diseases [3].

Plant diseases manifest in physical changes in the parts of the plant like roots, leaves, stems, etc. Traditionally, diagnosis is made through observation of changes in certain parts of the plants (i.e. color of the leaf), and this allows expert observers to assess the type of plant disease and its severity. Visual assessments have been successfully applied over the years [4]. However, diagnosis made via optical observation of physical characteristics of the plant has a considerably high degree of complexity [5]. This high degree of complexity means that experienced agronomists and plant pathologists might misdiagnose the plant diseases [5]. The accurate and early diagnosis of the disease is important to the disease management efforts as they are the preliminary steps that are required before an efficient disease management plan is formulated. If a diagnosis is wrong or if it is made late, this can result in a poorly coordinated disease management plan that will inevitably lead to loss of agricultural crops. An effective system for disease diagnosis is important, as this would help anybody regardless of experience from the novice farmer to the experienced agronomist when diagnosing infected plants from the optical observation of the leaves [6]. Also, the ease of accessibility of an efficient diagnostic system to farmers in the form of a mobile application will be of great advantage as this would be particularly useful in regions of the world with infrastructural and financial challenges that limit their access to specialized equipment like microscopes and DNA sequencing-based methods that are used for early diagnosis.

Machine learning systems have experienced tremendous growth in the last couple of years due to the increased power of computation systems such as Graphical Processing Units (GPU) systems that have allowed for the growth and development of new methodologies and models. One of those is deep learning systems [7], which are artificial neural network (ANN) architectures that have a significant number of layers for processing as against traditional neural network methodologies that do not possess many processing layers [5].

Deep learning, a subset of machine learning, has recently attracted a lot of attention due to its ability to analyze patterns (unsupervised) and make classifications (supervised) better than other models [8]. Deep learning models have been used successfully in multiple domains like image recognition [9], data mining [10], and other applications that require high degree of processing. Deep learning models based on Convolutional neural networks (CNNs) have also been utilized

Manuscript received March 22, 2021; revised September 9, 2021

A. O. Adedoja is a postgraduate student at the Department of Computer Systems Engineering, Tshwane University of Technology, Pretoria, South Africa. (e-mail: damdey@yahoo.com).

P. A. Owolawi is the Head of Department of Computer Systems Engineering, Tshwane University of Technology, Pretoria, South Africa. (e-mail: owolawipa@tut.ac.za).

T. Mapayi is a lecturer at the Department of Computer Systems Engineering, Tshwane University of Technology, Pretoria, South Africa. (e-mail: mapayit@tut.ac.za).

Chunling Tu is a lecturer at the Department of Computer Systems Engineering, Tshwane University of Technology, Pretoria, South Africa. (e-mail: duc@tut.ac.za).

in the domain of agriculture for use cases like counting of fruits [11], detection of fruits [12], the recognition of plants [13], and particularly in use for diagnosis of diseases [5, 6, 14-17]. CNNs make up some of the most powerful techniques to model complex processes and for pattern recognition in applications that use huge amount of data like image pattern recognition. This is because CNNs possess an automatic features extraction capability that allows them to extract the images from the input images without any feature engineering of any kind, and thereby avoiding complex preprocessing on images [18]. According to Lee, et al. [19], applications using machine learning in developed countries is greater than 60% and is already being used for novel purposes like its use for diabetes diagnosis [20] to its uses in mobile voice recognition and dialog systems like Apple's Siri, the Google Assistant, Amazon's Alexa and Microsoft Cortana [21]. Its usage must be therefore be encouraged for plant disease diagnosis domain as many farmers around the world do not have access to the sophisticated technologies needed for the fast and efficient diagnosis required for early plant disease detection. Also, with the penetration of mobile phones in recent years globally, farmers in the developing nations can have access to cellphones that can be leveraged in the near future for use as a diagnostic tool for farmers [22].

The remainder of this paper is organized as follows. In Section II, the literature review that briefly describes past research work done in the domain is presented. Section III details the materials and methods, with the datasets and the model used in this paper. In Section IV, the experimental setup is presented, and the experimental results are discussed, while the conclusion is drawn in Section V.

II. LITERATURE REVIEW

Plant diseases can be confirmed using disease detection techniques and there are numerous ways in which plant diseases can be detected and diagnosed. These techniques can be broken down into two, the direct and indirect methods. The direct methods consist of the methods that make a scientific diagnosis using serological and molecular methods. The indirect methods consist of biomarker-based and plant properties/stress-based disease detection [23]. Various parts of a plant can manifest the physical characteristics that are evidence of vegetable pathologies. Although a lot of research work has been done into diagnosing plants using parts of the plants like the roots [24], kernels [25], fruits [26] [27], stems, and leaves, this research work concentrates on leaves.

One of the earliest successful deep learning architectures plant disease classification was the AlexNet, and it was developed by Krizhevsky, et al. [28]. The authors trained a large CNN using datasets from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competitions, the ILSVRC-2010, and ILSVRC-2012 subsets, and they contained over 1.2 million images. The authors used a neural network that consisted of five different convolutional layers, some with max-pooling layer and three layers that are all fully connected with a 1000-way softmax. The neural network used had 60 million parameters, 650,000 neurons. The size of the database used made overfitting a possibility. Hence the authors used the 'dropout' regularization method to prevent overfitting. The authors were able to get top-1 and top5 error rates of 37.5% and 17.0%, which was a lot better than others at that time. The research study showed how successful deep learning could be when applied to images. The research work presented is similar to the one conducted by Mohanty, et al. [6] on plant disease diagnosis. The authors compared the performance of two deep learning models, GoogLeNet and AlexNet in detecting plant diseases, the researchers used a public dataset that contained 54,306 healthy and diseases images of plant leaves to identify 26 diseases and 14 different crop species to achieve an accuracy of up to 99.35% on a test set. However, the images used to train the plant diseases were captured in controlled conditions and not in actual real conditions in the field. Cap, et al. [29] also developed a system for on-site plant leaf disease detection using deep learning, the authors focused on the leaf localization method using wide-angle images taken on-site as against traditional narrow ranged images that contained one or limited number of targets in the image. The authors believed that deep learning diagnosis done on narrow range images are not practical for real life scenarios where the angle of capture might be wide and contain multiple number of targets. The authors were able to detect 'fully leaf regions' from wide range images and make a diagnosis using a CNN model trained on a dataset of 60,000 images gotten from the Saitama Agricultural Technology Research Center, Japan. The authors were able to achieve a performance of 78.0%. Khan, et al. [30], in a study for real-time plant health assessment, used transfer learning to accurately detect leaf diseases. The authors used a dataset that included images from fruit trees (apple, grape, peach, and strawberry) and vegetable plants (potato and tomato) and used the Amazon Web Services (AWS) machine learning services for training and deployment. To train the model, the authors used the AWS Sagemaker, which is a cloud-based environment for training and testing. The proposed model is called the DeepLens Classification and Detection Model (DCDM) and is based on a Deep Convolutional Neural Network (DCNN). After training, the model is then deployed on an Internet of Things (IoT) device called the AWS DeepLens which is a DL based camera equipped with 4 Mega-Pixel sensors used for ML related project implementation. The authors evaluated the performance of the DCDM architecture and compared it against other CNN architectures including DenseNet, DarkNet, ResNet-50, AlexNet, VGG-16, VGG-19, SqueezeNet. The DCDM architecture with an average accuracy of 98.78% on test images achieved an improved performance over the other architectures and was also better in terms of its computational processing time.

There has been some efforts to harness the power of mobile computing alongside deep learning for the use of plant disease recognition. Wang, et al. [31] decided to tackle the challenge of early disease diagnosis problem in plants using mobile applications. The authors underlying diagnosis technology was an image processing algorithm based on candidate hotspot detection used alongside statistical inference method. This technology was used by a mobile application that captured images and had the image diagnosed online and returned a result. The approach used by the authors however required a considerable amount of feature engineering. Valdoria, et al.

[32] developed an android application that detected plant diseases on terrestrial plants in the Philippines and achieved an accuracy of 80% on the developed app. Pan, et al. [33] in research to develop a smart mobile diagnosis system used densely connected convolutional networks to develop an intelligent diagnosis system for citrus plant diseases that used mobile services computing. The authors built an image dataset that contained citrus diseases of six different varieties and trained a densely connected convolutional networks (DenseNet). The developed system was deployed as an applet on the WeChat platform where the users can upload images and get a response with diagnosis of the uploaded image. The results had the accuracy exceeding 88%. Elhassouny and Smarandache [34] proposed the use of a smart mobile application model that utilises deep CNN to recognize tomato leaf diseases. The model was trained on 7176 images of tomato leaves to recognize the 10 most common types of tomato leaf diseases. The deep CNN implemented in the study, called MobileNets, is optimized for mobile devices that reduces the amount of computation in the first layers.

TABLE I
Quantitative Data of Dataset Images

Plant	Images (Number)
Apple	3172
Blueberry	1502
Cherry	1906
Corn	3852
Grape	4063
Orange	5507
Peach	2657
Bell Pepper	2475
Potato	2152
Raspberry	371
Soybean	5090
Squash	1835
Strawberry	1565
Tomato	18162

Mrisho, et al. [35] presented a deep learning model called Nuru. Nuru was developed by PlantVillage as a simple and inexpensive way for in-field diagnosis of the viral cassava diseases–CMD and CBSD without the use of an internet connection. Nuru is especially useful in rural places around the world with low internet penetration. In the study, the authors evaluated the diagnostic capability of Nuru against agricultural extension officers and farmers in the diagnosis of cassava diseases. Nuru was able to diagnose symptoms of cassava at an accuracy of 65%, higher than those of agricultural extension officers (40–58%) and the farmers (18–31%). This app shows the potential of the use of deep learning in plant disease diagnosis using mobile applications and the comparison of effectiveness of the developed model against farmers and agricultural extension officers.

III. MATERIALS AND METHODS

In this research project we aim to develop a mobile application that uses deep learning to diagnose plant diseases.

A. Datasets

The PlantVillage dataset is a dataset that contains 54,309 images of both healthy and diseased plants. The dataset contains images of size 256x256 of 14 crops and 38 different healthy and diseased plants. The images in the dataset were taken at experimental research stations with association with Land Grant Universities in the USA (e.g. Florida State, Penn State amongst others) [36]. The crop species in the dataset are Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Bell Pepper, Potato, Raspberry, Soybean, Squash, Strawberry, and Tomato. The dataset contains three varieties of the images namely: colour, grayscale and segmented.

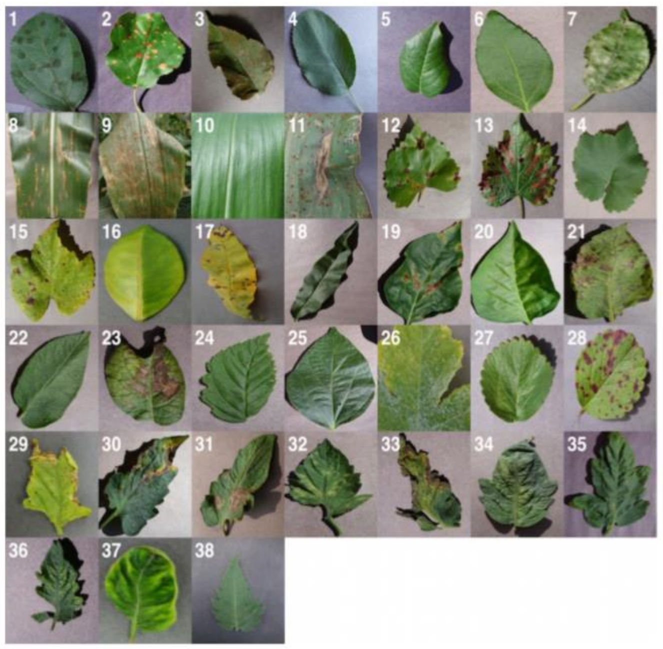


Fig. 1. Sample Images from the PlantVillage Dataset

This research work uses the colour images in the dataset, as previous research works have demonstrated that the segmented and grayscale versions do not improve the performance of the trained model.

Fig. 1 shows sample images of the database and table 1 gives an overview of the dataset. The entire database is split into training and validation sets, by randomly splitting the 54,309 images. The split used is 80/20 with 80% forming the training set, and 20% form the validation set. The 80/20 split is used mostly in neural network applications, other splits (like the 70/30, 75/25) do not have sufficient impact on the performance of the developed model [37]. This study utilizes 43, 447 for training the NASNet-mobile model and 10862 are used to validate the performance of the model for the plant leave images.

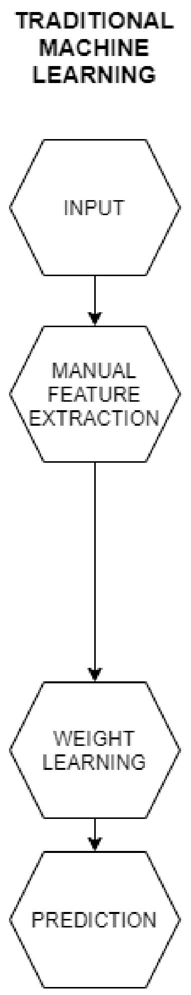


Fig. 2. Difference between traditional and convoluted neural network

B. *Deep learning-based classification*

Machine learning use-cases require the need for creating a classifier, and then applied the classifier to the extracted features from their experiment dataset. This is easily achieved in normal use-cases, but for complex tasks like disease recognition, the feature engineering required consumes immense time and effort. The automatic extraction of features is what makes deep learning models stand out. Fig. 2 shows the differences between the traditional machine learning approach and deep learning approach. Deep learning approach is built on artificial neural networks (ANN). Neural networks constitute layers of multiple neurons; connections are made between neurons in adjacent layers. The neurons must learn to convert a mapped input already pre-processed and pre-extracted featured into its corresponding output. Although a traditional deep learning architecture can extract features automatically, there was a need for CNN to reduce the number of potential parameters present in the neural network, and also train the models efficiently in less time. CNNs are a class of neural networks that allow for low variations in inputs and need low preprocessing before executing [38]. Fig. 3 shows the difference between a common neural network and a convoluted neural network. A convolution layer in the neural network allows for the possibility to process images regardless of size or complexity with fewer parameters. This is done because the number of weights that the network needs for training is determined by the number and the size of the convolution kernels, but not the number of features or weights or image size.

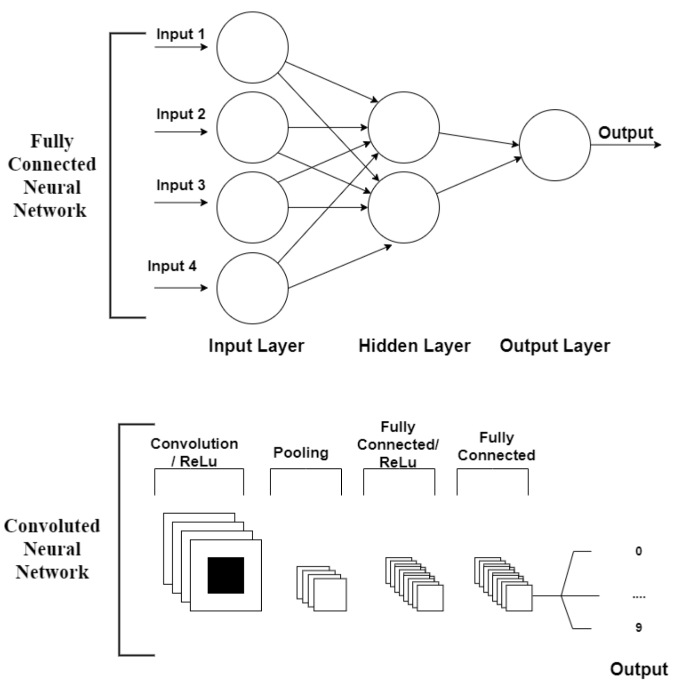


Fig. 3. Fully Connected Neural Network & Convoluted Neural Network.

CNNs are made up of three different parts, convolution, pooling, and fully connected layers and a detailed description of this is shown fig. 4. Feature extraction is carried out in the convolution and the pooling layers with the fully connected layer for classification.

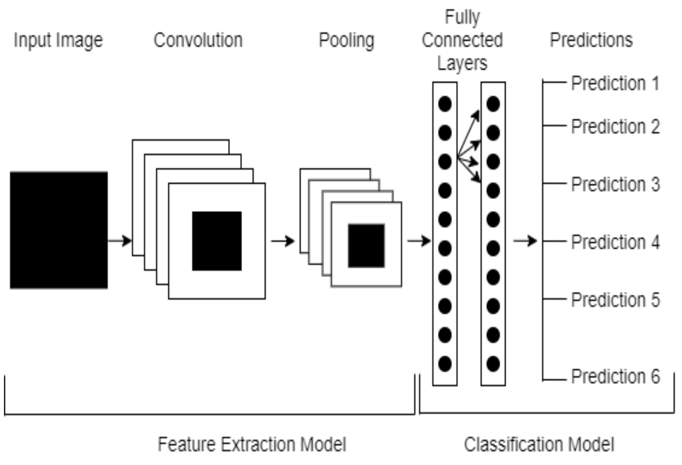


Fig. 4. CNN Model in detail

1) NASNet

NASNet is a model that was developed by the Google ML group in 2017 while working on novel ways to build ConvNets, and It is based on the Neural Architecture Search (NAS) idea conceptualized by the group [40].

2) NAS

The Neural architecture search (NAS) method developed by [41] is a method used to find the best architectures based on gradients. Zoph and Le [41] observed that a neural network's connectivity and structure can be specified by a string of variable length. This makes it possible to generate the string using a recurrent network that acts as "The controller", with string as a representation of "Child Network".

The child network is then trained on the real data with accuracy on the validation set of the data produced. Using the accuracy as a reward signal, the policy gradient is computed to update the controller as seen in fig. 5. As such over the next iterations, the controller learns and gives higher probabilities to architectures of higher accuracy thereby returning strings (child networks) with higher accuracies. Using NAS, Zoph and Le [41] were able to design a novel ConvNet model that performed better than most human-designed architecture. The resulting model tested was able to achieve a test error rate of 3.65, which is 0.09 percent better and 1.05x faster than the previous model that was designed using a similar architectural scheme.

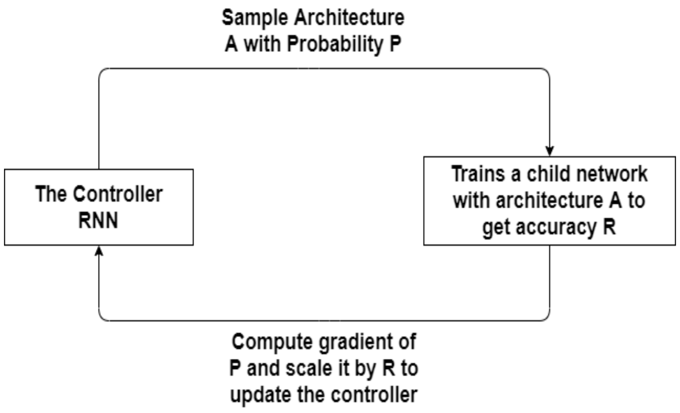

Sample Architecture A with Probability P
The Controller RNN
Trains a child network with architecture A to get accuracy R
Compute gradient of P and scale it by R to update the controller

Fig. 5. Representation of the Neural Architecture Search

3) Composition of NASNet

NASNet is a CNN architecture constructed using the scalable NAS method afore-mentioned and the approach of the Google ML group was based on reinforcement learning. There is a parent AI, a Recurrent Neural Network (RNN) "The Controller" that reviews the efficiency of the child AI "Child Network" in a CNN and adjusts the architecture of the "Child Network". These adjustments are made on the number of layers, the regularization methods, weights and more, are used to improve the efficiency of the "Child Network" as seen in fig. 6. The operational blocks available to the controller RNN to build the child network is listed below:

- Identity
- 1 × 3 then 3 × 1 convolution
- 1 × 7 then 7 × 1 convolution
- 3 × 3 dilated convolution
- 3 × 3 average pooling
- 3 × 3 max pooling
- 5 × 5 max pooling
- 7 × 7 max pooling
- 1 × 1 convolution
- 3 × 3 convolution
- 3 × 3 depthwise-separable—convolution
- 5 × 5 depthwise-separable—convolution
- 7 × 7 depthwise-separable—convolution

Using all these operational blocks, the RNN builds the NASNet architecture. The architecture is trained with two different image sizes to produce the two different types of NASNet architectures, the NASNetLarge and the NASNetMobile. The NASNetmobile is a lot more reliable than the NASNetLarge because of the difference in the parameters - 53,26,716 parameters to the 8,89,49,818 parameters of NASNetLarge [40]. Every NASNet architecture has a block as its smallest unit. A cell is a combination of the blocks and is formed by concatenating various operational blocks like those stated above and multiple cells form the NASNet architecture.

The controller RNN optimizes the cells with blocks, and as such are not fixed because they are optimized for a selected dataset. Every single block is an operational module and the operations that can be performed by a block include the following:

- Convolutions
- Max-Pooling
- Average-Pooling
- Separable Convolutions
- Identity Mapping, inter alia

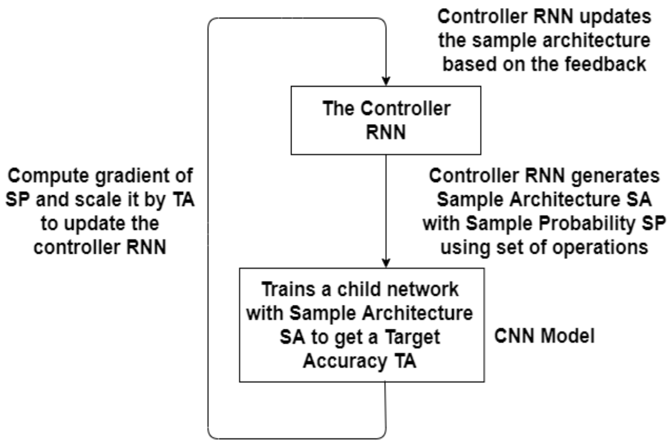

Controller RNN updates the sample architecture based on the feedback
The Controller RNN
Compute gradient of SP and scale it by TA to update the controller RNN
Controller RNN generates Sample Architecture SA with Sample Probability SP using set of operations
Trains a child network with Sample Architecture SA to get a Target Accuracy TA
CNN Model

Fig. 6. The role of the Controller RNN in NASNet architecture

Each block maps the present and the previous input (H0 and H1) to a single output feature map as seen in fig. 8. NASNet uses the element-wise addition which is much more intuitive than vector wise operations. Two types of convolutional cells are used when using a feature map as an input and they are:

- Normal Cells: These are convolutional cells that usually return feature maps of the same dimension. For example, if a cell allows an input of a block that has a feature map of size H × W with a stride of 1, the output calculated will ultimately be the same size as that of the feature map.
- Reduction Cells: These are also convolutional cells that return feature maps, with the height and width of the feature map reduced by a factor of two (e.g. if the stride is 2, the size is reduced by 2) [42].

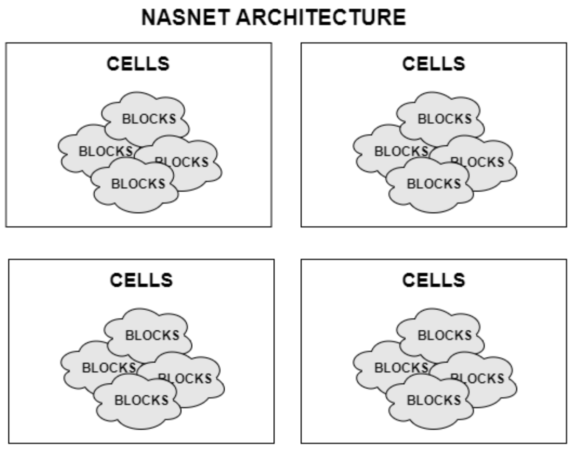


Fig. 7. Taxonomy of a NASNet Architecture

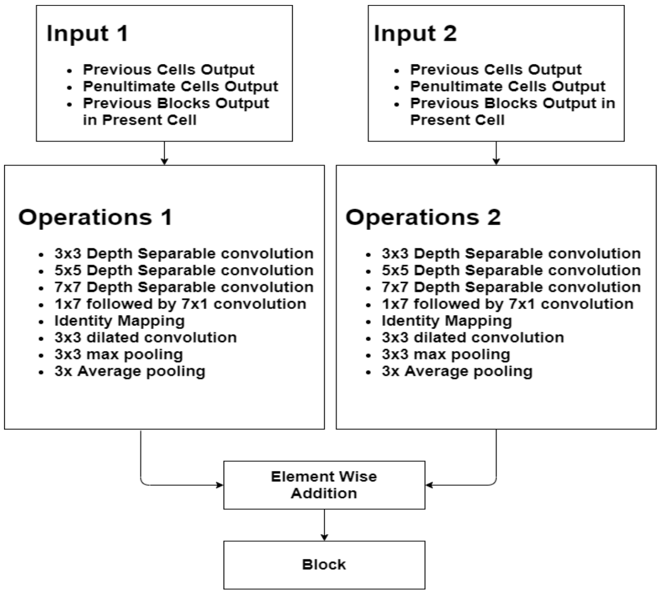

Input 1
Previous Cells Output
Penultimate Cells Output
Previous Blocks Output in Present Cell
Input 2
Previous Cells Output
Penultimate Cells Output
Previous Blocks Output in Present Cell
Operations 1
3x3 Depth Separable convolution
5x5 Depth Separable convolution
7x7 Depth Separable convolution
1x7 followed by 7x1 convolution
Identity Mapping
3x3 dilated convolution
3x3 max pooling
3x Average pooling
Operations 2
3x3 Depth Separable convolution
5x5 Depth Separable convolution
7x7 Depth Separable convolution
1x7 followed by 7x1 convolution
Identity Mapping
3x3 dilated convolution
3x3 max pooling
3x Average pooling
Element Wise Addition
Block

Fig. 8. Block formation in NASNet architecture

The development of the network is based on three different factors:

- Cell structure
- Number of cells to be stacked (N)
- Number of filters in the first layer (F)

The values of N and F are fixed in the initial stages of the search. However, the values of N and F in the first layer are changed to alter the depth and width of the network. As soon as the search is completed, models are constructed of various sizes to fit the datasets. The cells are then connected in the most optimized structure to create the best NASNet architecture possible. The variations in convolutional nets are the differences in the normal and reduction cells which are searched by the controller RNN. The structures can be searched in the search space as seen in fig. 10. In the search space, each cell is connected to two input hidden states. A sample of hidden states can be seen in fig. 9. Hidden layers are then formed using pairwise combinations and then updated by concentration. Hidden layers can also undergo convolution and pooling operations. Using the results from the optimization and the best cells are then selected in the NASNet architecture. This makes the search faster and generalized features can be obtained.

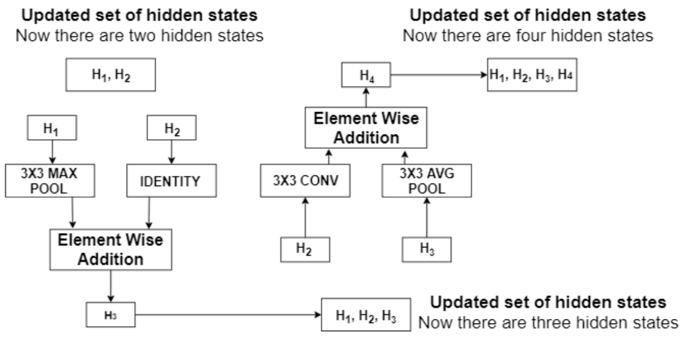

Updated set of hidden states
Now there are two hidden states
Updated set of hidden states
Now there are four hidden states
H1, H2
H4
H1, H2, H3, H4
H1
H2
Element Wise Addition
3X3 MAX POOL
IDENTITY
3X3 CONV
3X3 AVG POOL
Element Wise Addition
H2
H3
H3
H1, H2, H3
Updated set of hidden states
Now there are three hidden states

Fig. 9. Hidden state formation inside a block

4) *Reinforcement Learning*

NASNet trains with reinforcement when an accuracy R is achieved on a dataset. The accuracy R is used as the reward signal, using reinforcement learning to train the RNN controller. To find the optimal architecture, the controller is asked to maximize its expected reward, represented by J(θc) as shown in equation (1).

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R] \tag{1}$$

The reward signal R is non-differentiable. A gradient policy is used to iteratively update the expected reward θc. The reinforce rule is used as indicated in equation (2).

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}\left[\nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R\right] \tag{2}$$

An empirical approximation of the above quantity is calculated in equation (3).

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)R_k \tag{3}$$

where m is the number of varied architectures sampled by the controller in a singular batch and T is the number of hyperparameters that the controller would predict for the design of neural network architecture. Rk is the validation accuracy that the k-th neural network architecture receives after being training on a specific training dataset. The approximation in equation(3) is for the gradient. However, it has the downside of having high variance. To reduce the variance of the estimate, a baseline function described in (4) is used.

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)(R_k-b) \tag{4}$$

Baseline b is the exponential moving average of the accuracies of the architecture in the preceding batches.

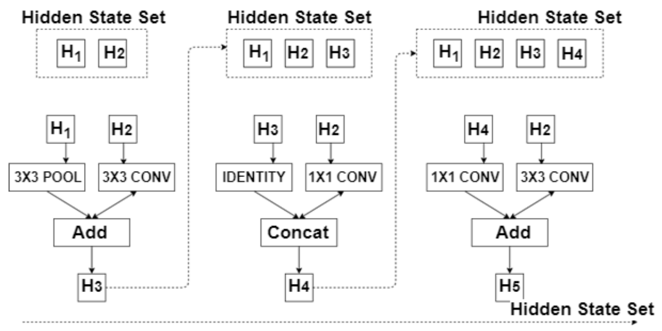


Fig. 10. NASNet Search Space Schematic Diagram

C. System Architecture

The system architecture is shown in fig. 11. The training and inference stages contain different components. The training stage is the part where the NASNet model is trained using transfer learning. After it has been trained, the trained model is downloaded. This trained model is loaded into a flask-powered microservice. Flask is a python microservice that is very lightweight and has only one dependency–Python, and this makes it easy to start up. This microservice exposes an API that takes in an input image and makes a diagnosis on the image using the saved trained model. This microservice is deployed on one of Amazon Web Services (AWS) cloud-based services called the Elastic Compute Cloud (EC2) that offers state-of-the-art computing and storage facilities. EC2 provides virtual machines called instances that users can rent and scale at will [43].

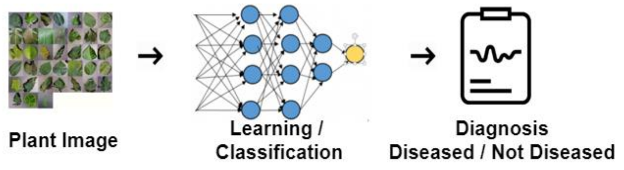


Fig. 11. Proposed System Architecture

A client mobile app is developed using React Native. React Native is a JavaScript framework, developed by Facebook to build native mobile application using JavaScript. It facilitates, the writing of application codes using JavaScript, and it would be compiled to native code. It is based on ReactJS, a library also developed by Facebook [44]. Applications developed using React Native can be deployed on iOS and Android devices. The client mobile app is used to capture the image using either the phone camera or choosing an already captured image. The captured image is then sent in a request to the web service. The web-service receives the image, makes a diagnosis, returns a result with its classification and this is displayed on the mobile app interface.

IV. Experimental Setup and Results

In this section, the training model for the deep learning architecture and the proposed system is discussed in detail and the experimental results are also presented. The technical resources used for the research work are also presented in this section. The proposed approach is evaluated by conducting a series of experiments using the PlantVillage dataset mentioned earlier. Below are the details of the experiments.

A. *Dataset*

The NASNet model is trained on the PlantVillage dataset. The images used are the colour images, the grayscale images are not considered for this research project as other experiments [6] have shown that they do not improve the accuracy of the model.

The dataset will be pre-processed in several forms.

1. **Image Resizing:** The images are resized, and they are in a square-shaped input as it is the preference of a lot of Deep Learning architectures.
2. **Data Augmentation:** Images are augmented to generate more training data. The size of a dataset has a huge impact on the model accuracy, which means more data, better results. Techniques such as rotate, flip, lighting change, and picture enlargement are applied to the images in the dataset as done by [45].
3. **Image Filtering:** These are techniques to modify or enhance an image. This research project uses a couple of techniques: An average filter that is a simple sliding window spatial filter that is used to reduce noise in images. It works by replacing the centre value in the window with the average of the pixel values. Gaussian filter is applied on the image by using a Gaussian function; median filter that is a sliding window spatial filter is applied such that the centre values in the window are

replaced with the median of all pixel values in the window.

B. *Model*

The model used in this study is the NASNet model and it has two variants, NASNetMobile and NASNetLarge. NASNetmobile is used in this paper. This is because of its reliability when compared with the NASNetLarge and NASNetmobile has 5326716 parameters when compared to the 889,49,818 parameters in NASNetLarge [40].

C. *Implementation*

The implementation is divided into two parts, the training of the model and the deployment of the client mobile application that does the diagnosis. The training/test dataset is an 80-20 split. The model is implemented in python using the deep learning library fast.ai [46] that allows for the use of GPU acceleration. Fast.ai provides high-level deep learning components that allow for quick prototyping to provide state-of-the-art results in deep learning domains. The experiments are performed on the Google Colaboratory platform (Colab). Colab is a free Google service developed for artificial intelligence developers. The service runs on jupyter notebooks and it provides free access to GPU servers and Python 2/3 development environment. To train the network using the fast.ai framework, the following steps as illustrated in Algorithm 1 are involved:

- Clone the dataset from the GitHub repository.
- Create a data bunch by choosing the following
- Set the path for the input dataset
- Set the ratio of training/validation sets
- Label the inputs
- Set the batch size to be used.
- Set the kind of data augmentation to be used.
- Select the number of CPUs to be used.
- Import the pre-trained models and define them.
- Create a learner object from the data object and the pre-trained model, define the metrics expected (e.g., accuracy, error etc.), together with any callback functions needed (e.g. CSVLogger to log the results of the training).
- Set the number of epochs, then train
- After training, metrics are returned with proper values.

The mobile application is to be developed in React Native. After the completion of the development, an APK is generated to be tested on an android device and is also tested on an iOS device. The following technologies are used in the development of the mobile application:

1. **React Native** – React Native is an open-source mobile application framework created by Facebook. It is used to develop applications for Android, iOS, Web, and UWP by enabling developers to use React.
2. **Flask -** Flask is a micro web framework for Python. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. In this study, it is used to power the microservice that serves the mobile application. Flask is used for rendering an API that the mobile app consumes. It then connects with the DL model to gain the prediction.
3. **Cloud Server** – This is the server where the Flask application is going to reside, Since it is web-accessible, it has cross-platform support (i.e. iOS, Android, UWP), code reusability, and maintenance. It reduces the computational load of the DL model off the phone and into a scalable cloud service.

ALGORITHM 1: TRAINING ALGORITHM

Algorithm for training

```
Input:
Dataset from GitHub Repository
Process:
    1. Create data bunch
        • Batch size
        • Training Data : Validation Data
        • Dataset path
        • Data Augmentation Technique
        • Number of CPUs
    2. Import model to be used
    3. Create Learner Object
    4. Define Metrics to be expected
    5. Set number of epochs
    6. Train
Output:
Trained model with accuracy results
```

D. *Model Training*

Transfer learning is used to train an already pre-trained NASNet model that was trained on the ImageNet dataset. Below are the different hyperparameters used and the experimented values.

- **Number of epochs:** An epoch is a full run of the training dataset through the model currently being trained. This is to learn the appropriate number of epochs that allows the network to converge properly.
  Baseline: Five
- **Dropout rate:** Various dropout rates are used all the way from 0.0 to 0.9, with 0.1 increments.
  Baseline: 0.5
- **Validation/Training Data Ratio:** 1:9 to 9:1.
  Baseline: 2:8
- **Batch Size:** Various batch sizes are tested from an initial 8 to 64. A training dataset can be divided into one or more batches. This is important if you cannot fit your whole data into the memory of the machine that is processing the model. It is the number of samples that will be passed into the model at once.
  Baseline: 32

- **Input Image Size:** This is the size of the input images, Sizes from 128 x 128 up to 256 x 256 will be tested.
  Baseline: 256 x 256.

E. *Mobile Application Deployment*

After the model has been trained successfully, it is downloaded to an already developed python flask microservice that is deployed on an AWS EC2 instance with Docker as described in fig. 12. The flask microservice renders an API endpoint that takes in a request that contains the leave image. The leave image is saved and gets a diagnosis from the trained NASNet model and the feedback of the request is sent back. The feedback of the request is received by the mobile app and then displayed on the user-interface of the app.

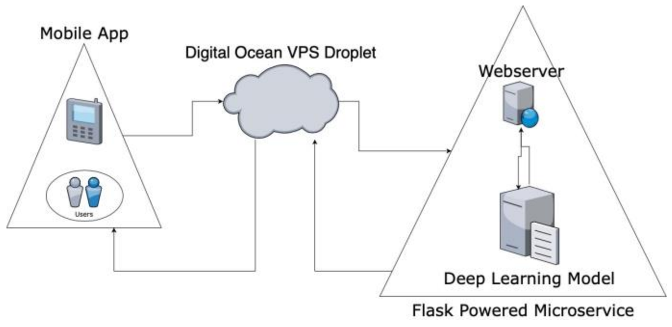


Fig. 12: Diagnosis system design

Steps involved in the diagnosis:

- Mobile app sends a request with the image of the leaf image to the server.
- The Droplet contains the microservice. The webserver on the microservice gets the request, with the leave image.
- The web-server sends the leave image to the deep learning model for prediction.
- The resulting prediction that decides the state of the plant leaf, as either diseased or healthy, is sent back to the web server, which then sends the prediction to the mobile app.

F. *Results*

The NASNet model to be trained is trained on a variety of afore-mentioned parameters. Additionally, the mobile application developed is tested on 20 randomly selected images of varying sizes for its response time. Below are the results of the experiments.

TABLE 2: BASE PARAMETERS

Hyperparameter	**Value**
Epochs	5
Dropout Rate	0.5
Batch Size	32
Input Image Size	256x256
Validation/Training Data Ratio	2:8
Learning Rate	0.01
Filtering	None

TABLE 3: RESULT FROM BASE PARAMETERS

Training Loss	**Validation Loss**	**Accuracy**
0.151575	0.099513	0.965749

TABLE 4: PERFORMANCE COMPARISON OF FILTERING ON ACCURACY

Filter Size	**Training Loss**	**Validation Loss**	**Accuracy**
No Filter	0.170315	0.113067	0.963079
Average Filter			
3x3	0.153105	0.104462	0.964736
5x5	0.174656	0.133590	0.958107
10x10	0.253591	0.164824	0.943744
Gaussian Filter			
3x3	0.164019	0.102222	0.964000
5x5	0.160866	0.114305	0.960409
7x7	0.184944	0.117713	0.961974
Median Filter			
3x3	0.148121	0.100840	0.966854
5x5	0.182138	0.126222	0.956910
7X7	0.201356	0.131876	0.956818

Experiment 1: Initial setup and baseline hyperparameters

The model was trained on an initial baseline set of hyperparameters that are derived from similar research by Tiwari and Richmond [14] on techniques and methodologies on identifying crop diseases. This is going to be used as a benchmark for the other test results to see the effect of the changes of the hyperparameters on the performance accuracy rate.

Table 3 shows the results of the experiment. The initial learning rate is set to 1e-2, with the model trained for 5 epochs, at a dropout rate of 0.5, a batch size of 32, with input image size of 256x256 pixels, with the validation/training data ratio of 2:8. With the initial model training, an accuracy value of 96.57%, training loss of 0.151575 and validation loss of 0.099513 are achieved.

Experiment 2: Initial setup and baseline hyperparameters.

Experiments are carried out to investigate the effect of these filters on accuracy. Although the baseline test does not use any filter, the filter-based experiments used the averaging, Gaussian, and median filters with varying windows.

Table 4 shows the results of the experiment, with the accuracy values of 96.47% achieved for the average filter (3x3 window), 96.40% for the Gaussian filter (3x3 window) and 96.68% for the median filter (3x3 window) as against a value of 96.30% with no filter. Fig. 13 shows a graphical representation of the results obtained.

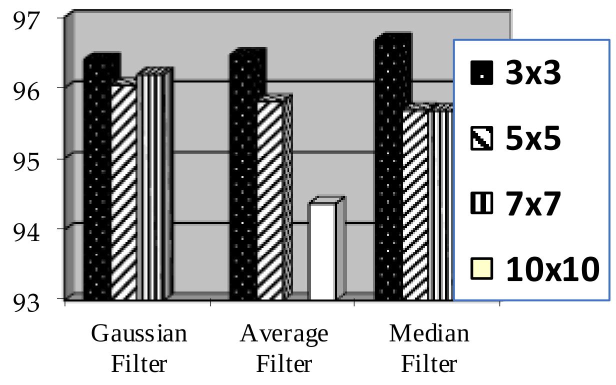


Fig. 13. Graphical representation of filter experiment results

Experiment 3: Ratio of validation data to training data results.
The effect that the split of the validation, the training data ratio has on the accuracy is also examined. The baseline parameter ensures the splitting in a 2:8 ratio. Such is progressively adjusted up until it reaches a 9:1 ratio while holding other baseline set of hyperparameters constant.

TABLE 5: PERFORMANCE COMPARISON OF PERFORMANCE OF DIFFERENT RATIO TRAINING/VALIDATION DATA RATIO FOR THE DATASET USED.

Training	Training Loss	Validation Loss	Accuracy
0.2	0.157282	0.101149	0.966486
0.3	0.170966	0.123526	0.959057
0.4	0.191113	0.1293114	0.957002
0.5	0.196933	0.147369	0.950685
0.6	0.229605	0.166938	0.944235
0.7	0.262834	0.192039	0.937890
0.8	0.297685	0.242702	0.921416
0.9	0.436562	0.343948	0.890433

The best accuracy rate and validation loss achieved is on the baseline parameter of a 2:8 ratio; this achieved a 96.64% accuracy value. This accuracy value progressively reduces as the splitting ratio is continuously adjusted in the experiment. The last ratio of 9:1 achieved the least accuracy value of 89.04% in the experimental accuracy performance set. Table 5 shows the performance and fig. 14 shows a graphical representation of the experimental values.

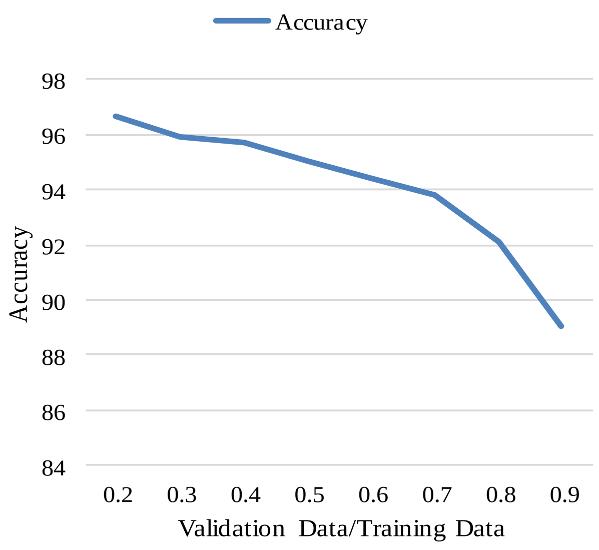


Fig.14. Graphical Representation of performance of different ratio training/validation data ratio for the dataset used.

However, the first two dropout rate, 0.1 and 0.2 produce models that are overfitted with their validation loss greater than their training loss. The closest accuracy rate that is not overfitted is a dropout rate of 0.5. Such is consistent with the baseline value suggested and leading to maximum regularization.

Table 6 shows a detailed representation of the results of the experiment. Fig. 15 shows a graphical representation of the experiments.

TABLE 6: PERFORMANCE COMPARISON OF DROPOUT RATE ON ACCURACY

Dropout rate	Training Loss	Validation Loss	Accuracy
0.1	0.079877	0.081221	0.973943
0.2	0.089208	0.092597	0.970997
0.3	0.161278	0.113834	0.960960
0.4	0.183559	0.128273	0.956404
0.5	0.151575	0.099513	0.965749
0.6	0.187764	0.105838	0.963355
0.7	0.225926	0.122344	0.957555
0.8	0.316459	0.147102	0.951662
0.9	0.479185	0.209871	0.932419

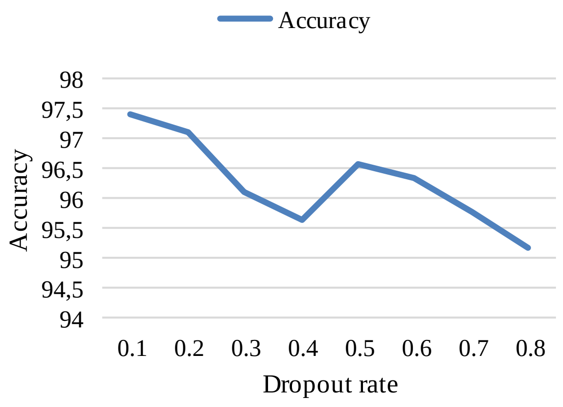


Fig. 15. Graphical Representation of the effect of accuracy against dropout rate

Experiment 5: Image size experiment.

The experiment performed varied the size of the leaf images in the dataset used for the model training and monitored the effect of the variation in the image sizes on the model performance. Although the baseline value of 256x256 image size is used in the experiment, the experiment also investigated the sizes 64x64, 128x128, and 256x256 while holding other baseline sets of hyperparameters constant.

TABLE 7: PERFORMANCE COMPARISON OF IMAGE SIZE ON ACCURACY

Image Size	Training Loss	Validation Loss	Accuracy
64	0.808506	6.208295	0.802320
128	0.322468	0.229584	0.924777
256	0.151575	0.099513	0.965749

Although the next logical image size is 512x512, the training environment is unable to handle training images of this size as it is continually timed out. After the model training, the accuracy values of the model increased as the image size increased and the highest accuracy value of 96.57% achieved is on the 256x256 image sizes, while the lowest accuracy value of 80.23% is achieved on image sizes of 64x64 with the trained model very underfitted. Table 7 shows the values of the accuracy measures on the various image sizes. Fig. 16 shows a graph of the accuracy values against their respective image sizes.

Experiment 6: Batch size Experiment.

The batch size is the number of dataset samples passed into the model, the baseline value being 32. The experimental batch size values investigated ranged from 8 to 64 when holding other baseline sets of hyperparameters constant.

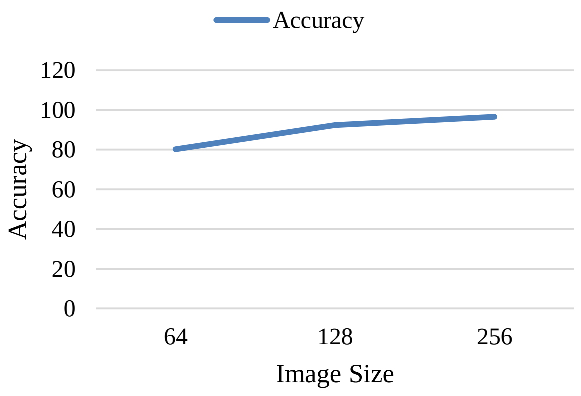


Fig. 16. Graph showing effect of image size on accuracy.

Experimental studies show that the batch size of 32 achieved the highest accuracy value of 96.57%, while the lowest accuracy value of 95.30% is achieved via a batch size of 8. Table 8 details the performance accuracy values obtained in the experimental studies. Fig. 17 shows the graphical representation of the different accuracy values obtained, against their respective batch sizes.

TABLE 8: PERFORMANCE COMPARISON OF BATCH SIZE ON ACCURACY

Batch Size	Training Loss	Validation Loss	Accuracy
8	0.251792	0.182867	0.953043
16	0.171711	0.120710	0.964184
32	0.151575	0.099513	0.965749
64	0.154251	0.110910	0.962619

Experiment 7: Epoch Experiment.

The epoch is used to describe the number of times a dataset passes through the training algorithm. The baseline value of the epoch used is 5. The experiment performed starts with an initial value of 1 epoch, then 5 epoch, and then increases by 5 epochs up until it reaches 35 epochs while holding other baseline sets of hyperparameters constant. The training environment is unable to handle the next value of 40 epochs, constantly timing out. Experimental studies showed that the greater the epoch value, the greater the accuracy value and the lower the validation loss. The highest accuracy value of 99.09% is achieved at 35 epochs, achieving as close to an optimal fit as possible.

The least achieved accuracy is at 1 epoch, and the accuracy value is 91.65%, with the validation loss sufficiently high. The difference in the validation loss and the training loss shows the model to be seriously underfitted at this point. Table 9 details the values obtained after each experimental run. Fig. 18 presents a graphical representation of the epoch values against the accuracy.

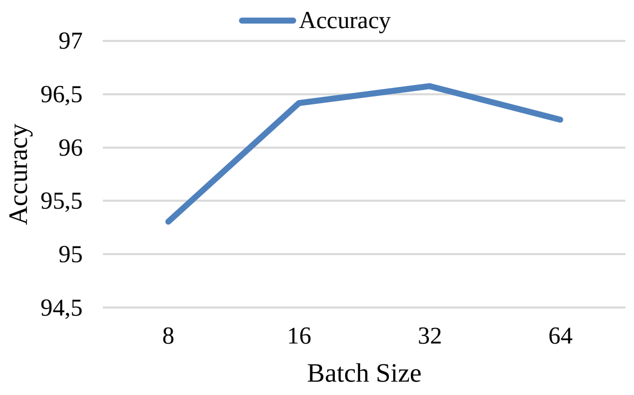


Fig. 17. Graphical representation of accuracy against batch size.

TABLE 9: PERFORMANCE COMPARISON OF THE EFFECT OF VARIABLE EPOCHS ON ACCURACY

Epochs	Training Loss	Validation Loss	Accuracy
1	0.395950	0.279173	0.916582
5	0.151575	0.099513	0.965749
10	0.093075	0.062832	0.978271
15	0.067690	0.044993	0.985913
20	0.553122	0.039122	0.98712
25	0.036893	0.033198	0.988767
30	0.033079	0.026128	0.990793
35	0.026869	0.024630	0.990977

Experiment 8: Optimal Hyperparameters with Filters
Experimental studies are also conducted to investigate the impact of applying the best-performing window sizes of the afore-mentioned filters (Gaussian, median and average filters) when combined with the best-performing parameter values of the proposed deep learning model (see Table 10). The results obtained are compared with the best accuracy value obtained when no filter was used (see Table 11).

Experimental findings showed that the application of the filters has a slight effect on the accuracy of the trained model, with all models trained using filtered images having higher accuracy values when compared to the models trained on unfiltered images. Accuracy values achieved are 99.24% for unfiltered images, 99.28% for average filtered images, 99.31% for Gaussian filtered images, and 99.28% for median filtered images. The proposed deep learning model applied on Gaussian filtered images achieved the highest accuracy value of 99.31%. Fig. 19 shows a graphical representation of the comparison of the best results obtained from the models trained using filtered images with the models trained on unfiltered images using the best hyperparameters.

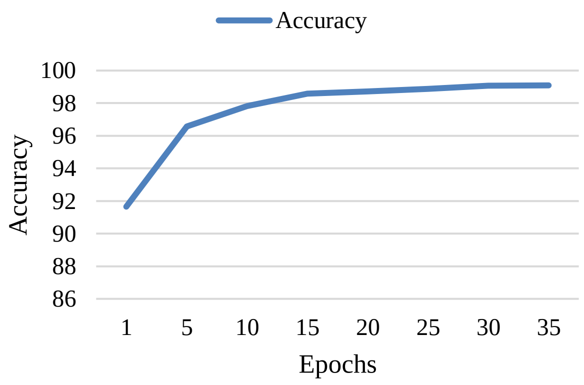


Fig. 18. Graphical representation of different epochs on the accuracy of a model

TABLE 10: BEST PERFORMING VALUE OF EXPERIMENTED PARAMETERS

Parameter	Value
Validation/Training Ratio	2:8
Dropout rate experiment	0.5
Image Size	256x256
Batch Size	32
Epoch	35

TABLE 11: COMPARISON OF THE RESULTS OBTAINED FROM THE MODELS TRAINED USING FILTERED IMAGES WITH BEST PERFORMING WINDOW SIZES AND UNFILTERED IMAGES USING BEST HYPERPARAMETERS

Filter (Window Size = 3x3)	Training Loss	Validation Loss	Accuracy
No Filter	0.027846	0.022291	0.992450
Average Filter	0.038150	0.020694	0.992818
Gaussian Filter	0.035963	0.019991	0.993095
Median Filter	0.036853	0.022398	0.992818

Experiment 9: Mobile Application & Web Service Response Time
The trained NASNetMobile model was then uploaded to a Python-powered Flask web Service deployed to a cloud platform. The web service was developed to accept an image in its request parameters. The image was saved on the cloud server. The service then classified the received image using the trained model. This returned a disease-diagnosis report via the web service as a JSON string. The web service was tested for its response time; and 20 images of varying sizes are tested, as shown in Table 12. Experimental studies showed that the bigger the image size, the longer the response time

required to return the plant-disease diagnosis report. A mobile application was also developed for the plant disease diagnostic system and this was performed via React Native. Fig. 20 and fig. 21 show the screenshots from the mobile application.
The response time of the proposed mobile-based plant disease diagnosis report using web-service is also examined. Table 12 shows the response time in ms.

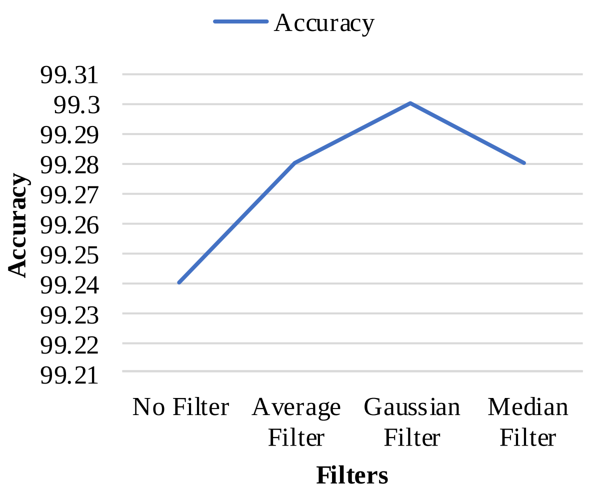


Fig. 19. Graphical representation of the results obtained from the models trained using filtered images with best performing window sizes and unfiltered images using best hyperparameters.

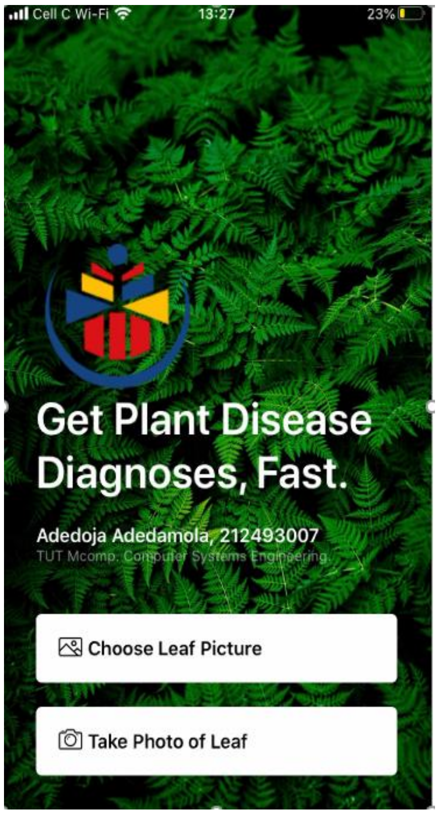
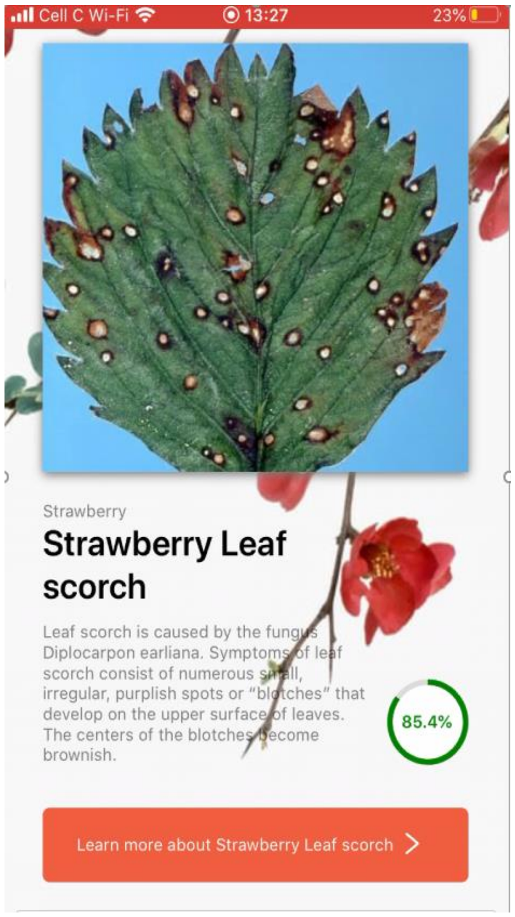

Cell C Wi-Fi
13:27
23%
Get Plant Disease Diagnoses, Fast.
Adedoja Adedamola, 212493007
TUT Mcomp. Computer Systems Engineering.
Choose Leaf Picture
Take Photo of Leaf

Fig. 20. Mobile Application Screenshot.



Leaf scorch is caused by the fungus Diplocarpon earliana. Symptoms of leaf scorch consist of numerous small, irregular, purplish spots or "blotches" that develop on the upper surface of leaves. The centers of the blotches become brownish.

Fig. 21: Mobile Application Screenshot.

TABLE 12: WEB SERVICE RESPONSE TIME IN MS

Image Size (kb)	Response Time(ms)
11	358
12	429
14	291
30	300
35	307
37	361
45	420
49	302
103	386
120	366
126	373
192	505
212	375
279	467
338	369
452	412
1314	630
1408	656
3997	1496
11	358

TABLE 13: PERFORMANCE COMPARISON OF THE MODEL USED IN THIS RESEARCH AGAINST SOME OTHER METHODS IN LITERATURE

Model (Author)	**Accuracy**
AlexNet - [16]	98.66%
AlexNet - [48]	97.49%
VGG16 - [48]	97.29%
PlantDiseaseNet - [47]	93.67%
Inception V3 – black leaf spot - [49]	98%
Inception V3 – red mite damage - [49]	96%
Inception V3 – red mite damage - [49]	95%
Inception V3 – cassava brown streak - [49]	98%
Inception V3 – cassava mosaic - [49]	96%
Proposed NASNetMobile Deep-learning model (unfiltered images)	**99.25%**
Proposed NASNetMobile Deep-learning model (average-filtered images)	**99.28%**
Proposed NASNetMobile Deep-learning model (median-filtered images)	**99.28%**
Proposed NASNetMobile Deep-learning model(Gaussian-filtered images)	**99.31%**

G. *Performance comparison with models in literature*

The results of the model investigated in this study to diagnose plant diseases using images of leaves of those plants are compared with results obtained in previous studies. Table 13 compares the performance of the proposed NASNet model with other methods in the literature. The accuracy values 99.24%, 99.28%, 99.31%, and 99.28% achieved from the proposed NASNet deep-learning model applied to unfiltered images, average-filtered images, Gaussian-filtered images, and median-filtered images, respectively, are higher than the accuracy values of 93.67%, 98.66%, 98%, and 97.49%, obtained in [16, 47-49].

V. CONCLUSION

This research paper has described the use of a state-of-the-art NASNet deep learning architecture using efficient parameters on very domain-specific dataset that is capable of effectively diagnosing plant diseases with a built mobile application. The application used a NASNet-Mobile model using transfer learning and the trained model was deployed to a flask microservice on an amazon EC2 instance, and communicates through the microservice to diagnose diseases using the plant leave images captured through the mobile application. The proposed model achieved promising results of 99.24%, 99.28%, 99.31%, and 99.28%. Although the response time on the API micro-service that serves the diagnosis seems very impressive, it becomes slower when the image-size is bigger. Also, the developed mobile application for remote plant disease diagnosis seems promising because of the capability of fast remote accessibility to plant disease diagnosis it can provide for farmers.

REFERENCES

[1] P. A. J. A. P. P. O'Brien, "Biological control of plant diseases," *Australasian Plant Pathology*, vol. 46, no. 4, pp. 293-304, 2017.

[2] B. Lugtenberg, "Introduction to plant-microbe interactions," in *Principles of Plant-Microbe Interactions*: Springer, 2015, pp. 1-2.

[3] S. Chakrabortya, A. Tiedemannb, and P. J. E. P. Tengc, "Climate change: potential impact on plant diseases," vol. 108, no. 317, p. 326, 2000.

[4] C. Bock, P. Parker, A. Cook, and T. J. P. D. Gottwald, "Visual rating and the use of image analysis for assessing different symptoms of citrus canker on grapefruit leaves," *Plant Disease* ,vol. 92, no. 4, pp. 530-541, 2008.

[5] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers Electronics in Agriculture,* vol. 145, pp. 311-318, 2018.

[6] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in plant science,* vol. 7, p. 1419, 2016.

[7] Y. LeCun, Y. Bengio, and G. J. n. Hinton, "Deep learning," *nature,* vol. 521, no. 7553, p. 436, 2015.

[8] Y. Tan, W. Yan, S. Huang, D. Du, and L. Xia, "Thermal Infrared Human Recognition Based on Multi-scale Monogenic Signal Representation and Deep Learning," *IAENG International Journal of Computer Science,* vol. 47, no. 3, pp540-549, 2020.

[9] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks,* vol. 3361, no. 10, p. 1995, 1995.

[10] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, A. L. García, I. Heredia, P. Malík and L. Hluchý, "Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey," *Artificial Intelligence Review,* pp. 1-48, 2019.

[11] M. Rahnemoonfar and C. J. S. Sheppard, "Deep count: fruit counting based on deep simulated learning," *Sensors,* vol. 17, no. 4, p. 905, 2017.

[12] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. J. S. McCool, "Deepfruits: A fruit detection system using deep neural networks," *Sensors,* vol. 16, no. 8, p. 1222, 2016.

[13] M. Gao, L. Lin, and R. O. Sinnott, "A mobile application for plant recognition through deep learning," in *2017 IEEE 13th International Conference on e-Science (e-Science),* 2017, pp. 29-38: IEEE.

[14] N. S. Tiwari and J. J. B. Richmond, "The development of methodology and techniques for crop disease identification," *bioRxiv*, p. 702621, 2019.

[15] E. C. Too, L. Yujian, S. Njuki, L. J. C. Yingchun, and E. i. Agriculture, "A comparative study of fine-tuning deep learning models for plant disease identification," *Computers and Electronics in Agriculture,* 2018.

[16] M. Brahimi, K. Boukhalfa, and A. Moussaoui, "Deep learning for tomato diseases: classification and symptoms visualization," *Applied Artificial Intelligence,* vol. 31, no. 4, pp. 299-315, 2017.

[17] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic, "Deep neural networks based recognition of plant diseases by leaf image classification," *Computational intelligence neuroscience,* vol. 2016, 2016.

[18] P. Jiang, Y. Chen, B. Liu, D. He, and C. J. I. A. Liang, "Real-time detection of apple leaf diseases using deep learning approach based on improved convolutional neural networks," *IEEE Access,* vol. 7, pp. 59069-59080, 2019.

[19] P. Lee, D. Stewart, and C. J. D. T. T. L. Calugar-Pop, "Technology, media and telecommunications predictions," 2017.

[20] O. Karan, C. Bayraktar, H. Gümüşkaya, and B. J. E. S. w. A. Karlık, "Diagnosing diabetes using neural networks on small mobile devices," vol. 39, no. 1, pp. 54-60, 2012.

[21] A. Luckow, M. Cook, N. Ashcraft, E. Weill, E. Djerekarov, and B. Vorster, "Deep learning in the automotive industry: Applications and tools," in *2016 IEEE International Conference on Big Data (Big Data),* 2016, pp. 3759-3768: IEEE.

[22] J. Amara, B. Bouaziz, and A. Algergawy, "A Deep Learning-based Approach for Banana Leaf Diseases Classification," in *BTW (Workshops)*, 2017, pp. 79-88.

[23] S. Sankaran, A. Mishra, R. Ehsani, C. J. C. Davis, and E. i. Agriculture, "A review of advanced techniques for detecting plant diseases," *Computers and electronics in agriculture*, vol. 72, no. 1, pp. 1-13, 2010.

[24] S. Smith and S. J. F. P. B. Dickson, "Quantification of active vesicular-arbuscular mycorrhizal infection using image analysis and other techniques," *Functional Plant Biology*, vol. 18, no. 6, pp. 637-648, 1991.

[25] I. S. Ahmad, J. F. Reid, M. R. Paulsen, and J. B. Sinclair, "Color classifier for symptomatic soybean seeds using image processing," *Plant disease*, vol. 83, no. 4, pp. 320-327, 1999.

[26] N. Aleixos, J. Blasco, F. Navarron, E. J. C. Molto, and e. i. agriculture, "Multispectral inspection of citrus in real-time using machine vision and digital signal processors," *Computers and electronics in agriculture*, vol. 33, no. 2, pp. 121-137, 2002.

[27] F. López-García, G. Andreu-García, J. Blasco, N. Aleixos, J.-M. J. C. Valiente, and E. i. Agriculture, "Automatic detection of skin defects in citrus fruits using a multivariate image analysis approach," *Computers and Electronics in Agriculture*, vol. 71, no. 2, pp. 189-197, 2010.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[29] H. Q. Cap, K. Suwa, E. Fujita, S. Kagiwada, H. Uga, and H. Iyatomi, "A deep learning approach for on-site plant leaf detection," in *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, 2018, pp. 118-122: IEEE.

[30] A. Khan, U. Nawaz, A. Ulhaq, and R. W. Robinson, "Real-time plant health assessment via implementing cloud-based scalable transfer learning on AWS DeepLens," *Plos one*, vol. 15, no. 12, p. e0243243, 2020.

[31] G. Wang, Y. Sun, J. J. C. i. Wang, and neuroscience, "Automatic image-based plant disease severity estimation using deep learning," *Computational intelligence and neuroscience*, vol. 2017, 2017.

[32] J. C. Valdoria, A. R. Caballeo, B. I. D. Fernandez, and J. M. M. Condino, "iDahon: An Android Based Terrestrial Plant Disease Detection Mobile Application Through Digital Image Processing Using Deep Learning Neural Network Algorithm," in *2019 4th International Conference on Information Technology (InCIT)*, 2019, pp. 94-98: IEEE.

[33] W. Pan, J. Qin, X. Xiang, Y. Wu, Y. Tan, and L. J. I. A. Xiang, "A smart mobile diagnosis system for citrus diseases based on densely connected convolutional networks," *IEEE Access*, vol. 7, pp. 87534-87542, 2019.

[34] A. Elhassouny and F. Smarandache, "Smart mobile application to recognize tomato leaf diseases using Convolutional Neural Networks," in *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, 2019, pp. 1-4: IEEE.

[35] L. M. Mrisho, N. A. Mbilinyi, M. Ndalahwa, A. M. Ramcharan, A. K. Kehs, P. C. McCloskey, H. Murithi, D. P. Hughes and J. P. Legg, "Accuracy of a Smartphone-Based Object Detection Model, PlantVillage Nuru, in Identifying the Foliar Symptoms of the Viral Diseases of Cassava–CMD and CBSD," *Frontiers in plant science*, vol. 11, p. 1964, 2020.

[36] D. Hughes and M. Salathé, "An open access repository of images on plant health to enable the development of mobile disease diagnostics," *arXiv preprint arXiv:1511.08060*, 2015.

[37] T. L. Fine, *Feedforward neural network methodology*. Springer Science & Business Media, 2006.

[38] J. Amara, B. Bouaziz, and A. J. D. f. B. Algergawy, Technologie und Web -Workshopband, "A deep learning-based approach for banana leaf diseases classification," *Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband* 2017.

[39] K. O'Shea and R. J. a. p. a. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458* , 2015.

[40] K. Radhika, K. Devika, T. Aswathi, P. Sreevidya, V. Sowmya, and K. Soman, "Performance Analysis of NASNet on Unconstrained Ear Recognition," in *Nature Inspired Computing for Data Science*: Springer, 2020, pp. 57-82.

[41] B. Zoph and Q. V. J. a. p. a. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[42] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697-8710.

[43] S. Garfinkel, "An evaluation of Amazon's grid computing services: EC2, S3, and SQS," 2007.

[44] V. Novick, *React Native-Building Mobile Apps with JavaScript*. Packt Publishing Ltd, 2017.

[45] E. Zawadzka-Gosk, K. Wołk, and W. Czarnowski, "Deep learning in state-of-the-art image classification exceeding 99% accuracy," in *World Conference on Information Systems and Technologies*, 2019, pp. 946-957: Springer.

[46] J. Howard and S. J. I. Gugger, "Fastai: A Layered API for Deep Learning," *Information*, vol. 11, no. 2, p. 108, 2020.

[47] M. Arsenovic, M. Karanovic, S. Sladojevic, A. Anderla, and D. Stefanovic, "Solving current limitations of deep learning based approaches for plant disease detection," *Symmetry*, vol. 11, no. 7, p. 939, 2019.

[48] A. K. Rangarajan, R. Purushothaman, and A. Ramesh, "Tomato crop disease classification using pre-trained deep learning algorithm," *Procedia computer science*, vol. 133, pp. 1040-1047, 2018.

[49] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, "Deep Learning for Image-Based Cassava Disease Detection," (in English), *Frontiers in plant science*, Original Research vol. 8, no. 1852, 2017-October-27 2017.

BIOGRAPHIES

Adedamola O Adedoja received his BSc. (Hons), Computer Science from the Bowen University, Iwo, Nigeria in 2010 and BTech in Information Technology in 2015 from Tshwane University of Technology, Pretoria, South Africa. He is currently studying towards a master's degree with the Tshwane University of Technology, Pretoria, South Africa. His research interests are focused on big data analytics, image processing, computer vision, pattern recognition, machine learning and deep learning.

Pius A. Owolawi (PhD) received his undergraduate degree in 2001 from the Federal University of Technology, Akure, Nigeria and also bagged his MSc and PhD in Electrical Engineering from University of KwaZulu-Natal, South Africa in 2006 and 2010 respectively. He is currently the Head of Department of Computer Systems Engineering, Tshwane University of Technology, South Africa. His research interests include, RF, Green communication, radio-wave propagation (Microwave/ Millimeter wave systems), Satellite and free space optical communications, IOT, Embedded systems, Machine learning and data analytics. Prof. Owolawi was a recipient of Joint holder of best paper award for a paper presented at the 2nd international conference on applied and theoretical information systems research, in Taipei, Taiwan, 2012 and a recipient of the Vice Chancellor's teaching Excellence Award, 2015.

Temitope Mapayi received his BSc. (Hons), Computer Science from the University of Ado Ekiti in 2005, MSc in Computer Science in 2009 from the Nigeria's Premier University, University of Ibadan, and obtained PhD degree in Computer Science from the University of KwaZulu-Natal, Durban, South Africa in 2015. He works with the Department of Computer Systems Engineering, Tshwane University of Technology, Pretoria, South Africa. His research interests are focused on big data analytics, image processing, computer vision, pattern recognition, machine learning and deep learning. Dr Mapayi is a professional member of IAENG. He has authored and co-authored several scientific international journal and conference articles. He has also served as technical committee member of international conferences and reviewer to reputable international journals.

Chunling Tu received the Bachelor degree of computer science from Tianjin University of Technology and Education, China in 2002; MTech and MSc degrees in Electrical Engineering from Tshwane University of Technology (South Africa) and ESIEE Paris University (France) in 2010; DTech and PhD degrees of Electrical Engineering from the Tshwane University of Technology and University Paris East, France in 2015. She currently lectures at the Tshwane University of Technology. Her research interests include image processing, AI, industrial control, machine learning, deep learning and pattern recognition.