CSE 498R

Title: Detection and Classification of Plant Disease Using Neural Network Approach

Name	ID
Mohammad Burhan Uddin	1812673042
Md. Rakibul Islam Rakib	1911977642

Supervisor

Mr. Rifat Ahmed
Hassan

Lecturer

Department of Electrical and Computer Engineering

North South University

Summer 23

Department of Electrical and Computer Engineering

North South University

Declaration

This is to declare that no part of this report or the project has been previously submitted elsewherefor the fulfilment of any other degree or program. Proper acknowledgment has been provided for any material that has been taken from previously published sources in the bibliography section of this report.

Mohammad Burhan Uddin

ECE Department

North South University, Bangladesh

Md. Rakibul Islam Rakib

ECE Department

North South University, Bangladesh

Approval

The Thesis entitled “Detection of Plant Diseases based on Deep Convolutional Neural Network” by Mohammad Burhan Uddin (ID#1812673042) Md Rakibul Islam Rakib (ID#1911977642) has been accepted as satisfactory and approved for partial fulfilment of the requirement of BSc in CSE degree program in Summer 23

Supervisor’s Signature

MR.Rifat ahmed
hassan

Lecturer
Department of Electrical and Computer Engineering

North South University

Dhaka, Bangladesh.

Department Chair’s Signature

Dr.Rajesh Palit

Professor & Chairman Department of Electrical and
Computer Engineering North South University

Dhaka, Bangladesh.

Acknowledgment

First of all, we would like to express our profound gratitude to our honorable course instructor, **MR. Rifat Ahmed Hassan** for his constant and meticulous supervision, valuable suggestions, his patience, and encouragement to complete the thesis work. We would also like to thank the ECE department of North South University for providing us with the opportunity to have an industrial level design experience as part of our curriculum for the undergraduate program. Finally, we would like to thank our families and everybody who supported us and provided guidance for the completion of this research.

Abstract

Bangladesh's agriculture sector is an important one that greatly affects the nation's wealth. Shrubberies have advanced on a vital get-up-and-go basis and a potentially fatal component, solving the conundrum of determining the whole heating issue. However, plants are currently impacted by several factors. This illness detection challenge is well-solved by deep learning techniques. Convolutional neural networks are very good at complex picture categorization and object identification tasks. The main goal of the projected effort is to find a solution for the challenge of diagnosing plant illnesses using the least amount of computational resources and the most easy way possible, producing better results than standard representations. The PlantVillage Dataset from the open-source Kaggle platform is used in this work. We categorize Potato Leaf Disease based on the many types of leaf disease photos included in the dataset. Inception V3 can execute 98% of the classification correctly, CNN can obtain an average accuracy of 99.61%, and VGG16 can provide 100% accuracy when it comes to leaf disease classification. To enable a behavior surveillance procedure for the identified illness, a graphical user interface (GUI) for the system is also being finalized. From photos captured in the outdoors, CNN is able to identify plant illnesses and extract important traits. The deep CNN model is promising and can have a considerable influence on the efficiency of disease identification. It also has potential for disease detection in real-time agricultural systems, as indicated by the accuracy results in disease identification.

Index Terms—Plant disease detection, Plant disease Classification, Potato Plant Disease, Image Processing, CNN, VGG 16, Inception V3, Color thresholding, Deep learning.

Table of Contents

Chapter 1: Introduction

The amount and productivity of agricultural land determine the economic development of a nation. For most people, agriculture provides their main source of income. Depending on the fertility of the soil and the availability of resources, farmers plant a range of crops. Variations in meteorological parameters including rainfall, temperature, and soil fertility can lead to crop infections by fungus, bacteria, and viruses [1]. They organize the plants to avoid sicknesses and enhance the production and quality of their products by using the proper pesticides and weedkillers. Plant disease identification and research are conducted by visual study of the plants' patterns. Early diagnosis of plant diseases is beneficial since they may be managed. In many countries, farmers lack the resources or knowledge to communicate with experts. One such technique for detection is the visual inspection of leaf patterns by professionals. But it requires a large number of specialized personnel. In this case, an automated plant infection or disease monitoring system will be useful. By comparing the stored plant disease symptoms with the leaves of the plants in the agricultural farm area, automation will be less expensive. The three plant diseases included in this category are bacterial blight, anthracnose, and Cercospora leaf spot. On the leaf, anthracnose causes brown or tan spots with irregular shapes to appear. These areas will be adjacent to the leaf veins. If the infection is severe, leaf drop will happen. Cercospora spray spot plants have tiny brown dots with a magenta border. It eliminates elasticities and consumes a grey focus. A hole is left when the leaf material turns friable and reedy and falls out. Proteomics Blight disease can infect a plant's trunk, branches, shoots, buds, flowers, leaves, and fruit. A tiny, pale green dot appears on the leaf and grows throughout it. The lesion area eventually turns into a dead, dry patch [2]. To diagnose the illness or impurity using image processing, a leaf trial is required. Plant disease classification involves a number of interconnected processes, including image capture, pre-processing, dissection, feature mining, and organization. In this study, a plant leaf detection system is developed and constructed to process leaf image detection and categorize the leaf for disease diagnosis using image processing. A few procedures are discussed and looked at in order to carry out the plan in picture segmentation and classification in order to determine which approach is most suited and feasible for improving accuracy in identifying the sickness on the plant leaf. Utilizing the CNN, VGG 16, and Inception V3 classifiers, image segmentation is studied.

Chapter 2: Literature Review

Using CNN and deep learning structures, the section assigns thematic trends to agricultural submissions. Previous methods for image processing, machine learning, and deep learning were abandoned in order to address various plant pests. Most of these systems typically work like this: A digital camera is used to capture the first cardinal photographs. After that, the photographs are prepared for the following stages using image distribution techniques including image heightening, segmentation, shade intergalactic restoration, and filtering. The basic components of the picture are then unaffected and castoff as classifier input [3]. Thus, the total classification accuracy is determined by the feature mining techniques used in image processing. Nevertheless, subsequent research has shown that the advanced recital may be achieved by systems trained on generic data.

In recent years, image processing has been the focus of research on the identification and detection of plant leaf disease. These studies [4] [5] demonstrate the authors' recommendation of the K-means method for image subdivision. The approach to divide the Dataset into k distinct, non-overlapping clusters was dropped in order to distinguish between the critical component and the nurturing, as well as the contaminated region on the foliage, with each statistic point corresponding to a single assembly. For subdivisions, the author advises use color-based thresholding. In order to conceal forbidden regions, verges were installed on the various networks, and pictures were updated to other color spaces, such as LAB and HSV.

[6] Here, machine learning techniques like Decision Trees, Support Vector Machines (SVMs), and artificial neural networks have been abandoned. Antiquated techniques for classifying images have relied on manually designed regions like SIFT [7], HOG [8], and SURF [9], monitored by employing information on the processes of learning in these characteristic areas. Consequently, the recitation of all these methods heavily relies on the fundamental predefined frameworks. But according to a recent machine learning study, learnt representations are more organized and helpful. The primary benefit of representation learning is in its ability to automatically sift through large amounts of image data and identify features that might be used to precisely classify images. The author [10] used CNN methods such as AlexNet and Inception Net to identify 26 distinct plant bugs.

[11] The author achieved excellent levels of classification precision by identifying a variety of plant illnesses using different CNN approaches. They also used real photos to construct the CNN architecture as part of their methodology. In [12] Unlike plant bugs, the author used a deep learning structure to identify 13. The deep learning framework was authorized for use in CNN training. The author extensively examined the shortcomings of several deep learning techniques in the field of agriculture. The scientists intended to use a nine-layer CNN model to identify plant pests. They employed data-augmentation techniques to increase the quantity of the data and the PlantVillage dataset for their research. The authors [13] projected a nine-layer Convolution Neural Network model to categorize plant diseases. After removing the PlantVillage dataset and using data-augmentation techniques to boost the amount of data for testing scenarios, they assessed the presentation. The authors' accuracy was greater than that of a traditional machine-learning approach. In [14] With enhanced hyperparameters such as a max epoch, minibatch size, and bias learning rate, pretrained AlexNet and GoogleNet were still used to distinguish three different soybean bugs from pictures of healthy leaves. Six of the pre-trained networks that were used were VGG16, AlexNet, GoogleNet, ResNet, and DenseNet.

[15] Here, the author identifies three different kinds of bugs and two different forms of pest damage in cassava shrubberies using a transfer-learning approach. The scientists subsequently carried out more research on the identification of cassava plant diseases, obtaining an accuracy of 80.6% by employing a CNN model based on a smartphone.

Chapter 3: Methodology

To categorize potato plant disease, a substantial collection of plant leaf images is needed. The pictures are from a set known as PlantVillage. The method chosen is extensively thought out in this subject.

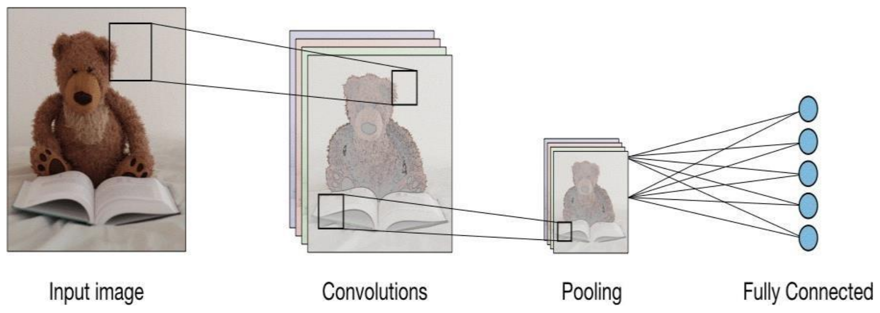


Fig. 1. Convolutional layer, pooling layer and fully connected layer.

In Fig. 1, We present the operation of a convolutional network. This illustrates the connectivity between the convolutional, pooling, and fully connected layers.

a) Dataset

For the purposes of training and testing classification, a suitable and large dataset is necessary. The expert-it's dataset, known as the PlantVillage Dataset [15], was sourced from the open-access Kaggle platform. It includes several images and tags of plant leaves. It is a dataset of 54,305 photographs of healthy and diseased leaves that were shot in various settings. Images of both healthy and diseased leaves from 14 other plant types are contained in 38 distinct classes within the dataset. In order to detect potato diseases, we have chosen 2,152 photos of three different types of potato leaves: healthy leaves, late blight, and early blight. The samples in the Dataset are categorized in Table I. A few instances from the database are shown in Fig. 2.

b) Dataset Preprocessing

In order to get the Dataset ready for training, different resolution pictures are shrunk to 128 by 128 pixels. The neural network may be biased by the training photos' varying lighting and backgrounds because they were taken in an abandoned setting. To test this, the segmented database and greyscale versions of the tests were also used. Next, we divided the dataset into three categories: training (80%), validation (10%), and testing (10%).

TABLE I

DATASET USE FOR THE CLASSIFICATION

No.	**Type**	**Number**
1	Healthy Leaf	152
2	Early Blight Leaf	1,000
3	Late Blight Leaf	1,000

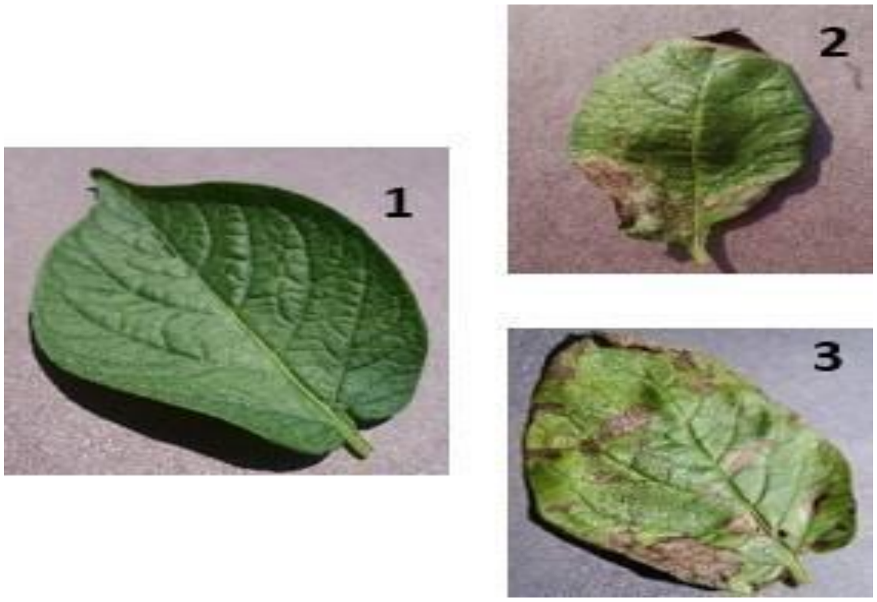


Fig. 2. Sample images from the Dataset, (1) Healthy Potato Leaf, (2) Early blight Potato Leaf, and (3) Late blight Potato leaf.

c) Segmentation

The primary goal of image segmentation is to extract the region of interest (ROI) of the tomato leaf from the image. In this case, the algorithm will be evaluating the plant leaf in the foreground, while the grey backdrop of the Dataset contains no information. Consequently, the backdrop of the image must be removed using a mask so that only the leaf-containing pixels and a black background remain. After masking the surrounding area of the leaf image, the healthy portion will be further veiled in order to calculate the proportion of the leaf affected by the disease

d) Design

The proposed system architecture, as seen in Fig. 3, comprises gathering information from a large dataset, processing several convolution layers, and classifying plant diseases to ascertain whether the plant image is healthy or diseased.

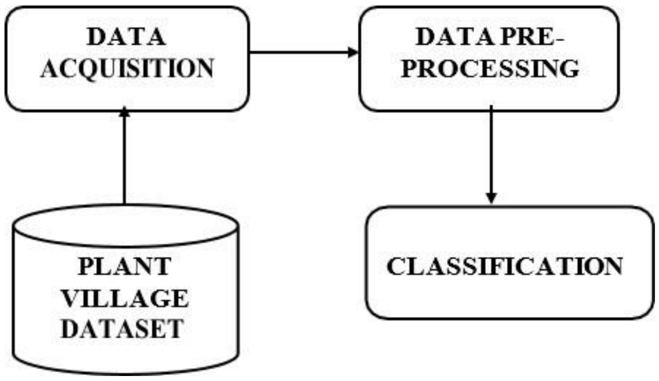


Fig. 3. Working sequences of the proposed plant disease detection system.

e) Proposed CNN Model

Convolutional neural networks are a kind of deep neural networks. A CNN is better suited for processing 2D data, such photos, by combining input data with well-read features prior to employing 2D convolutional layers. CNNs are capable of classifying pictures without the need for human feature extraction or removal. Features are directly extracted from images by the CNN model. In contrast to the extracted features, which are well-read and not pre-trained, the network is trained on several image groups. The Convolutional Neural Network (CNN) model processes images across several layers. The Convo Layer, Fully, Soft-max Layer, Connected Layer, Input Layer, Output Layer, and Pooling Layer are a few of them. [16] Depending on the issue at hand, several CNN architectures are used. The suggested model uses three convolutional layers, with a max-pooling layer in between. MPL, the last layer, has full connectivity. The ReLu activation function is applied to the output of each fully connected and convolutional layer.

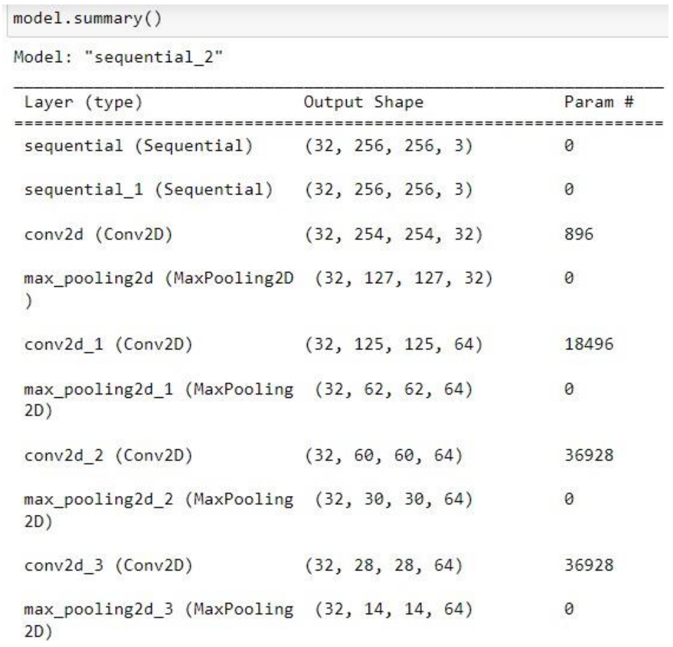
TABLE II

ARCHITECTURE OF THE PROPOSED CNN MODEL

Layer	Type	Filter Size	Stride	Output Size
L1	Conv	3 × 3	1	128 × 128 × 32
	Pool	2 × 2	2	64 × 64 × 32
L2	Conv	4 × 4	1	61 × 61 × 32
	Pool	2 × 2	2	64 × 64 × 32

L3	Conv	1 × 1	1	30 × 30 × 28
	Pool	2 × 2	2	15 × 15 × 128

For three channels and fifty epochs with a 128 by 128 image size, we use a batch size of 32 in this illustration. The input image is filtered using 32 3 x 3 kernels in the first convolutional layer. Later, the second convolution layer—which has 6464 kernels—is fed the output obtained from max pooling. Finally, there is a final convolutional layer with 256 size 11 kernels and 512 neurons that are completely coupled. After this layer's output is received, the softmax function generates a probability distribution for each of the four output classes. [17]. The proposed model (CNN) is presented in Table II. Training the model is done by adaptive moment training.



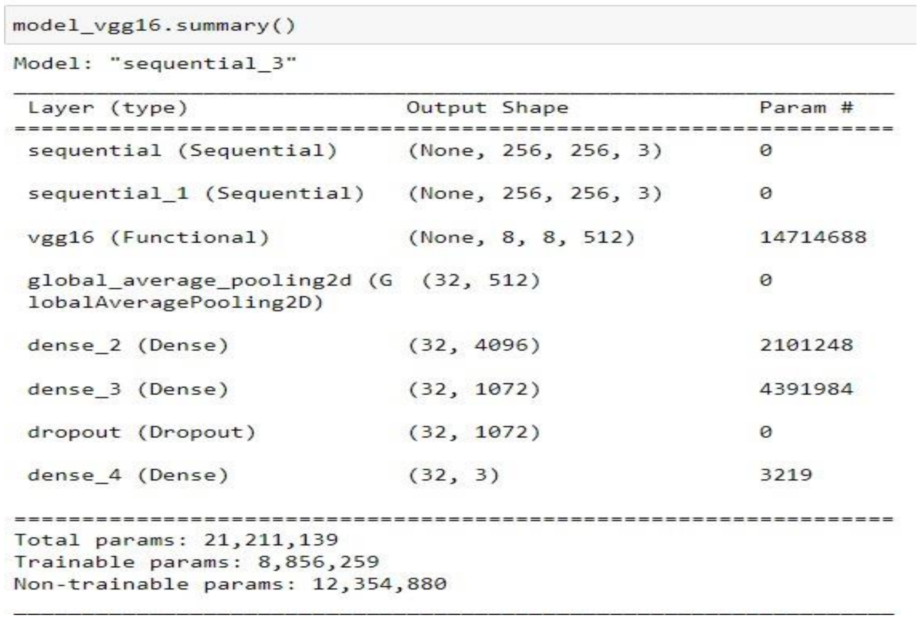
```
model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0
 )

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0
 2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0
 2D)
```

Fig. 4. Various layers, output shape and number of parameters of the CNN Model.

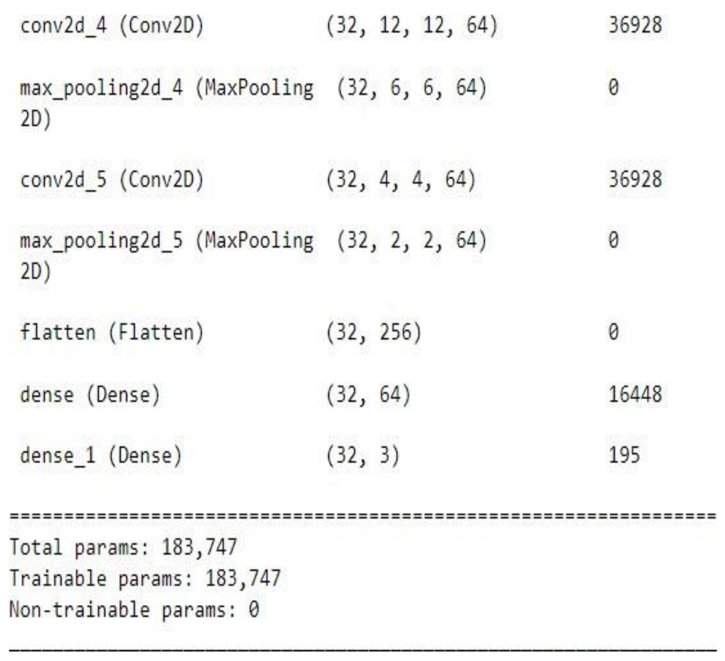
As shown in Fig. 4, estimation (Adam) was performed using a batch size of 32 for 50 epochs.

f) VGG 16 Model

VGG 16 is a CNN model designed for large-scale picture analysis. Completing two tasks is necessary for the best identification of plant diseases. Object localization, which finds objects from various classes inside an image, is the initial phase. Image categorization, the second stage, entails grouping photos into various categories. The CNN model consists of seven tiers. Every layer processes information in a unique way. The following are the seven tiers: Convolutional Layer, Input Layer, Output Layer, Pooling Layer, Fully Convolutional Layer, Fully Connected Layer, and Soft-max Layer [18]. An overview of the VGG16 model in practice is presented in Fig. 5.



```
model_vgg16.summary()

Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0

 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 global_average_pooling2d (G  (32, 512)                0
 lobalAveragePooling2D)

 dense_2 (Dense)             (32, 4096)                2101248

 dense_3 (Dense)             (32, 1072)                4391984

 dropout (Dropout)           (32, 1072)                0

 dense_4 (Dense)             (32, 3)                   3219

=================================================================
Total params: 21,211,139
Trainable params: 8,856,259
Non-trainable params: 12,354,880
_________________________________________________________________
```

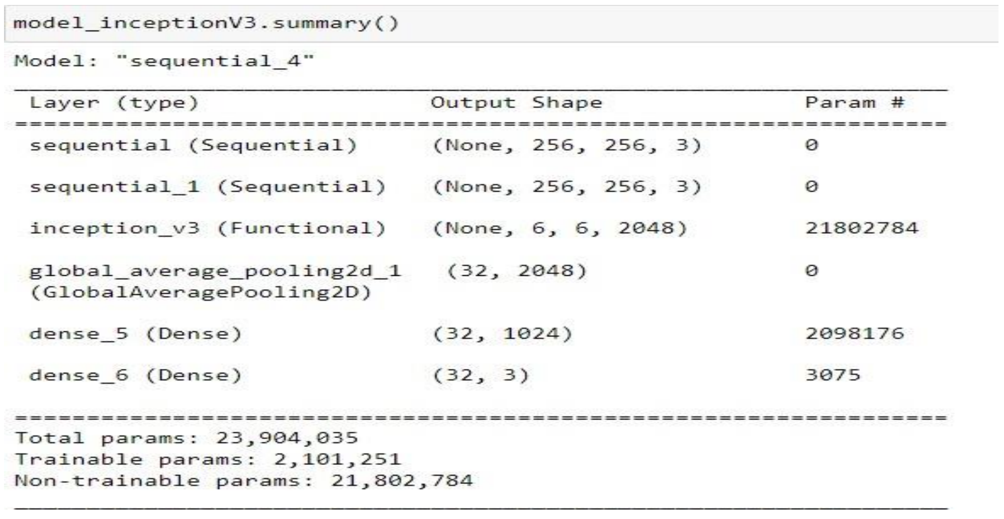


```
 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0
 2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0
 2D)

 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448

 dense_1 (Dense)             (32, 3)                   195

=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5. Various layers, output shape and number of parameters of the VGG16 model.

g) Inception V3

Compared to its predecessors, the 2015-released Inception v3 model has 42 layers and a lower error rate. When a model had several deep layers of convolutions, the data were over-fit. To avoid this, the concept V3 model makes use of the notion of using many filters on the same level, each with a different size. Consequently, our model is broader than deeper since the inception models have parallel layers instead of deep layers. The Inception model consists of many Inception modules. [19]. Figure 6 presents the model summary.



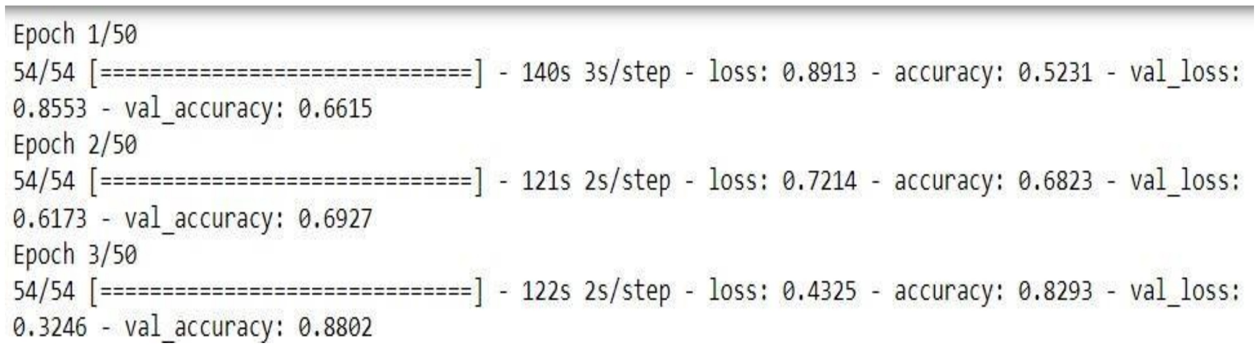
```
model_inceptionV3.summary()
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0

 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 global_average_pooling2d_1   (32, 2048)               0
 (GlobalAveragePooling2D)

 dense_5 (Dense)             (32, 1024)                2098176

 dense_6 (Dense)             (32, 3)                   3075

=================================================================
Total params: 23,904,035
Trainable params: 2,101,251
Non-trainable params: 21,802,784
_________________________________________________________________
```

Fig. 6. Various layers, output shape and number of parameters of the Inception V3 model.

Chapter 4: Results & Discussion

Ten percent is utilized for testing, ten percent for validation, and eighty percent of the dataset is used for training. Numerous models with different topologies and learning rates are tested. The learning parameter, kernel size, and filter size were among the network parameters that were chosen via trial and error. Research has shown that training is more effective when the ReLu activation function is used.

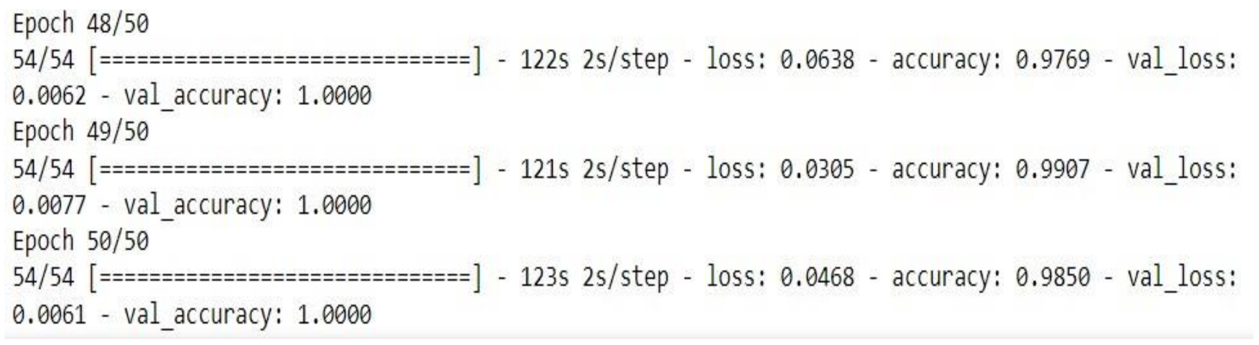
a) CNN Model



```
Epoch 1/50
54/54 [==============================] - 140s 3s/step - loss: 0.8913 - accuracy: 0.5231 - val_loss:
0.8553 - val_accuracy: 0.6615
Epoch 2/50
54/54 [==============================] - 121s 2s/step - loss: 0.7214 - accuracy: 0.6823 - val_loss:
0.6173 - val_accuracy: 0.6927
Epoch 3/50
54/54 [==============================] - 122s 2s/step - loss: 0.4325 - accuracy: 0.8293 - val_loss:
0.3246 - val_accuracy: 0.8802
```

•

•



```
Epoch 48/50
54/54 [==============================] - 122s 2s/step - loss: 0.0638 - accuracy: 0.9769 - val_loss:
0.0062 - val_accuracy: 1.0000
Epoch 49/50
54/54 [==============================] - 121s 2s/step - loss: 0.0305 - accuracy: 0.9907 - val_loss:
0.0077 - val_accuracy: 1.0000
Epoch 50/50
54/54 [==============================] - 123s 2s/step - loss: 0.0468 - accuracy: 0.9850 - val_loss:
0.0061 - val_accuracy: 1.0000
```

Fig. 7. Accuracy results of the CNN model.

The classification accuracy is 66.15% in the first epoch and rises to 99.61% after 50 epochs, as Fig. 7 illustrates. The model's three convolution layers are followed by a max-pooling layer, which improves classification accuracy. The ReLu activation function is applied to each layer. The model's training accuracy vs validation graphs are shown in Fig. 8.

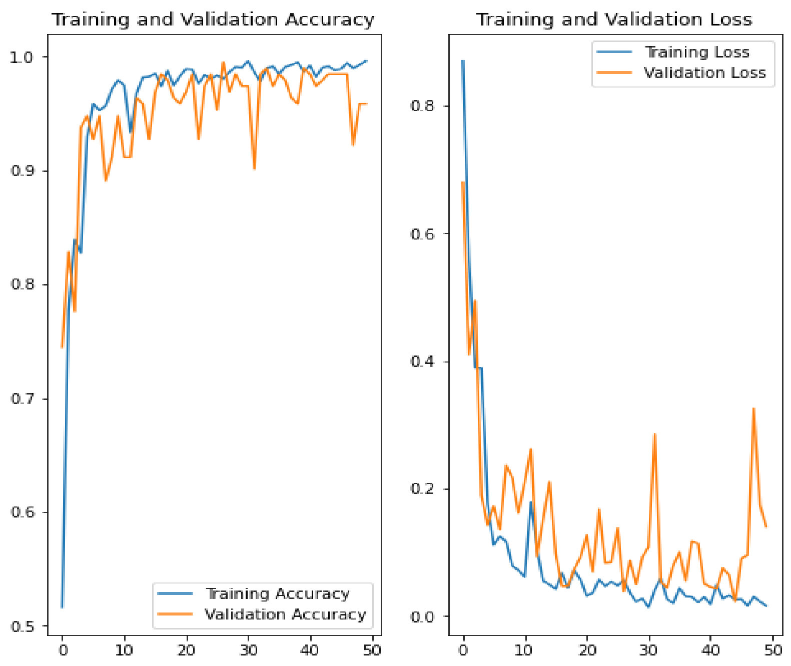


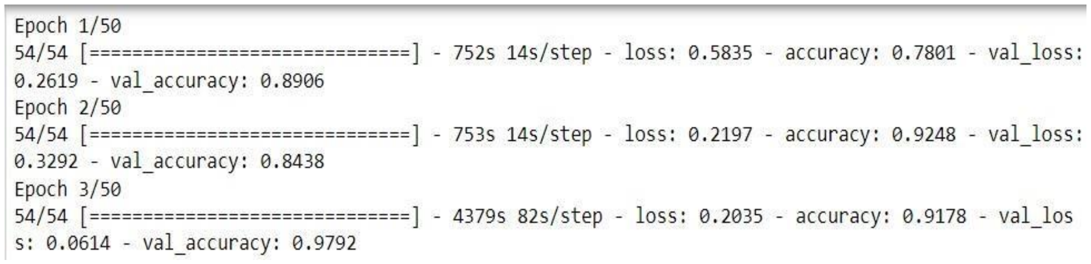
Fig. 8. Training vs validation accuracy and loss of the CNN model.

The over-fitting of the model is demonstrated by the graphs. Over-fitting happens when the model fits the training set too closely. Because of this, the model finds it difficult to generalize to new examples that do not belong in the training set. A number of strategies have been devised to avoid over-fitting. Weight penalties like regularization, dropout, and data augmentation are used in these strategies. Trials were carried out to ascertain the effect of every strategy on the model's functionality. The first experiment we conducted included improving the training data by rotating, flipping, and rescaling the images because the dataset was too small in comparison to the total number of trainable parameters in the model. Only the training data are used for data augmentation. By itself, data augmentation significantly lessens over-fitting.

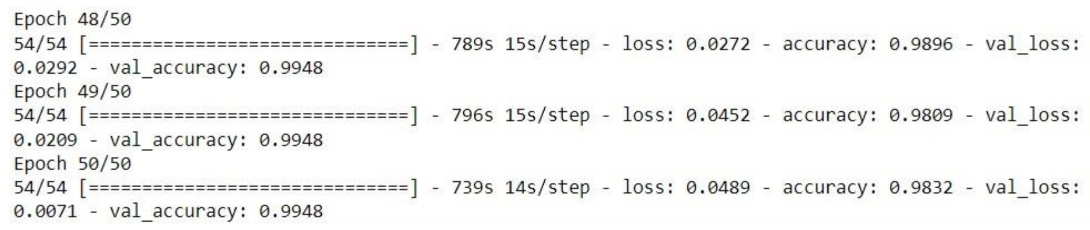
It also raises the accuracy of validation. Dropping out and regularization are also employed. In terms of performance, both models marginally outperformed the model. Consequently, we add a dropout layer with a probability of 0.5 when the MLP yields high classification accuracy.

The accuracy and loss outcomes for the VGG16 model are displayed in Fig. 9. It achieved 78% accuracy in the first epoch and 100% accuracy in the 50th epoch, indicating strong performance. The training vs validation accuracy and loss curve are shown in Fig. 10.

b) VGG 16 Model



```
Epoch 1/50
54/54 [==============================] - 752s 14s/step - loss: 0.5835 - accuracy: 0.7801 - val_loss:
0.2619 - val_accuracy: 0.8906
Epoch 2/50
54/54 [==============================] - 753s 14s/step - loss: 0.2197 - accuracy: 0.9248 - val_loss:
0.3292 - val_accuracy: 0.8438
Epoch 3/50
54/54 [==============================] - 4379s 82s/step - loss: 0.2035 - accuracy: 0.9178 - val_los
s: 0.0614 - val_accuracy: 0.9792
```



```
Epoch 48/50
54/54 [==============================] - 789s 15s/step - loss: 0.0272 - accuracy: 0.9896 - val_loss:
0.0292 - val_accuracy: 0.9948
Epoch 49/50
54/54 [==============================] - 796s 15s/step - loss: 0.0452 - accuracy: 0.9809 - val_loss:
0.0209 - val_accuracy: 0.9948
Epoch 50/50
54/54 [==============================] - 739s 14s/step - loss: 0.0489 - accuracy: 0.9832 - val_loss:
0.0071 - val_accuracy: 0.9948
```

Fig. 9. Accuracy results of the VGG 16 model.

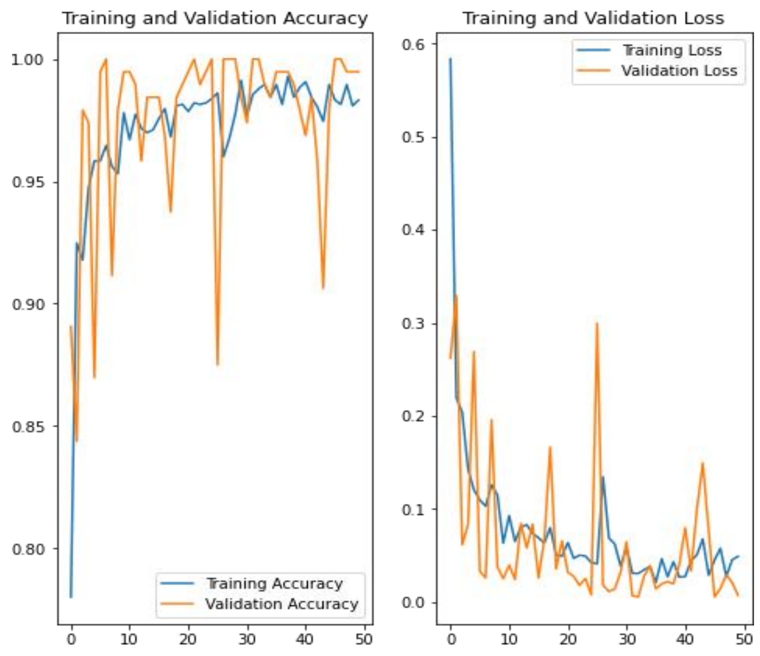
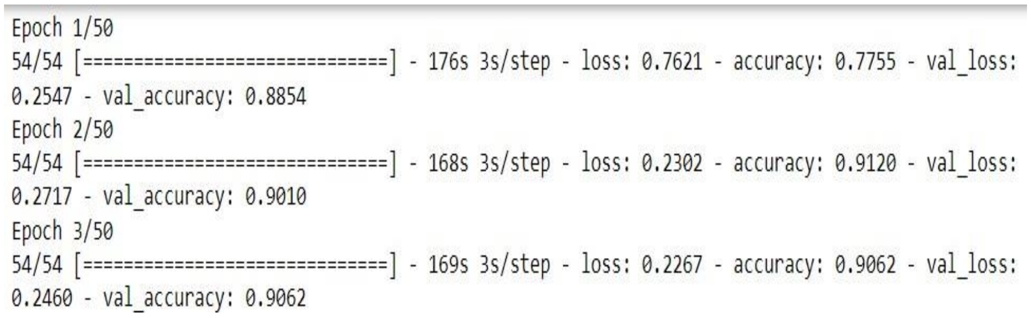


Fig. 10. Training vs validation accuracy and loss of the VGG 16 model.

The accuracy and loss outcomes for the VGG16 model are displayed in Fig. 9. It achieved 78% accuracy in the first epoch and 100% accuracy in the 50th epoch, indicating strong performance. The training vs validation accuracy and loss curve are shown in Fig. 10.

c) Inception V3 Model

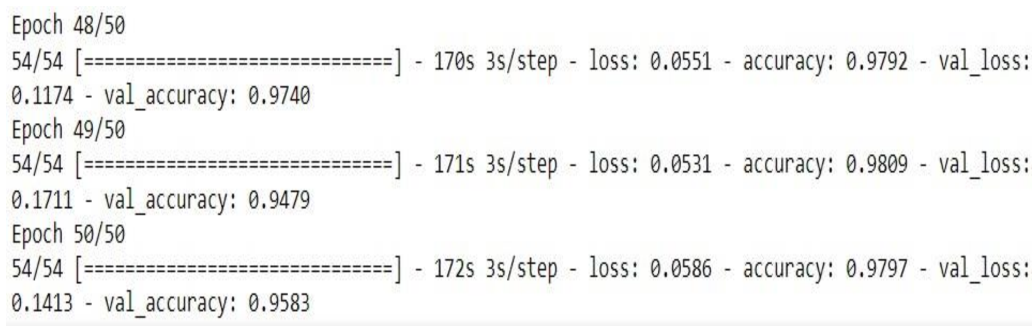
The accuracy and loss outcomes for the Inception V3 model are displayed in Fig. 11. It achieves 77% accuracy in the first epoch and achieves good performance with 97% accuracy in the 50th epoch. The training vs validation accuracy and loss curve are shown in Fig. 12.



```
Epoch 1/50
54/54 [==============================] - 176s 3s/step - loss: 0.7621 - accuracy: 0.7755 - val_loss:
0.2547 - val_accuracy: 0.8854
Epoch 2/50
54/54 [==============================] - 168s 3s/step - loss: 0.2302 - accuracy: 0.9120 - val_loss:
0.2717 - val_accuracy: 0.9010
Epoch 3/50
54/54 [==============================] - 169s 3s/step - loss: 0.2267 - accuracy: 0.9062 - val_loss:
0.2460 - val_accuracy: 0.9062
```

•

•



```
Epoch 48/50
54/54 [==============================] - 170s 3s/step - loss: 0.0551 - accuracy: 0.9792 - val_loss:
0.1174 - val_accuracy: 0.9740
Epoch 49/50
54/54 [==============================] - 171s 3s/step - loss: 0.0531 - accuracy: 0.9809 - val_loss:
0.1711 - val_accuracy: 0.9479
Epoch 50/50
54/54 [==============================] - 172s 3s/step - loss: 0.0586 - accuracy: 0.9797 - val_loss:
0.1413 - val_accuracy: 0.9583
```

Fig. 11. Accuracy results of the Inception V3 model.

.

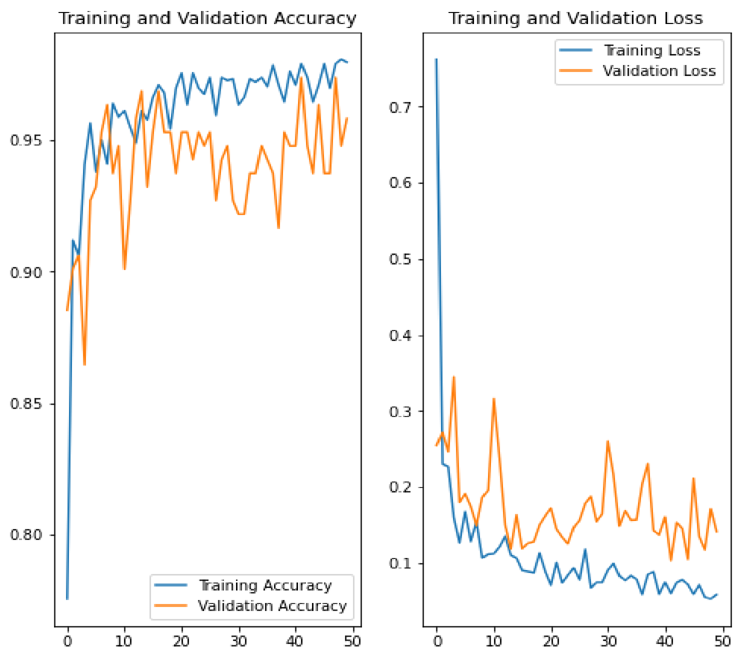


Fig. 12. Training vs validation accuracy and loss of the Inception V3 model.

TABLE III Accuracy of each model

	Training Accuracy	Test Accuracy
CNN	97.50%	99.61%
VGG16	98.32%	100%
INCEPTION V3	97.97%	98.83%

Table III shows All models with their average accuracy and chooses the best fit model for this Dataset. Fig. 13 demonstrate the confidence of the best-fit model VGG16.

TABLE IV

COMPARISON TABLE

Reference	**Applied Model**	**Dataset**	**Accuracy**
[10]	AlexNet and GoogleNet	PlantVillage Dataset	99.27% in AlexNet 99.34% in GoogleNet
[12]	Finetuned CNN Architecture	Stanford Background Dataset	96.3%
[13]	Inception V3 Based on GoogleNet	Cassava Leaf Disease Image Dataset	93%
[15]	Nine-layer Deep CNN	Plant Leaf Disease Dataset	96.46%
This Work	**VGG 16**	**PlantVillage Dataset**	**100%**

Table IV compares different models applied in different Dataset chosen from several research papers. We compare it with their accuracy. And Finlay shows our proposed model in the PlantVillage Data-set with 100% accuracy.

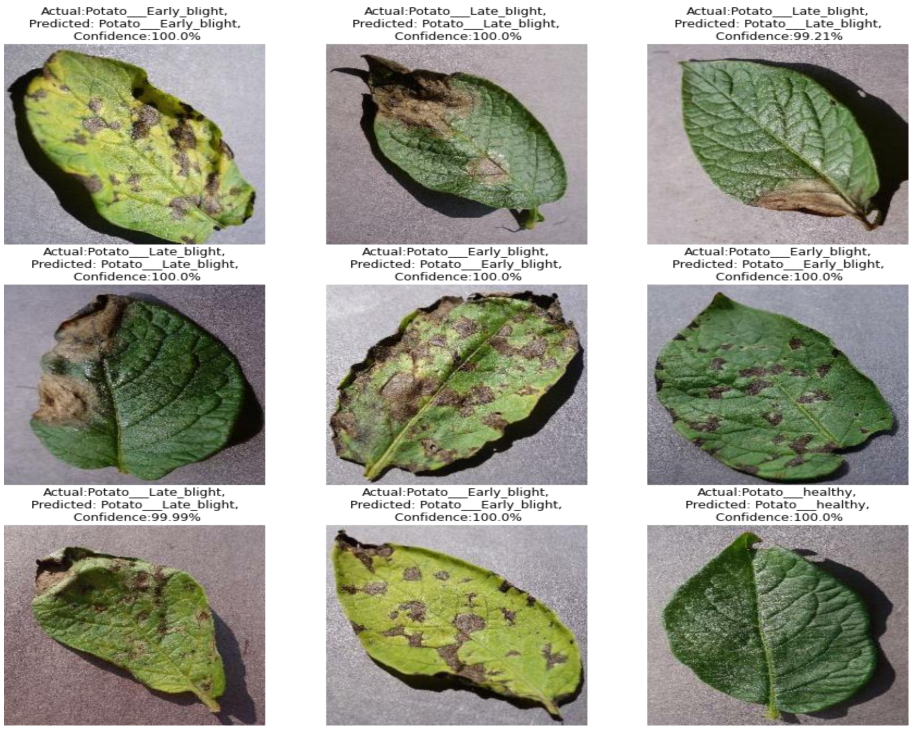

Actual:Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence:100.0%
Actual:Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence:100.0%
Actual:Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence:99.21%
Actual:Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence:100.0%
Actual:Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence:100.0%
Actual:Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence:100.0%
Actual:Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence:99.99%
Actual:Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence:100.0%
Actual:Potato___healthy,
Predicted: Potato___healthy,
Confidence:100.0%

Fig. 13. Predicted class and the confidence of using the VGG16 model.

Chapter 5: Conclusion

This training uses CNN, VGG16, and Inception V3 to identify and categorize Potato Plant bugs. Using images gathered from the natural world, the system was trained to attain a 100% classification accuracy. This demonstrates how VGG 16 can gather vital data from the natural world that is required to identify plant diseases.

According to our friends, this was the first time that pictures shot in a natural setting have been used to get meaningful results. Experiments also show that when the dataset is limited, adding additional data to the training set improves network learning. Regularization and dropout were also shown to have an influence on overcoming over-fitting. A First API server based on the concept may likewise be developed. After that, a Google Cloud website and deployment model may be created. At last, we are able to create an application that will support farming.

REFERENCES

[1] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, "Deep learning for image-based cassava disease detection," *Frontiers in Plant Science*, vol. 8, pp. 1–7, 2017.

[2] P. K. Sethy, N. K. Barpanda, A. K. Rath, and S. K. Behera, "Deep feature- based rice leaf disease identification using support vector machine," *Computers and Electronics in Agriculture*, vol. 175, p. 105527, 2020.

[3] H. M. Alexander, K. E. Mauck, A. E. Whitfield, K. A. Garrett, and C. M. Malmstrom, "Plant-virus interactions and the agro-ecological interface," *European Journal of plant pathology*, vol. 138, pp. 529–547, 2014.

[4] R. M. Prakash, G. Saraswathy, G. Ramalakshmi, K. Mangaleswari, and T. Kaviya, "Detection of leaf diseases and classification using digital image processing," in *International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1– 4, 2017.

[5] A. Devaraj, K. Rathan, S. Jaahnavi, and K. Indira, "Identification of plant disease using image processing technique," in *International Conference on Communication and Signal Processing (ICCSP)*, pp. 0749–0753, 2019.

[6] S. P. Mohanty, D. P. Hughes, and M. Salathe', "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7,

p. 1419, 2016.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints,"

International Journal of Computer Vision, vol. 60, pp. 91–110, 2004.

[8] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on*

Computer Vision and Pattern Recognition, vol. 1, pp. 886–893, 2005.

[9] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110,

pp. 346–359, 2008.

[10] S. P. Mohanty, D. P. Hughes, and M. Salathe', "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7,

p. 1419, 2016.

[11] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and electronics in agriculture*, vol. 145, pp. 311– 318, 2018.

[12] 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 and neuroscience*,

vol. 2016, 2016.

[13] G. Geetharamani and A. Pandian, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers & Electrical Engineering*, vol. 76, pp. 323–338, 2019.

[14] S. B. Jadhav, V. R. Udupi, and S. B. Patil, "Identification of plant diseases using convolutional neural networks," *International Journal of Information Technology*, vol. 13, no. 6, pp. 2461–2470, 2021.

[15] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, "Deep learning for image-based cassava disease detection," *Frontiers in plant science*, vol. 8, p. 1852, 2017.

[16] S. Y. Yadhav, T. Senthilkumar, S. Jayanthy, and J. J. A. Kovilpil- lai, "Plant disease detection and classification using cnn model with optimized activation function," in *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pp. 564– 569, IEEE, 2020.

[17] M. Akila and P. Deepan, "Detection and classification of plant leaf diseases by using deep learning algorithm," *International Journal of Engineering Research & Technology (IJERT)*, vol. 6, pp. 1–5, 2018.

[18] R. R, India., M. S H, and India., "Plant disease detection and classifi- cation using cnn," vol. 10, Sep 2021.

[19] C. Wang, D. Chen, H. Lin, B. Liu, C. Zeng, D. Chen, and G. Zhang, "Pulmonary image classification based on inception-v3 transfer learning model," *IEEE Access*, vol. PP, pp. 1–1, 10 2019.

Appendix 1

Codes

```
import tensorflow as tf

from tensorflow.keras import models, layers

import matplotlib.pyplot as plt
```

In [3]:

```
IMAGE_SIZE = 256

BATCH_SIZE = 32

CHANNELS = 3

EPOCHS = 50
```

In [4]:

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "plantvillage_dataset",

    shuffle = True,

    image_size = (IMAGE_SIZE, IMAGE_SIZE),
    batch_size = BATCH_SIZE

)
```

```
Found 2152 files belonging to 3 classes.
```

```
class_names = dataset.class_names
class_names
```

```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

```
len(dataset)
```

```
68
```

```
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

In [5]:

Out[5]:

```
(32, 256, 256, 3)

[1 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1]
```

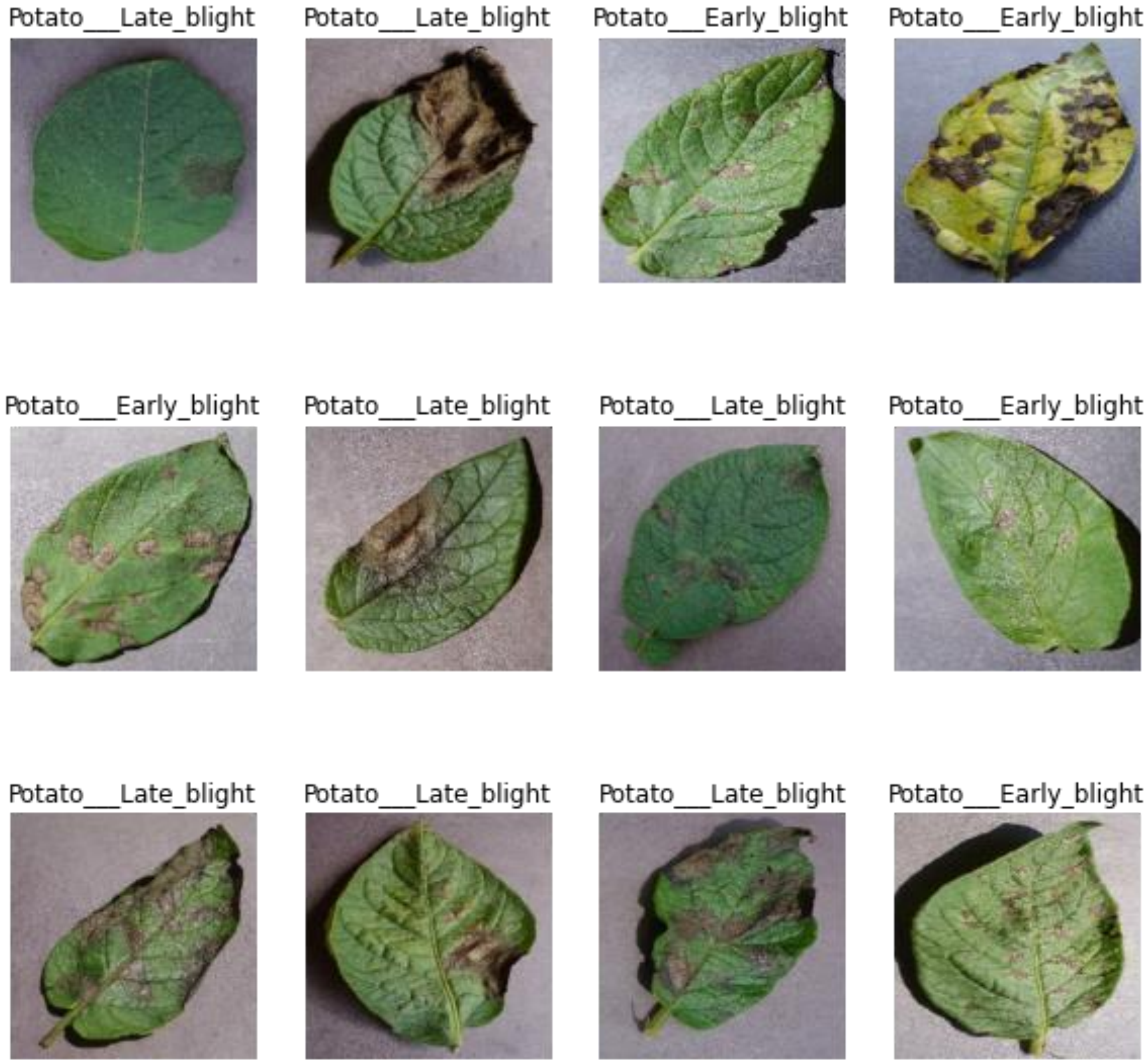
In [6]:

```
plt.figure(figsize=(10, 10))

for image_batch, label_batch in dataset.take(1):

    for i in range(12):

        ax = plt.subplot(3,4,i+1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[label_batch[i]])
        plt.axis("off")
```

Out[6]:

In [7]:

In [8]


Potato___Late_blight
Potato___Late_blight
Potato___Early_blight
Potato___Early_blight
Potato___Early_blight
Potato___Late_blight
Potato___Late_blight
Potato___Early_blight
Potato___Late_blight
Potato___Late_blight
Potato___Late_blight
Potato___Early_blight

```
len(dataset)
```

```
68
```

In [9]: Out[9]:

Data Preprocessing

80% ==> training , 20% ==> 10% validation and 10% test

In [10]:

```
train_size = 0.8
len(dataset)*train_size
```

```
54.400000000000006
```

Out[10]:

In [11]:

```
train_ds= dataset.take(54)
```

```
len(train_ds)
```

```
54
```

```
test_ds = dataset.skip(54)

len(test_ds)
```

```
14
```

```
val_size = 0.1
len(dataset)*val_size
```

```
6.800000000000001
```

```
val_ds = test_ds.take(6)

len(val_ds)
```

```
6
```

```
test_ds = test_ds.skip(6)

len(test_ds)
```

```
8
```

```
def get_dataset_partitions_tf(ds, train_split= 0.8, val_split=0.1,
test_split=0.1,shuffle= True, shuffle_size= 10000):


    ds_size = len(ds)


    if shuffle:

        ds = ds.shuffle(shuffle_size, seed=12) train_size =
    int(train_split * ds_size) val_size = int(val_split *
    ds_size)


    train_ds = ds.take(train_size)
```

Out[11

]: In

[12]:

```
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

Out[12

]: In

[13]:

```
train_ds, val_ds, test_ds= get_dataset_partitions_tf(dataset)

len(train_ds)

54 len(val_ds)

6 len(test_ds)

8
```

Out[13

]: In

[14]:

Out[14

]: In

[15]:

Out[15

]: In

[16]:

In [21]:

In [22]:

In [17]: In [18]: Out[18]: In [19]: Out[19]: In [20]: Out[20]:

Data Resize, Rescale and data augmentation

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)


resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)

])
```

In [23]:

```
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2),


])
```

CNN

In [24]:

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)

n_classes = 3


model = models.Sequential([
    resize_and_rescale,
    data_augmentation,

    layers.Conv2D(32,(3,3),activation = 'relu', input_shape = input_shape),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),

    layers.MaxPooling2D(2,2),
    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation = 'relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation = 'relu'),

    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation = 'relu'),
    layers.MaxPooling2D(2,2),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),

])

model.build(input_shape = input_shape)
```

In [25]:

```
model.summary()

Model: "sequential_2"
_________________________________________________________________

 Layer (type)                Output Shape              Param #
=================================================================

 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0
 )

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0
 2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0
 2D)

 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0
 2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0
 2D)


 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448

 dense_1 (Dense)             (32, 3)                   195

=================================================================
Total params: 183,747

Trainable params: 183,747

Non-trainable params: 0
_________________________________________________________________
```

In [26]:

```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = ['accuracy']
)
```

In [27]:

```
history = model.fit(

    train_ds,

    epochs = EPOCHS,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data = val_ds

)
```

```
Epoch 1/50

54/54 [==============================] - 140s 3s/step - loss: 0.8913 - accuracy: 0.5231

- val_loss: 0.8553 - val_accuracy: 0.6615
```

```
Epoch 2/50
54/54 [==============================] - 121s 2s/step - loss: 0.7214 - accuracy: 0.6823
- val_loss: 0.6173 - val_accuracy: 0.6927

Epoch 3/50

54/54 [==============================] - 122s 2s/step - loss: 0.4325 - accuracy: 0.8293

- val_loss: 0.3246 - val_accuracy: 0.8802

Epoch 4/50

54/54 [==============================] - 123s 2s/step - loss: 0.3567 - accuracy: 0.8634

- val_loss: 0.3151 - val_accuracy: 0.8490

Epoch 5/50

54/54 [==============================] - 123s 2s/step - loss: 0.3249 - accuracy: 0.8733

- val_loss: 0.2598 - val_accuracy: 0.8438

Epoch 6/50

54/54 [==============================] - 123s 2s/step - loss: 0.2683 - accuracy: 0.8976

- val_loss: 0.2162 - val_accuracy: 0.9010

Epoch 7/50

54/54 [==============================] - 122s 2s/step - loss: 0.2821 - accuracy: 0.8883

- val_loss: 0.2070 - val_accuracy: 0.9271

Epoch 8/50

54/54 [==============================] - 122s 2s/step - loss: 0.2135 - accuracy: 0.9230

- val_loss: 0.1664 - val_accuracy: 0.9427

Epoch 9/50

54/54 [==============================] - 122s 2s/step - loss: 0.1507 - accuracy: 0.9410

- val_loss: 0.1473 - val_accuracy: 0.9375

Epoch 10/50

54/54 [==============================] - 123s 2s/step - loss: 0.1978 - accuracy: 0.9190

- val_loss: 0.2302 - val_accuracy: 0.8906

Epoch 11/50

54/54 [==============================] - 123s 2s/step - loss: 0.1249 - accuracy: 0.9514

- val_loss: 0.3114 - val_accuracy: 0.8906

Epoch 12/50

54/54 [==============================] - 123s 2s/step - loss: 0.0970 - accuracy: 0.9647

- val_loss: 0.1321 - val_accuracy: 0.9427

Epoch 13/50
```

```
 54/54 [==============================] - 123s 2s/step - loss: 0.0820 - accuracy: 0.9751
 - val_loss: 0.3659 - val_accuracy: 0.9010
 Epoch 14/50
 54/54 [==============================] - 123s 2s/step - loss: 0.0684 - accuracy: 0.9716
 - val_loss: 0.3340 - val_accuracy: 0.8958
 Epoch 15/50
54/54 [==============================] -    123s  2s/step  - loss:  0.0900  - accuracy:  0.9722
- val_loss: 0.2308 - val_accuracy: 0.9167
Epoch 16/50
54/54 [==============================] -    122s  2s/step  - loss:  0.0603  - accuracy:  0.9745
- val_loss: 0.3264 - val_accuracy: 0.9010
Epoch 17/50
54/54 [==============================] -    122s  2s/step  - loss:  0.1189  - accuracy:  0.9549
- val_loss: 0.0672 - val_accuracy: 0.9792
Epoch 18/50
54/54 [==============================] -    122s  2s/step  - loss:  0.0737  - accuracy:  0.9803
- val_loss: 0.2375 - val_accuracy: 0.9323
Epoch 19/50
54/54 [==============================] -    122s  2s/step  - loss:  0.0605  - accuracy:  0.9797
- val_loss: 0.1590 - val_accuracy: 0.9375
Epoch 20/50


 54/54 [==============================] - 122s 2s/step - loss: 0.0539 - accuracy: 0.9826
 [illegible]
 Epoch 21/50
 54/54 [==============================] - 122s 2s/step - loss: 0.0775 - accuracy: 0.9734
 [illegible]
 Epoch 22/50
 54/54 [==============================] - 122s 2s/step - loss: 0.0635 - accuracy: 0.9769
 [illegible]
 Epoch 23/50
 54/54 [==============================] - 121s 2s/step - loss: 0.0378 - accuracy: 0.9844
 - val_loss: 0.0633 - val_accuracy: 0.9740
 Epoch 24/50
 54/54 [==============================] - 122s 2s/step - loss: 0.0595 - accuracy: 0.9769
 - val_loss: 0.0613 - val_accuracy: 0.9740
 Epoch 25/50
 54/54 [==============================] - 122s 2s/step - loss: 0.0568 - accuracy: 0.9797
 - val_loss: 0.2323 - val_accuracy: 0.9427
 Epoch 26/50
 54/54 [==============================] - 1687s 32s/step - loss: 0.0443 - accuracy: 0.986
```

```
7 - val_loss: 0.0980 - val_accuracy: 0.9583
Epoch 27/50
54/54 [==============================] - 117s 2s/step - loss: 0.0387 - accuracy: 0.9873
- val_loss: 0.1231 - val_accuracy: 0.9479
Epoch 28/50
54/54 [==============================] - 118s 2s/step - loss: 0.0599 - accuracy: 0.9769
- val_loss: 0.1740 - val_accuracy: 0.9323
Epoch 29/50
54/54 [==============================] - 118s 2s/step - loss: 0.0305 - accuracy: 0.9873
- val_loss: 0.1134 - val_accuracy: 0.9531
Epoch 30/50
54/54 [==============================] - 119s 2s/step - loss: 0.0315 - accuracy: 0.9884
- val_loss: 0.0605 - val_accuracy: 0.9688
Epoch 31/50
54/54 [==============================] - 120s 2s/step - loss: 0.0403 - accuracy: 0.9878
- val_loss: 0.1796 - val_accuracy: 0.9479
Epoch 32/50
54/54 [==============================] - 120s 2s/step - loss: 0.0359 - accuracy: 0.9873
- val_loss: 0.0797 - val_accuracy: 0.9583
Epoch 33/50
54/54 [==============================] - 120s 2s/step - loss: 0.0702 - accuracy: 0.9774
- val_loss: 0.2233 - val_accuracy: 0.9062
Epoch 34/50
54/54 [==============================] - 120s 2s/step - loss: 0.0453 - accuracy: 0.9838
- val_loss: 0.1175 - val_accuracy: 0.9635
Epoch 35/50
54/54 [==============================] - 121s 2s/step - loss: 0.0271 - accuracy: 0.9902
- val_loss: 0.1337 - val_accuracy: 0.9531
Epoch 36/50
54/54 [==============================] - 121s 2s/step - loss: 0.0256 - accuracy: 0.9913
- val_loss: 0.0051 - val_accuracy: 1.0000
Epoch 37/50
54/54 [==============================] - 121s 2s/step - loss: 0.0475 - accuracy: 0.9803
- val_loss: 0.2365 - val_accuracy: 0.9167
```

```
Epoch 38/50
54/54 [==============================] - 121s 2s/step - loss: 0.0304 - accuracy: 0.9861
- val_loss: 0.1747 - val_accuracy: 0.9479
```

```
Epoch 39/50
54/54 [==============================] - 122s 2s/step - loss: 0.0421 - accuracy: 0.9838
- val_loss: 0.1660 - val_accuracy: 0.9479
Epoch 40/50
54/54 [==============================] - 121s 2s/step - loss: 0.0747 - accuracy: 0.9745
- val_loss: 0.1338 - val_accuracy: 0.9375
Epoch 41/50
54/54 [==============================] - 2979s 56s/step - loss: 0.0429 - accuracy: 0.983
8 - val_loss: 0.0645 - val_accuracy: 0.9688
Epoch 42/50
                                            119s 2s/step - loss: 0.0218 - accuracy: 0.9925

   54/54 [==============================] - 118s 2s/step - loss: 0.0361 - accuracy: 0.9850
```

```
scores = model.evaluate(test_ds)
8/8 [==============================] - 7s 639ms/step - loss: 0.0162 - accuracy: 0.9961
```

In [29]:

```
scores
```

```
[0.01620321162045002, 0.99609375]
```

```
history
```

```
<keras.callbacks.History at 0x1ce25f960a0>
```

```
history.params
```

Out[29]:

```
{'verbose': 1, 'epochs': 50, 'steps': 54}
```

In [30]:

Out[30]:

```
history.history.keys()
```

In [31]:

Out[31]:

In [32]:

Out[32]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
history.history['accuracy']
```

```
[0.5231481194496155,
 0.6822916865348816,
 0.8292824029922485,
 0.8634259104728699,
 0.8732638955116272,
 0.8975694179534912,
 0.8883101940155029,
 0.9230324029922485,
 0.9409722089767456,
 0.9189814925193787,
 0.9513888955116272,
 0.9646990895271301,
 0.9751157164573669,
 0.9716435074806213,
 0.9722222089767456,
 0.9745370149612427,
 0.9548611044883728,
 0.9803240895271301,
 0.9797453880310059,
 0.9826388955116272,
 0.9733796119689941,
 0.9768518805503845,
 0.984375,
 0.9768518805503845,
 0.9797453880310059,
 0.9866898059844971,
 0.9872685074806213,
 0.9768518805503845,
 0.9872685074806213,
 0.9884259104728699,
 0.9878472089767456,
 0.9872685074806213,
 0.9774305820465088,
 0.9837962985038757,
 0.9901620149612427,
 0.9913194179534912,
 0.9803240895271301,
 0.9861111044883728,
 0.9837962985038757,
 0.9745370149612427,
 0.9837962985038757,
 0.9924768805503845,
 0.9849537014961243,
 0.9913194179534912,
 0.9942129850387573,
 0.9930555820465088,
 0.9849537014961243,
 0.9768518805503845,
 0.9907407164573669,
 0.9849537014961243]
```

In [33]:

Out[33]:

In [34]:

```
acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

loss = history.history['loss']

val_loss = history.history['val_loss']


plt.figure(figsize = (8,8))

plt.subplot(1,2,1)

plt.plot(range(EPOCHS), acc, label = 'Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label = 'Validation Accuracy')
plt.legend(loc = 'lower right')

plt.title('Training and Validation Accuracy')

plt.axis("on")
```

```
plt.subplot(1,2,2)

plt.plot(range(EPOCHS), loss, label = 'Training Loss')
plt.plot(range(EPOCHS), val_loss, label = 'Validation Loss')
plt.legend(loc = 'upper right')

plt.title('Training and Validation Loss')

plt.axis("on")

plt.show()
```

```
54/54 [==============================] -    122s 2s/step - loss:  0.0539 - accuracy:  0.9826
- val_loss: 0.2790 - val_accuracy: 0.9167
Epoch 21/50
54/54 [==============================] -    122s 2s/step - loss:  0.0775 - accuracy:  0.9734
- val_loss: 0.1957 - val_accuracy: 0.9323
Epoch 22/50
54/54 [==============================] -    122s 2s/step - loss:  0.0635 - accuracy:  0.9769
- val_loss: 0.0725 - val_accuracy: 0.9688
Epoch 23/50
54/54 [==============================] -    121s 2s/step - loss:  0.0378 - accuracy:  0.9844
- val_loss: 0.0633 - val_accuracy: 0.9740

Epoch 24/50

54/54 [==============================] - 122s 2s/step - loss: 0.0595 - accuracy: 0.9769

- val_loss: 0.0613 - val_accuracy: 0.9740

Epoch 25/50

54/54 [==============================] - 122s 2s/step - loss: 0.0568 - accuracy: 0.9797

- val_loss: 0.2323 - val_accuracy: 0.9427

Epoch 26/50

54/54 [==============================] - 1687s 32s/step - loss: 0.0443 - accuracy: 0.986

7 - val_loss: 0.0980 - val_accuracy: 0.9583

Epoch 27/50

54/54 [==============================] - 117s 2s/step - loss: 0.0387 - accuracy: 0.9873

- val_loss: 0.1231 - val_accuracy: 0.9479

Epoch 28/50

54/54 [==============================] - 118s 2s/step - loss: 0.0599 - accuracy: 0.9769

- val_loss: 0.1740 - val_accuracy: 0.9323

Epoch 29/50

54/54 [==============================] - 118s 2s/step - loss: 0.0305 - accuracy: 0.9873

- val_loss: 0.1134 - val_accuracy: 0.9531

Epoch 30/50

54/54 [==============================] - 119s 2s/step - loss: 0.0315 - accuracy: 0.9884

- val_loss: 0.0605 - val_accuracy: 0.9688

Epoch 31/50

54/54 [==============================] - 120s 2s/step - loss: 0.0403 - accuracy: 0.9878

- val_loss: 0.1796 - val_accuracy: 0.9479

Epoch 32/50
```

```
54/54 [==============================] - 120s 2s/step - loss: 0.0359 - accuracy: 0.9873
- val_loss: 0.0797 - val_accuracy: 0.9583
Epoch 33/50
54/54 [==============================] - 120s 2s/step - loss: 0.0702 - accuracy: 0.9774
- val_loss: 0.2233 - val_accuracy: 0.9062
Epoch 34/50
54/54 [==============================] - 120s 2s/step - loss: 0.0453 - accuracy: 0.9838
- val_loss: 0.1175 - val_accuracy: 0.9635
Epoch 35/50
54/54 [==============================] - 121s 2s/step - loss: 0.0271 - accuracy: 0.9902
- val_loss: 0.1337 - val_accuracy: 0.9531
Epoch 36/50
54/54 [==============================] - 121s 2s/step - loss: 0.0256 - accuracy: 0.9913
- val_loss: 0.0051 - val_accuracy: 1.0000
Epoch 37/50
54/54 [==============================] - 121s 2s/step - loss: 0.0475 - accuracy: 0.9803
- val_loss: 0.2365 - val_accuracy: 0.9167
Epoch 38/50
54/54 [==============================] - 121s 2s/step - loss: 0.0304 - accuracy: 0.9861
- val_loss: 0.1747 - val_accuracy: 0.9479
```

```
Epoch 39/50

54/54 [==============================] - 122s 2s/step - loss: 0.0421 - accuracy: 0.9838

- val_loss: 0.1660 - val_accuracy: 0.9479

Epoch 40/50

54/54 [==============================] - 121s 2s/step - loss: 0.0747 - accuracy: 0.9745

- val_loss: 0.1338 - val_accuracy: 0.9375

Epoch 41/50

54/54 [==============================] - 2979s 56s/step - loss: 0.0429 - accuracy: 0.983

8 - val_loss: 0.0645 - val_accuracy: 0.9688

Epoch 42/50

54/54 [==============================] -   11   2s/ste   -       0.02    -          0.9925
- val_loss: 0.1281 - val_accuracy: 0.9688  9s   p        loss:   18      accuracy:
Epoch 43/50
54/54 [==============================] -   11   2s/ste   -       0.03    -          0.9850
- val loss: 0.0119 - val accuracy: 1.0000
Epoch 44/50
54/54 [==============================] -   11   2s/ste   -       0.02    -          0.9913
- val loss: 0.0447 - val accuracy: 0.9792
Epoch 45/50
54/54 [==============================] -   11   2s/ste   -       0.01    -          0.9942
- val loss: 0.0049 - val accuracy: 1.0000
Epoch 46/50
54/54 [==============================] -   12   2s/ste   -       0.02    -          0.9931
- val loss: 0.0400 - val accuracy: 0.9792
Epoch 47/50
54/54 [==============================] -   12   2s/ste   -       0.03    -          0.9850
- val loss: 0.5477 - val accuracy: 0.8906
Epoch 48/50
54/54 [==============================] -   12   2s/ste   -       0.06    -          0.9769
- val loss: 0.0062 - val accuracy: 1.0000
Epoch 49/50
54/54 [==============================] -   12   2s/ste   -       0.03    -          0.9907
- val loss: 0.0077 - val accuracy: 1.0000
Epoch 50/50
54/54 [==============================] -   12   2s/ste   -       0.04    -          0.9850
- val_loss: 0.0061 - val_accuracy: 1.0000
```

```
scores = model.evaluate(test_ds)

8/8 [==============================] - 7s 639ms/step - loss: 0.0162 - accuracy: 0.9961
```

In [29]:

```
scores
```

```
[0.01620321162045002, 0.99609375]
```

```
history
```

```
<keras.callbacks.History at 0x1ce25f960a0>
```

Out[29]:

```
history.params
```

In [30]:

Out[30]:

```
{'verbose': 1, 'epochs': 50, 'steps': 54}
```

In [31]:

```
history.history.keys()
```

Out[31]:

In [32]:

Out[32]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

history.history['accuracy']

[0.5231481194496155,
 0.6822916865348816,
 0.8292824029922485,
 0.8634259104728699,
 0.8732638955116272,
 0.8975694179534912,
 0.8883101940155029,
 0.9230324029922485,
 0.9409722089767456,
 0.9189814925193787,
 0.9513888955116272,
 0.9646990895271301,
 0.9751157164573669,
 0.9716435074806213,
 0.9722222089767456,
 0.9745370149612427,
 0.9548611044883728,
 0.9803240895271301,
 0.9797453880310059,
 0.9826388955116272,
 0.9733796119689941,
 0.9768518805503845,
 0.984375,
 0.9768518805503845,
 0.9797453880310059,
 0.9866898059844971,
 0.9872685074806213,
 0.9768518805503845,
 0.9872685074806213,
 0.9884259104728699,
 0.9878472089767456,
```

```
0.987
26850
74806
213,
0.977
43058
20465
088,
0.983
79629
85038
757,
0.990
16201
49612
427,
0.991
31941
79534
912,
0.980
32408
95271
301,
0.986
11110
44883
728,
0.983
79629
85038
757,
0.974
53701
49612
427,
0.983
79629
85038
757,
0.992
47688
05503
845,
0.984
95370
14961
243,
0.9913194179534912,
0.9942129850387573,
0.9930555820465088,
0.9849537014961243,
0.9768518805503845,
0.9907407164573669,
0.9849537014961243]
```

In [33]:

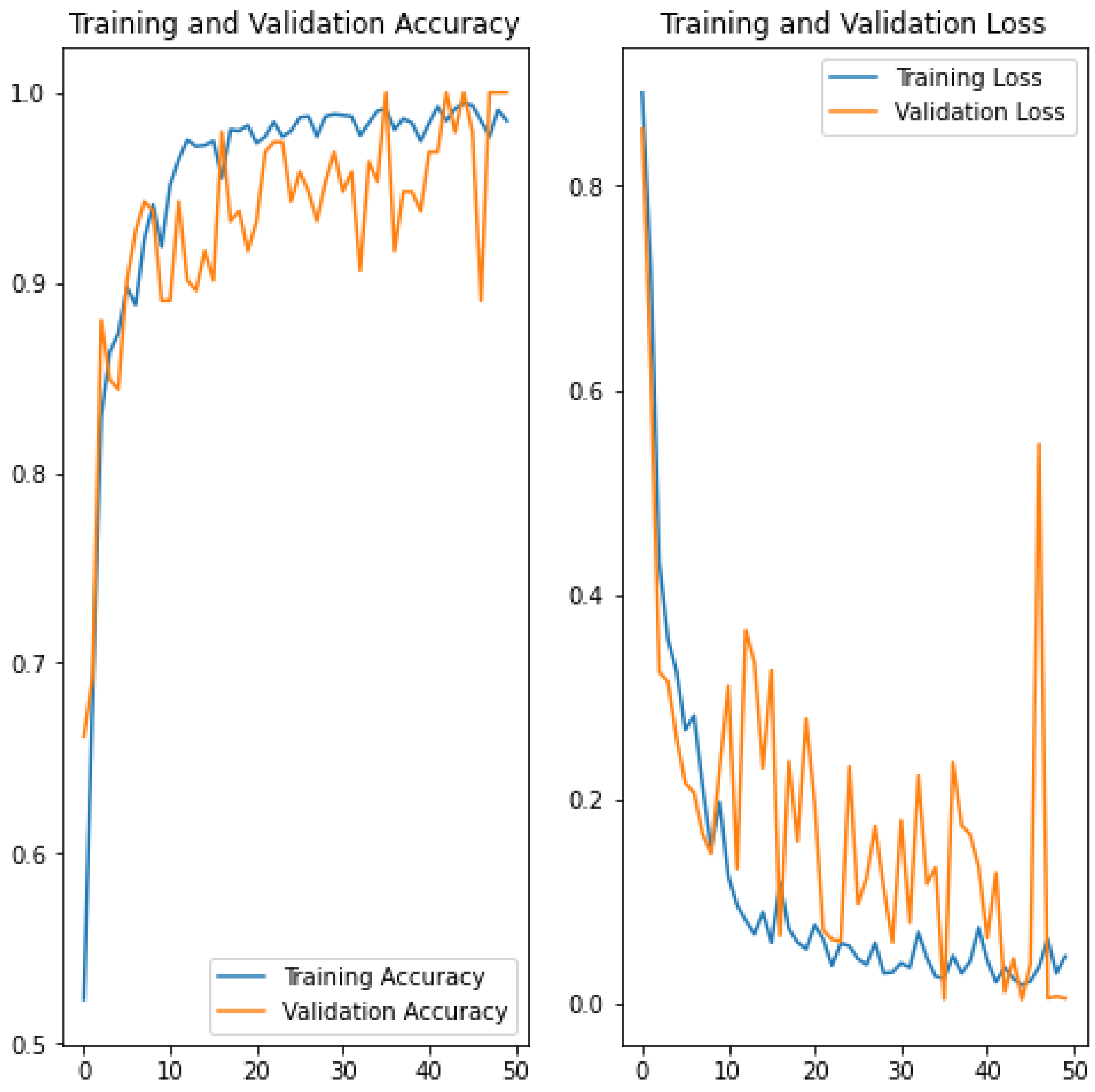
Out[33]:

In [34]:

```
acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

loss = history.history['loss']

val_loss = history.history['val_loss']
```

In [39]:

```
plt.figure(figsize = (8,8))

plt.subplot(1,2,1)

plt.plot(range(EPOCHS), acc, label = 'Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label = 'Validation Accuracy')
plt.legend(loc = 'lower right')

plt.title('Training and Validation Accuracy')

plt.axis("on")


plt.subplot(1,2,2)

plt.plot(range(EPOCHS), loss, label = 'Training Loss')
plt.plot(range(EPOCHS), val_loss, label = 'Validation Loss')
plt.legend(loc = 'upper right')

plt.title('Training and Validation Loss')

plt.axis("on")

plt.show()
```



```
import numpy as np

for images_batch, labels_batch in test_ds.take(1):
    first_image = image_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
```

```
p
l
t
.
i
m
s
h
o
w
(
f
i
```

```
    rst_image)
    print("Actual label:",class_names[first_label])
```

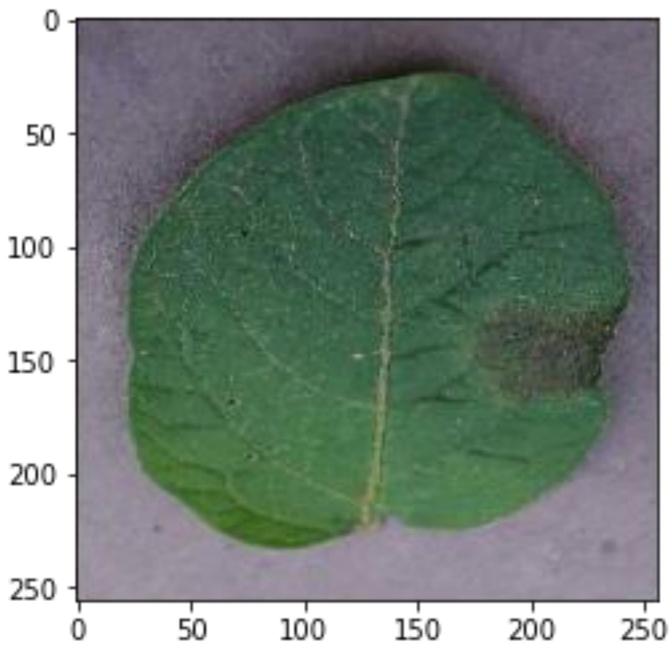
In [40]:

In [41]:

```
    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict
Actual label: Potato___Early_blight
```

```
1/1 [==============================] - 1s 644ms/step
predicted label: Potato___Early_blight
```



```
def predict(model, img):

    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())

    img_array = tf.expand_dims(img_array, 0)


    predictions = model.predict(img_array)


    predicted_class = class_names[np.argmax(predictions[0])]

    confidence = round(100 * (np.max(predictions[0])), 2)
```

In [42]:

```
    return predicted_class, confidence

plt.figure(figsize=(15, 15))
for images , labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i+1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual:{actual_class},\n Predicted: {predicted_class}, \n
Confidence:{confidence}%")

        plt.axis("off")
```

In

```
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
```

```
1/1 [==============================] - 0s 34ms/step
 1/1 [==============================] - 0s 35ms/step
```

Actual: Potato_ Early_blight. Predicted: Potato_ Early_blight. Confidence: 100.0%

Actual: Potato _Late_blight. Predicted: Potato_ Late_blight. Confidence:99.97%

Actual: Potato _Late_blight. Predicted: Potato_Late_blight. Confidence:99.49%

Actual: Potato_Late_blight. Predicted: Potato_ Late_blight.Confidence:100.0%

Actual:Potato_Early_blight. Predicted: Potato_ Early_ blight. Confidence:99.99%

Actual: Potato_ Early_ blight. Predicted: Potato_ Early_ blight.confidence :99.89%

Actual: Potato _Late_blight. Predicted: Potato_Late_blight.Confidence 99.98%

Actual: Potato _Early_ blight. Predicted: Potato_Early_blight. Confidence:100.0%

Actual:Potato_healthy. Predicted: Potato_healthy. Confidence:99.18%

```
model_version = 1
model.save(f"../models/{model_version}")
```

In [44]:

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compile
d_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compi
led_convolution_op while saving (showing 5 of 6). These functions will not be directly c
allable after loading.
```

```
INFO:tensorflow:Assets written to: ../models/1\assets
```

```
INFO:tensorflow:Assets written to: ../models/1\assets
```

VGG16

In [45]:

```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
```

In [46]:

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)

# input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)

n_classes = 3




IMG_SHAPE = (IMAGE_SIZE, IMAGE_SIZE, 3)

VGG16_MODEL=VGG16(input_shape=IMG_SHAPE, include_top=False, weights='imagenet')


for layer in VGG16_MODEL.layers[:-2]:

            layer.trainable = False


global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

hidden_1 = layers.Dense(4096, activation='relu')
hidden_2 = layers.Dense(1072, activation='relu')
dropout = layers.Dropout(0.2)

prediction_layer = tf.keras.layers.Dense(n_classes, activation='softmax')


model_vgg16 = tf.keras.Sequential([
  resize_and_rescale,
  data_augmentation,

  VGG16_MODEL,

  global_average_layer,
  hidden_1,

  hidden_2,

  dropout,
  prediction_layer

])


model_vgg16.build(input_shape = input_shape)


model_vgg16.summary()
Model: "sequential_3"
```

In [47]:

```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0


 vgg16 (Functional)          (None, 8, 8, 512)         14714688


 global_average_pooling2d (G  (32, 512)                0
 lobalAveragePooling2D)


 dense_2 (Dense)             (32, 4096)                2101248

 dense_3 (Dense)             (32, 1072)                4391984

 dropout (Dropout)           (32, 1072)                0
 dense_4 (Dense)             (32, 3)                   3219


=================================================================
Total params: 21,211,139

Trainable params: 8,856,259

Non-trainable params: 12,354,880
_________________________________________________________________
```

In [48]:

```
model_vgg16.compile(

    optimizer = 'adam',

    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = ['accuracy']


)
```

In [49]:

```
history = model_vgg16.fit(

    train_ds,

    epochs = EPOCHS,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data = val_ds


)
Epoch 1/50

54/54 [==============================] - 752s 14s/step - loss: 0.5835 - accuracy: 0.7801

- val_loss: 0.2619 - val_accuracy: 0.8906

Epoch 2/50

54/54 [==============================] - 753s 14s/step - loss: 0.2197 - accuracy: 0.9248

- val_loss: 0.3292 - val_accuracy: 0.8438

Epoch 3/50

54/54 [==============================] - 4379s 82s/step - loss: 0.2035 - accuracy: 0.917

8 - val_loss: 0.0614 - val_accuracy: 0.9792

Epoch 4/50

54/54 [==============================] - 795s 15s/step - loss: 0.1426 - accuracy: 0.9473

- val_loss: 0.0829 - val_accuracy: 0.9740

Epoch 5/50

54/54 [==============================] - 750s 14s/step - loss: 0.1204 - accuracy: 0.9583
```

```
- val_loss: 0.2692 - val_accuracy: 0.8698

14s/step - loss: 0.1260 - accuracy: 0.9560


Epoch 6/50

54/54 [==============================] - 756s 14s/step - loss: 0.1095 - accuracy: 0.9583

- val_loss: 0.0330 - val_accuracy: 0.9948

Epoch 7/50

54/54 [==============================] - 11885s 224s/step - loss: 0.1029 - accuracy: 0.9

647 - val_loss: 0.0257 - val_accuracy: 1.0000

Epoch 8/50

54/54 [==============================] - 736s 14s/step - loss: 0.1260 - accuracy: 0.9560




- val_loss: 0.1958 - val_accuracy: 0.9115

Epoch 9/50

- val_loss: 0.0374 - val_accuracy: 0.9792
Epoch 10/50
54/54 [==============================] - 808s 15s/step - loss: 0.0632 - accuracy: 0.9780

- val_loss: 0.0240 - val_accuracy: 0.9948
Epoch 11/50
54/54 [==============================] - 762s 14s/step - loss: 0.0927 - accuracy: 0.9670

- val_loss: 0.0306 - val_accuracy: 0.9948
Epoch 12/50
```

```
54/54 [==============================] - 789s 15s/step - loss: 0.0650 - accuracy: 0.9774
Epoch 13/50
54/54 [==============================] - 779s 14s/step - loss: 0.0806 - accuracy: 0.9716
Epoch 14/50
54/54 [==============================] - 753s 14s/step - loss: 0.0829 - accuracy: 0.9699
Epoch 15/50
54/54 [==============================] - 801s 15s/step - loss: 0.0734 - accuracy: 0.9711
Epoch 16/50
54/54 [==============================] - 779s 14s/step - loss: 0.0689 - accuracy: 0.9757
Epoch 17/50
54/54 [==============================] - 750s 14s/step - loss: 0.0633 - accuracy: 0.9797
Epoch 18/50
54/54 [==============================] - 942s 18s/step - loss: 0.0798 - accuracy: 0.9682
Epoch 19/50
54/54 [==============================] - 922s 17s/step - loss: 0.0505 - accuracy: 0.9809
Epoch 20/50
54/54 [==============================] - 945s 18s/step - loss: 0.0491 - accuracy: 0.9815
Epoch 21/50
54/54 [==============================] - 917s 17s/step - loss: 0.0637 - accuracy: 0.9786
Epoch 22/50
54/54 [==============================] - 940s 17s/step - loss: 0.0468 - accuracy: 0.9821
Epoch 23/50
54/54 [==============================] - 939s 17s/step - loss: 0.0502 - accuracy: 0.9815
Epoch 24/50
54/54 [==============================] - 881s 16s/step - loss: 0.0493 - accuracy: 0.9821
Epoch 25/50
54/54 [==============================] - 731s 14s/step - loss: 0.0423 - accuracy: 0.9838
Epoch 26/50
54/54 [==============================] - 726s 13s/step - loss: 0.0409 - accuracy: 0.9861
Epoch 27/50
54/54 [==============================] - 721s 13s/step - loss: 0.1344 - accuracy: 0.9601
Epoch 28/50
54/54 [==============================] - 724s 13s/step - loss: 0.0687 - accuracy: 0.9676
Epoch 29/50
54/54 [==============================] - 722s 13s/step - loss: 0.0621 - accuracy: 0.9774
Epoch 30/50
```

```
54/54 [==============================] - 19768s 373s/step - loss: 0.0385 - accuracy:
0.9913 - val_loss: 0.0320 - val_accuracy: 0.9844
```

```
Epoch 31/50
54/54 [==============================] -   88   16s/ste   -   0.059   -   0.97
- val_loss: 0.0645 - val_accuracy: 0.9740
Epoch 32/50
54/54 [==============================] -   85   16s/ste   -   0.030   -   0.98
- val_loss: 0.0066 - val_accuracy: 1.0000
Epoch 33/50
54/54 [==============================] -   90   17s/ste   -   0.030   -   0.98
- val_loss: 0.0054 - val_accuracy: 1.0000
Epoch 34/50
54/54 [==============================] -   76   14s/ste   -   0.034   -   0.98
- val_loss: 0.0278 - val_accuracy: 0.9896
Epoch 35/50
54/54 [==============================] -   96   18s/ste   -   0.038   -   0.98
- val_loss: 0.0390 - val_accuracy: 0.9844
Epoch 36/50
54/54 [==============================] -   93   17s/ste   -   0.021   -   0.98
- val_loss: 0.0141 - val_accuracy: 0.9948
Epoch 37/50
54/54 [==============================] -   96   18s/ste   -   0.046   -   0.98
- val_loss: 0.0194 - val_accuracy: 0.9948
Epoch 38/50
54/54 [==============================] -   91   17s/ste   -   0.027   -   0.99
- val_loss: 0.0218 - val_accuracy: 0.9948
Epoch 39/50
54/54 [==============================] -   79   15s/ste   -   0.043   -   0.98
- val_loss: 0.0194 - val_accuracy: 0.9896
Epoch 40/50
54/54 [==============================] -   72   13s/ste   -   0.026   -   0.98
- val_loss: 0.0402 - val_accuracy: 0.9792
Epoch 41/50

54/54 [==============================] - 10355s 195s/step - loss: 0.0272 - accuracy: 0.9

907 - val_loss: 0.0797 - val_accuracy: 0.9688

Epoch 42/50

54/54 [==============================] - 863s 16s/step - loss: 0.0441 - accuracy: 0.9844
- val_loss: 0.0332 - val_accuracy: 0.9844

Epoch 43/50
54/54 [==============================] - 740s 14s/step - loss: 0.0507 - accuracy: 0.9803
- val_loss: 0.0997 - val_accuracy: 0.9583
Epoch 44/50

[illegible]
- val_loss: 0.1494 - val_accuracy: 0.9062
Epoch 45/50

[illegible]
- val_loss: 0.0767 - val_accuracy: 0.9792
Epoch 46/50

[illegible]
- val_loss: 0.0054 - val_accuracy: 1.0000
Epoch 47/50

[illegible]
- val_loss: 0.0144 - val_accuracy: 1.0000
Epoch 48/50

[illegible]
- val_loss: 0.0292 - val_accuracy: 0.9948
Epoch 49/50
```

```
54/54 [==============================] - 796s 15s/step - loss: 0.0452 - accuracy: 0.9809
- val_loss: 0.0209 - val_accuracy: 0.9948
Epoch 50/50
54/54 [==============================] - 739s 14s/step - loss: 0.0489 - accuracy: 0.9832
- val_loss: 0.0071 - val_accuracy: 0.9948
```

In [50]:

```
scores = model_vgg16 .evaluate(test_ds)
```

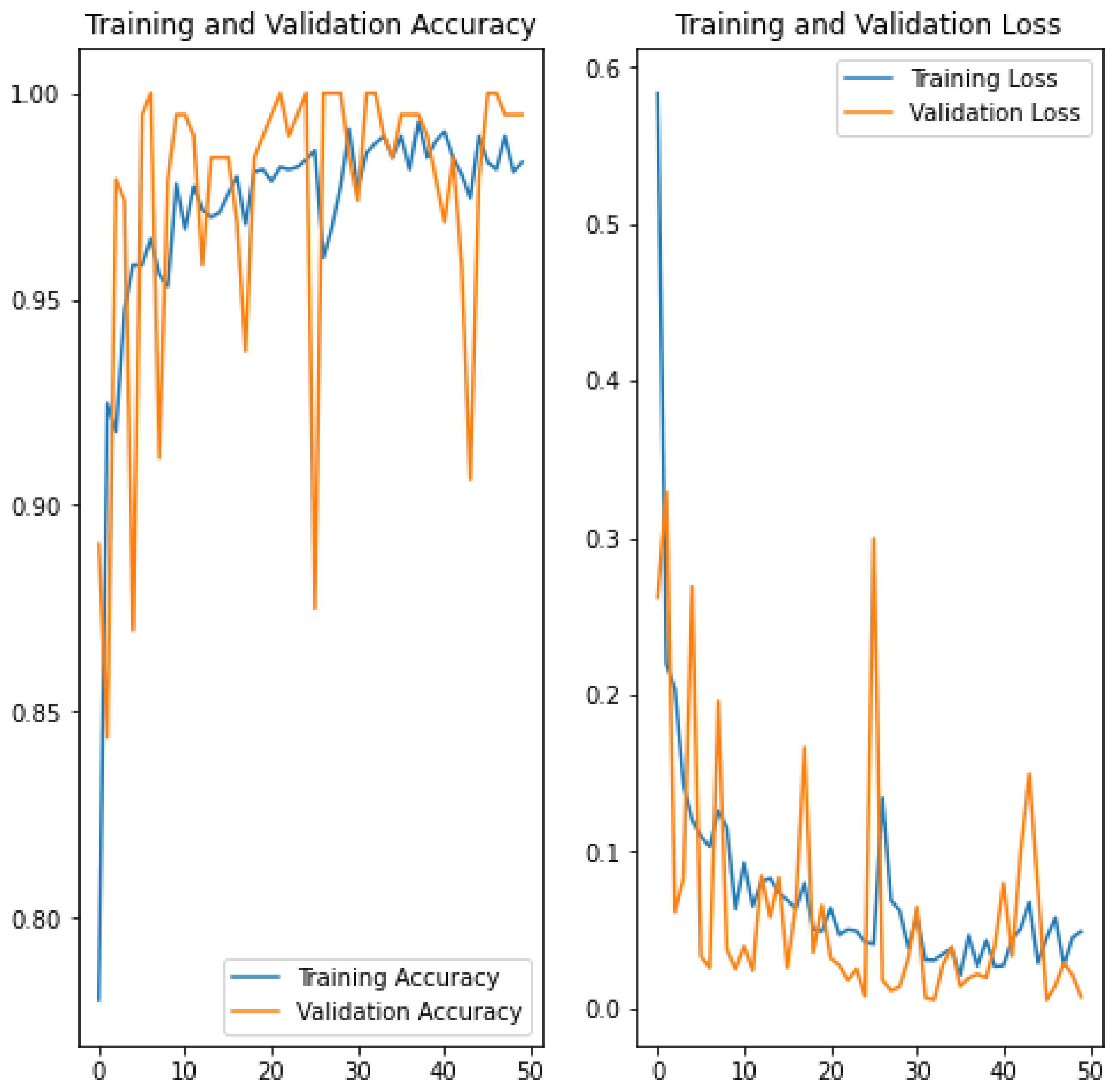
```
8/8 [==============================] - 95s 12s/step - loss: 0.0035 - accuracy: 1.0000
```

In [51]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

In [52]:

```
plt.figure(figsize = (8,8))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS), acc, label = 'Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label = 'Validation Accuracy')
plt.legend(loc = 'lower right')
plt.title('Training and Validation Accuracy')


plt.subplot(1,2,2)
plt.plot(range(EPOCHS), loss, label = 'Training Loss')
plt.plot(range(EPOCHS), val_loss, label = 'Validation Loss')
plt.legend(loc = 'upper right')
plt.title('Training and Validation Loss')
plt.show()
```



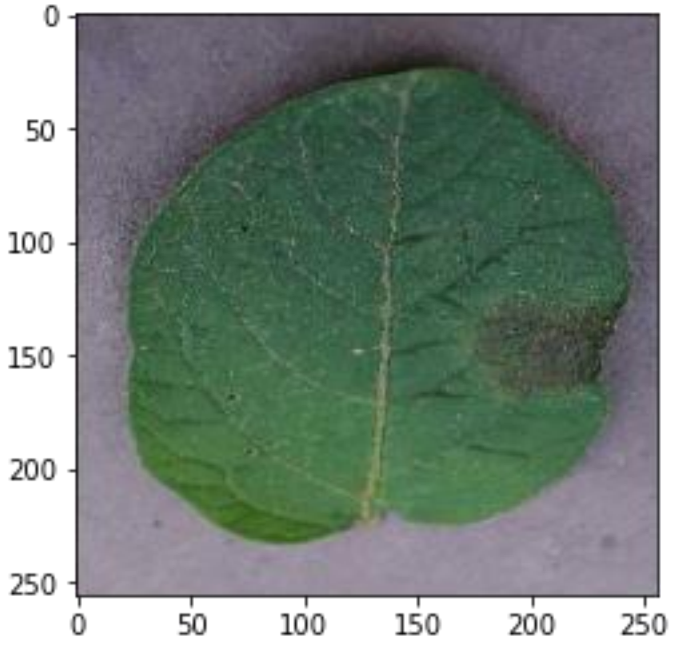
```
for images_batch, labels_batch in test_ds.take(1):
    first_image = image_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()


    print("first image to predict")

    plt.imshow(first_image)

    print("Actual label:",class_names[first_label])


    batch_prediction = model.predict(images_batch)
```

print("predicted

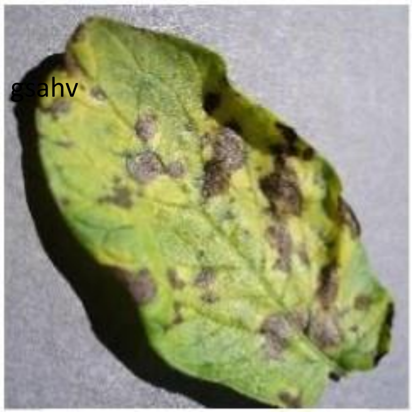
```
    label:",class_names[np.argmax(batch_prediction[0])])
```

In [53]:

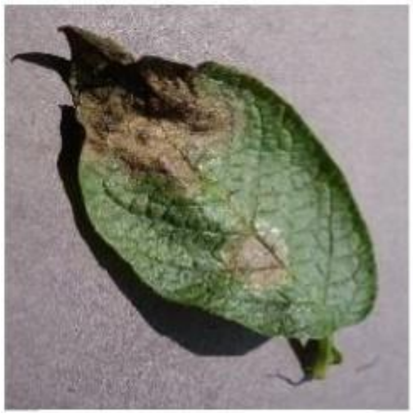
```
first image to predict

Actual label: Potato___Early_blight

1/1 [==============================] - 1s 609ms/step
predicted label: Potato___Early_blight
```



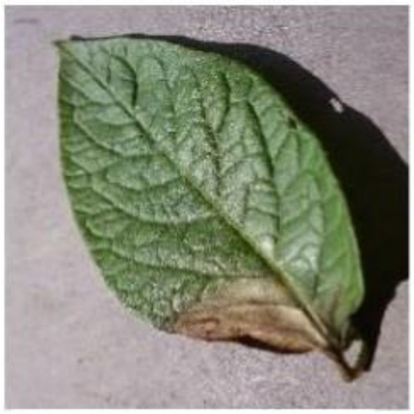
```
def predict(model_vgg16, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

           predictions = model_vgg16.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)

    return predicted_class, confidence
plt.figure(figsize=(15, 15))
for images , labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i+1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model_vgg16, images[i].numpy())
               actual_class = class_names[labels[i]]

        plt.title(f"Actual:{actual_class},\n Predicted: {predicted_class}, \n
Confidence:{confidence}%")

        plt.axis("off")

1/1 [==============================] - 1s 557ms/step
1/1 [==============================] - 0s 367ms/step
1/1 [==============================] - 0s 466ms/step
1/1 [==============================] - 0s 415ms/step
1/1 [==============================] - 0s 473ms/step
1/1 [==============================] - 0s 422ms/step
1/1 [==============================] - 0s 428ms/step
1/1 [==============================] - 0s 447ms/step
1/1 [==============================] - 0s 453ms/step
```



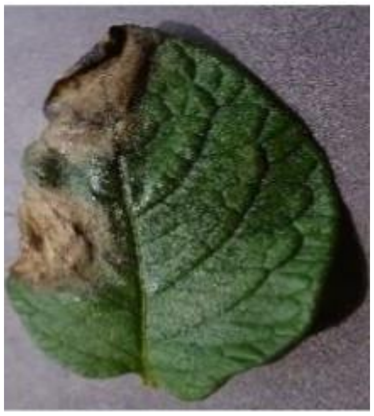
Actual: Potato_   Early _blight, Predicted: Potato_Earty_blight. Confidence:100.0%



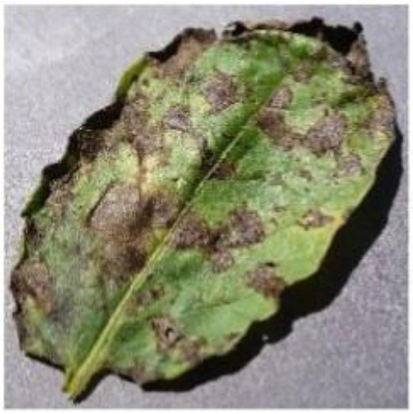
Actual:Potato_Late_blight, Predicted: Potato_   Late_blight,Confidence:100%



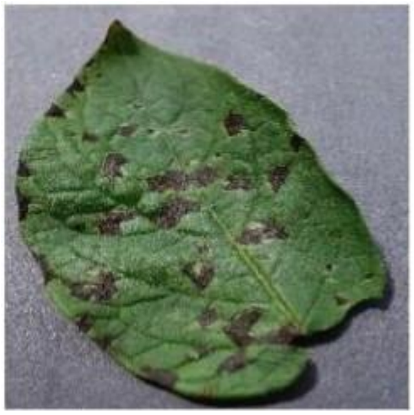
Actual:Potato_Late_blight, Predicted: Potato_Late_blight, Confidence:99.21%



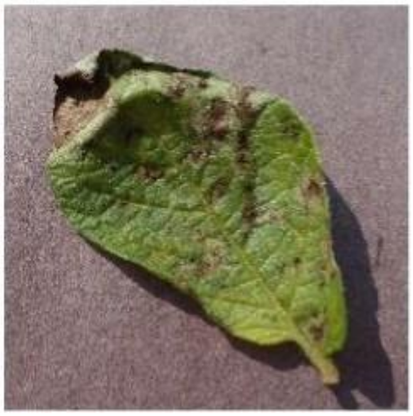
Actual: Potato_Late_ blight, Predicted: Potato_Late_blight Confidence :100%



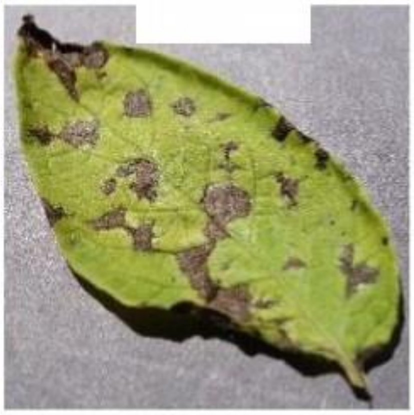
Actual:Potato_Earty_blight. Predicted: Potato_Earty_blight,Confidence:100%



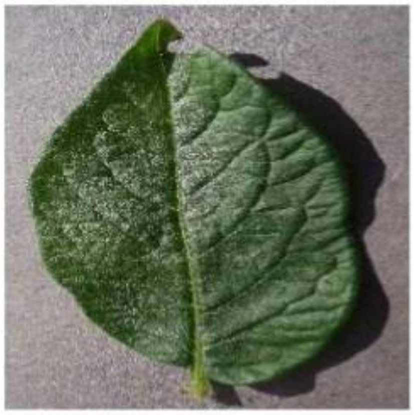
Actual:Potato_Early_blight. Predicted: Potato_Earty_blight.Confidence 100%



Actual:Potato_Late_blight. Predicted: Potato_Late_blight. Confidence:99.99%



Actual:Potato_Earty_blight. Predicted: Potato_Earty_blight. Confidence :100%



1Actual:Potato_healthy.Predicted: Potato_healthy, Confidence:100.0%

Inception Net

In [56]:

In [57]:

```
from tensorflow.keras.applications.inception_v3 import InceptionV3

input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)

n_classes = 3
```

```
IMG_SHAPE = (IMAGE_SIZE, IMAGE_SIZE, 3)
```

```
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=IMG_SHAPE)


for layer in base_model.layers:
    layer.trainable = False




global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
hidden_1 = layers.Dense(1024, activation='relu')
prediction_layer = tf.keras.layers.Dense(n_classes, activation='softmax')




model_inceptionV3 = tf.keras.Sequential([
  resize_and_rescale,
  data_augmentation,
  base_model,
  global_average_layer,
  hidden_1,
  prediction_layer
])
model_inceptionV3.build(input_shape = input_shape)
```

In [58]:

```
model_inceptionV3.summary()
Model: "sequential_4"
_________________________________________________________________

 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0

 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 global_average_pooling2d_1   (32, 2048)               0
 (GlobalAveragePooling2D)
 dense_5 (Dense)             (32, 1024)                2098176

 dense_6 (Dense)             (32, 3)                   3075

=================================================================
Total params: 23,904,035

Trainable params: 2,101,251

Non-trainable params: 21,802,784
_________________________________________________________________
```

In [59]:

```
model_inceptionV3.compile(

    optimizer = "adam",

    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = ['accuracy']


)
```

In [60]:

```
history = model_inceptionV3.fit(

    train_ds,
    epochs = EPOCHS,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data = val_ds

)

Epoch 1/50

54/54 [==============================] - 176s 3s/step - loss: 0.7621 - accuracy: 0.7755

- val_loss: 0.2547 - val_accuracy: 0.8854

Epoch 2/50

54/54 [==============================] - 168s 3s/step - loss: 0.2302 - accuracy: 0.9120

- val_loss: 0.2717 - val_accuracy: 0.9010

Epoch 3/50

54/54 [==============================] - 169s 3s/step - loss: 0.2267 - accuracy: 0.9062

- val_loss: 0.2460 - val_accuracy: 0.9062

Epoch 4/50

54/54 [==============================] - 170s 3s/step - loss: 0.1593 - accuracy: 0.9410

- val_loss: 0.3449 - val_accuracy: 0.8646

Epoch 5/50

54/54 [==============================] - 169s 3s/step - loss: 0.1266 - accuracy: 0.9566

- val_loss: 0.1797 - val_accuracy: 0.9271

Epoch 6/50

54/54 [==============================] - 169s 3s/step - loss: 0.1676 - accuracy: 0.9381

- val_loss: 0.1910 - val_accuracy: 0.9323

Epoch 7/50

54/54 [==============================] - 169s 3s/step - loss: 0.1285 - accuracy: 0.9502

- val_loss: 0.1743 - val_accuracy: 0.9531

Epoch 8/50

54/54 [==============================] - 172s 3s/step - loss: 0.1544 - accuracy: 0.9410

- val_loss: 0.1495 - val_accuracy: 0.9635

Epoch 9/50

54/54 [==============================] - 169s 3s/step - loss: 0.1070 - accuracy: 0.9641

- val_loss: 0.1863 - val_accuracy: 0.9375

Epoch 10/50

54/54 [==============================] - 170s 3s/step - loss: 0.1116 - accuracy: 0.9589

- val_loss: 0.1958 - val_accuracy: 0.9479
```

```
Epoch 11/50

54/54 [==============================] - 169s 3s/step - loss: 0.1124 - accuracy: 0.9612

- val_loss: 0.3160 - val_accuracy: 0.9010

Epoch 12/50

54/54 [==============================] - 170s 3s/step - loss: 0.1212 - accuracy: 0.9549

- val_loss: 0.2371 - val_accuracy: 0.9271

Epoch 13/50

54/54 [==============================] - 169s 3s/step - loss: 0.1353 - accuracy: 0.9491

- val_loss: 0.1500 - val_accuracy: 0.9583

Epoch 14/50

54/54 [==============================] - 169s 3s/step - loss: 0.1104 - accuracy: 0.9612

- val_loss: 0.1185 - val_accuracy: 0.9688

Epoch 15/50

54/54 [==============================] - 170s 3s/step - loss: 0.1067 - accuracy: 0.9578

- val_loss: 0.1632 - val_accuracy: 0.9323

Epoch 16/50

54/54 [==============================] - 169s 3s/step - loss: 0.0905 - accuracy: 0.9664

- val_loss: 0.1188 - val_accuracy: 0.9531

Epoch 17/50

54/54 [==============================] - 170s 3s/step - loss: 0.0890 - accuracy: 0.9711

- val_loss: 0.1260 - val_accuracy: 0.9688
```

```
Epoch 18/50

54/54 [illegible]
- val_loss: 0.1284 - val_accuracy: 0.9531
Epoch 19/50

54/54 [illegible] - 170s 3s/step - loss: 0.1134 - accuracy: 0.9543
- val_loss: 0.1501 - val_accuracy: 0.9531
Epoch 20/50

54/54 [illegible] - 160s 3s/step - loss: 0.0802 - accuracy: 0.9699
- val_loss: 0.1618 - val_accuracy: 0.9375
Epoch 21/50

54/54 [illegible] - 170s 3s/step - loss: 0.0710 - accuracy: 0.9757
- val_loss: 0.1723 - val_accuracy: 0.9531
Epoch 22/50

54/54 [==============================] - 170s 3s/step - loss: 0.1008 - accuracy: 0.9635
- val_loss: 0.1445 - val_accuracy: 0.9531

Epoch 23/50

54/54 [==============================] - 170s 3s/step - loss: 0.0740 - accuracy: 0.9757

- val_loss: 0.1340 - val_accuracy: 0.9427

Epoch 24/50

54/54 [==============================] - 170s 3s/step - loss: 0.0840 - accuracy: 0.9699

- val_loss: 0.1254 - val_accuracy: 0.9531

Epoch 25/50

54/54 [==============================] - 170s 3s/step - loss: 0.0936 - accuracy: 0.9676

- val_loss: 0.1469 - val_accuracy: 0.9479

Epoch 26/50

54/54 [==============================] - 170s 3s/step - loss: 0.0782 - accuracy: 0.9740

- val_loss: 0.1562 - val_accuracy: 0.9531

Epoch 27/50

54/54 [==============================] - 170s 3s/step - loss: 0.1178 - accuracy: 0.9595

- val_loss: 0.1783 - val_accuracy: 0.9271

Epoch 28/50

54/54 [==============================] - 170s 3s/step - loss: 0.0673 - accuracy: 0.9740

- val_loss: 0.1876 - val_accuracy: 0.9427

Epoch 29/50

54/54 [==============================] - 170s 3s/step - loss: 0.0746 - accuracy: 0.9728

- val_loss: 0.1546 - val_accuracy: 0.9479

Epoch 30/50

54/54 [==============================] - 170s 3s/step - loss: 0.0746 - accuracy: 0.9734

- val_loss: 0.1645 - val_accuracy: 0.9271
```

```
Epoch 31/50

54/54 [==============================] - 170s 3s/step - loss: 0.0907 - accuracy: 0.9635

- val_loss: 0.2603 - val_accuracy: 0.9219

Epoch 32/50
54/54 [==============================] - 170s 3s/step - loss: 0.0993 - accuracy: 0.9664

- val_loss: 0.2155 - val_accuracy: 0.9219
54/54 [==============================] - 170s 3s/step - loss: 0.0832 - accuracy: 0.9734
- val_loss: 0.1490 - val_accuracy: 0.9375
Epoch 34/50

54/54 [==============================] - 170s 3s/step - loss: 0.077[illegible] - accuracy: 0.97[illegible]
- val_loss: 0.1687 - val_accuracy: 0.9375
Epoch 35/50

54/54 [==============================] - 171s 3s/step - loss: 0.0825 - accuracy: 0.9740
- val_loss: 0.1564 - val_accuracy: 0.9479
Epoch 36/50
```

```
54/54 [==============================] - 170s 3s/step - loss: 0.0788 - accuracy: 0.9705
- val_loss: 0.1568 - val_accuracy: 0.9427

Epoch 37/50

54/54 [==============================] - 170s 3s/step - loss: 0.0587 - accuracy: 0.9786

- val_loss: 0.2038 - val_accuracy: 0.9375

Epoch 38/50

54/54 [==============================] - 170s 3s/step - loss: 0.0851 - accuracy: 0.9711

- val_loss: 0.2306 - val_accuracy: 0.9167

Epoch 39/50

54/54 [==============================] - 170s 3s/step - loss: 0.0885 - accuracy: 0.9647

- val_loss: 0.1429 - val_accuracy: 0.9531

Epoch 40/50

54/54 [==============================] - 171s 3s/step - loss: 0.0590 - accuracy: 0.9763

- val_loss: 0.1370 - val_accuracy: 0.9479

Epoch 41/50

54/54 [==============================] - 171s 3s/step - loss: 0.0746 - accuracy: 0.9711

- val_loss: 0.1605 - val_accuracy: 0.9479

Epoch 42/50

54/54 [==============================] - 171s 3s/step - loss: 0.0602 - accuracy: 0.9792

- val_loss: 0.1035 - val_accuracy: 0.9740

Epoch 43/50

54/54 [==============================] - 171s 3s/step - loss: 0.0743 - accuracy: 0.9740

- val_loss: 0.1533 - val_accuracy: 0.9479

Epoch 44/50

54/54 [==============================] - 170s 3s/step - loss: 0.0781 - accuracy: 0.9647

- val_loss: 0.1457 - val_accuracy: 0.9375

Epoch 45/50

54/54 [==============================] - 170s 3s/step - loss: 0.0718 - accuracy: 0.9711

- val_loss: 0.1049 - val_accuracy: 0.9635

Epoch 46/50

54/54 [==============================] - 170s 3s/step - loss: 0.0591 - accuracy: 0.9792

[illegible]
Epoch 47/50
54/54 [==============================] - 170s 3s/step - loss: 0.0711 - accuracy: 0.9699

[illegible]
Epoch 48/50
```

```
54/54 [==============================] - 170s 3s/step - loss: 0.0551 - accuracy: 0.9792

Epoch 49/50
54/54 [==============================] - 171s 3s/step - loss: 0.0531 - accuracy: 0.9809

Epoch 50/50
54/54 [==============================] - 172s 3s/step - loss: 0.0586 - accuracy: 0.9797
```

In [61]:

```
scores = model_inceptionV3 .evaluate(test_ds)
```

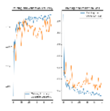
```
8/8 [==============================] - 22s 3s/step - loss: 0.0356 - accuracy: 0.9883
```

In [62]:

```
acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

loss = history.history['loss']

val_loss = history.history['val_loss']
```

In [63]:

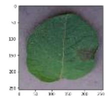
```
plt.figure(figsize = (8,8))
plt.subplot(1,2,1)

plt.plot(range(EPOCHS), acc, label = 'Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label = 'Validation Accuracy')
plt.legend(loc = 'lower right')

plt.title('Training and Validation Accuracy')


plt.subplot(1,2,2)

plt.plot(range(EPOCHS), loss, label = 'Training Loss')
plt.plot(range(EPOCHS), val_loss, label = 'Validation Loss')
plt.legend(loc = 'upper right')

plt.title('Training and Validation Loss')

plt.show()
```



In [64]:

```
for images_batch, labels_batch in test_ds.take(1):
    first_image = image_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()
    print("first image to predict")

    plt.imshow(first_image)

    print("Actual label:",class_names[first_label])


    batch_prediction = model.predict(images_batch)

    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict

Actual label: Potato___Early_blight

1/1 [==============================] - 1s 614ms/step
predicted label: Potato___Early_blight
```



```
def
predi
```

In [65]:

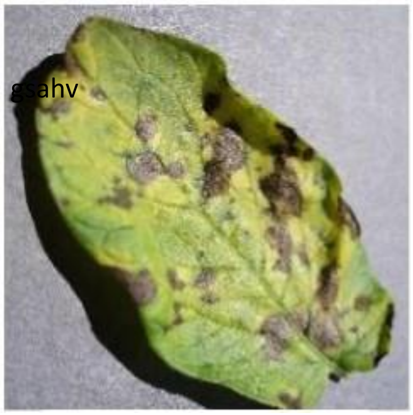
```
ct(model_inceptionV3, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model_inceptionV3.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

In [66]:

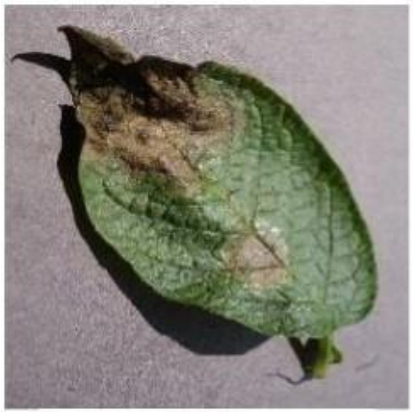
```
plt.figure(figsize=(15, 15))
for images , labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i+1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model_inceptionV3, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual:{actual_class},\n Predicted: {predicted_class}, \n
Confidence:{confidence}%")
```

```
        plt.axis("off")
```

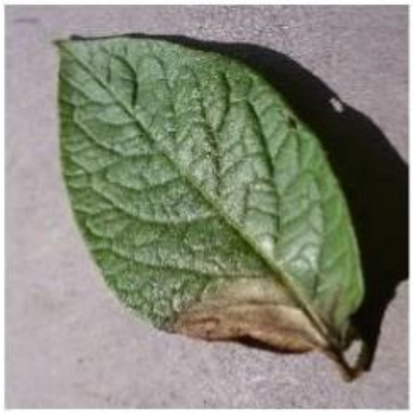
```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 110ms/step
```



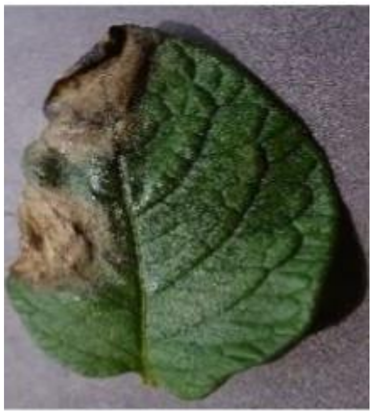
Actual: Potato_ Early _blight, Predicted: Potato_Earty_blight. Confidence:100.0%



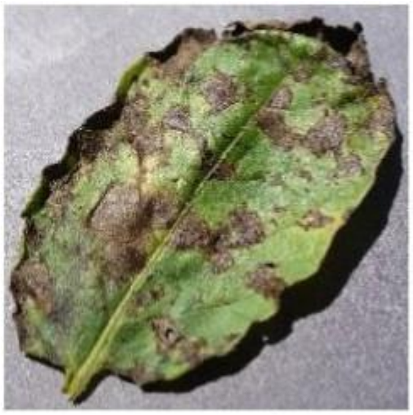
Actual:Potato_Late_blight, Predicted: Potato_ Late_blight,Confidence:100%



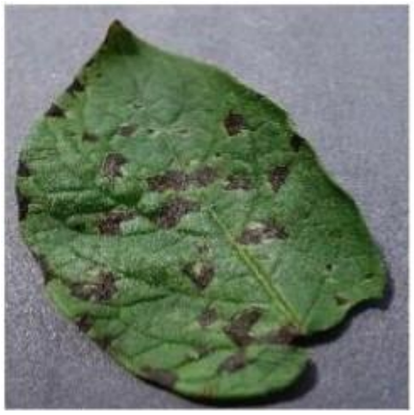
Actual:Potato_Late_blight, Predicted: Potato_Late_blight, Confidence:99.21%



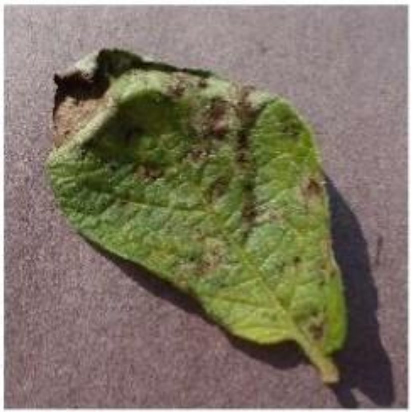
Actual: Potato_Late_ blight, Predicted: Potato_Late_blight Confidence :100%



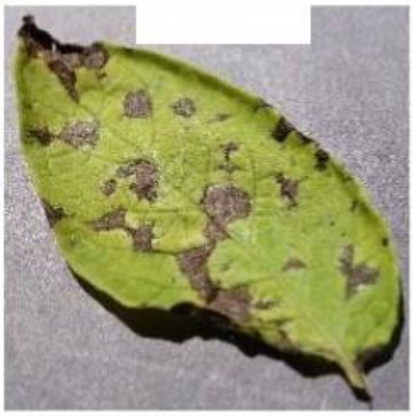
Actual:Potato_Earty_blight. Predicted: Potato_Earty_blight,Confidence:100%



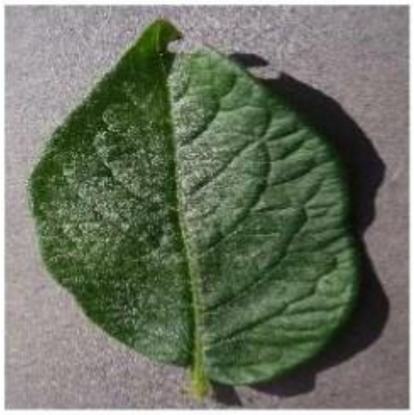
Actual:Potato_Early_blight. Predicted: Potato_Earty_blight.Confidence 100%



Actual:Potato_Late_blight. Predicted: Potato_Late_blight. Confidence:99.99%



Actual:Potato_Earty_blight. Predicted: Potato_Earty_blight. Confidence :100%



Actual:Potato_healthy.Predicted: Potato_healthy, Confidence:100.0%

Detection and Classification of Plant Diseases using Neural Network Approach

Mohammad Burhan Uddin
Electrical and Computer Engineering
North-South University
Dhaka, Bangladesh
mohammad.uddin18@northsouth.edu

Md.Rakibul Islam Rakib
Electrical and Computer Engineering
North-South University
Dhaka, Bangladesh
rakibul.rakib@northsouth.edu

Rifat Ahmed Hassan
Lecturer
Electrical and Computer Engineering
North South University
Dhaka, Bangladesh

Abstract -Bangladesh's agriculture sector is an important one that greatly affects the nation's wealth. Shrubberies have advanced on a vital get-up-and-go basis and a potentially fatal component, solving the conundrum of determining the whole heating issue. However, plants are currently impacted by several factors. This illness detection challenge is well-solved by deep learning techniques. Convolutional neural networks are very good at complex picture categorization and object identification tasks. The main goal of the projected effort is to find a solution for the challenge of diagnosing plant illnesses using the least amount of computational resources and the most easy way possible, producing better results than standard representations. The PlantVillage Dataset from the open-source Kaggle platform is used in this work. We categorize Potato Leaf Disease based on the many types of leaf disease photos included in the dataset. Inception V3 can execute 98% of the classification correctly, CNN can obtain an average accuracy of 99.61%, and VGG16 can provide 100% accuracy when it comes to leaf disease classification. To enable a behavior surveillance procedure for the identified illness, a graphical user interface (GUI) for the system is also being finalized. From photos captured in the outdoors, CNN is able to identify plant illnesses and extract important traits. The deep CNN model is promising and can have a considerable influence on the efficiency of disease identification. It also has potential for disease detection in real-time agricultural systems, as indicated by the accuracy results in disease identification.

Index Terms—Plant disease detection, Plant disease Classification, Potato Plant Disease, Image Processing, CNN, VGG 16, Inception V3, Color thresholding, Deep learning.

I. Introduction

The amount and productivity of agricultural land determine the economic development of a nation. For most people, agriculture provides their main source of income. Depending on the fertility of the soil and the availability of resources, farmers plant a range of crops. Variations in meteorological parameters including rainfall, temperature, and soil fertility can lead to crop infections by fungus, bacteria, and viruses [1]. They organize the plants to avoid sicknesses and enhance the production and quality of their products by using the proper pesticides and weedkillers. Plant disease identification and research are conducted by visual study of the plants' patterns. Early diagnosis of plant diseases is beneficial since they may be managed. In many countries, farmers lack the resources or knowledge to communicate with experts. One such technique for detection is the visual inspection of leaf patterns by professionals. But it requires a large number of specialized personnel. In this case, an automated plant infection or disease monitoring system will be useful. By comparing the stored plant disease symptoms with the leaves of the plants in the agricultural farm area, automation will be less expensive. The three plant diseases included in this category are bacterial blight, anthracnose, and Cercospora leaf spot. On the leaf, anthracnose causes brown or tan spots with irregular shapes to appear. These areas will be adjacent to the leaf veins. If the infection is severe, leaf drop will happen. Cercospora spray spot plants have tiny brown dots with a magenta border. It eliminates elasticities and consumes a grey focus. A hole is left when the leaf material turns friable and reedy and falls out. Proteomics Blight disease can infect a plant's trunk, branches, shoots, buds, flowers, leaves, and fruit. A tiny, pale green dot appears on the leaf and grows throughout it. The lesion area eventually turns into a dead, dry patch [2]. To diagnose the illness or impurity using image processing, a leaf trial is required. Plant disease classification involves a number of interconnected processes, including image capture, pre-processing, dissection, feature mining, and organization. In this study, a plant leaf detection system is developed and constructed to process leaf image detection and categorize the leaf for disease diagnosis using image processing. A few procedures are discussed and looked at in order to carry out the plan in picture segmentation and classification in order to determine which approach is most suited and feasible for improving accuracy in identifying the sickness on the plant leaf. Utilizing the CNN, VGG 16, and Inception V3 classifiers, image segmentation is studied.

II. Literature Review

Using CNN and deep learning structures, the section assigns thematic trends to agricultural submissions. Previous methods for image processing, machine learning, and deep learning were abandoned in order to address various plant pests. Most of these systems typically work like this: A digital camera is used to capture the first cardinal photographs. After that, the photographs are prepared for the following stages using image distribution techniques including image heightening, segmentation, shade intergalactic restoration, and filtering. The basic components of the picture are then unaffected and castoff as classifier input [3]. Thus, the total classification accuracy is determined by the feature mining techniques used in image processing. Nevertheless, subsequent research has shown that the advanced recital may be achieved by systems trained on generic data.

In recent years, image processing has been the focus of research on the identification and detection of plant leaf disease. These studies [4] [5] demonstrate the authors' recommendation of the K-means method for image subdivision. The approach to divide the Dataset into k distinct, non-overlapping clusters was dropped in order to distinguish between the critical component and the nurturing, as well as the contaminated region on the foliage, with each statistic point corresponding to a single assembly. For subdivisions, the author advises use color-based thresholding. In order to conceal forbidden regions, verges were installed on the various networks, and pictures were updated to other color spaces, such as LAB and HSV.

[6] Here, machine learning techniques like Decision Trees, Support Vector Machines (SVMs), and artificial neural networks have been abandoned. Antiquated techniques for classifying images have relied on manually designed regions like SIFT [7], HOG [8], and SURF [9], monitored by employing information on the processes of learning in these characteristic areas. Consequently, the recitation of all these methods heavily relies on the fundamental predefined frameworks. But according to a recent machine learning study, learnt representations are more organized and helpful. The primary benefit of representation learning is in its ability to automatically sift through large amounts of image data and identify features that might be used to precisely classify images. The author [10] used CNN methods such as AlexNet and Inception Net to identify 26 distinct plant bugs.

[11] The author achieved excellent levels of classification precision by identifying a variety of plant illnesses using different CNN approaches. They also used real photos to construct the CNN architecture as part of their methodology. In [12] Unlike plant bugs, the author used a deep learning structure to identify 13. The deep learning framework was authorized for use in CNN training. The author extensively examined the shortcomings of several deep learning techniques in the field of agriculture. The scientists intended to use a nine-layer CNN model to identify plant pests. They employed data-augmentation techniques to increase the quantity of the data and the PlantVillage dataset for their research. The authors [13] projected a nine-layer Convolution Neural Network model to categorize plant diseases. After removing the PlantVillage dataset and using data-augmentation techniques to boost the amount of data for testing scenarios, they assessed the presentation. The authors' accuracy was greater than that of a traditional machine-learning approach. In [14] With enhanced hyperparameters such as a max epoch, minibatch size, and bias learning rate, pretrained AlexNet and GoogleNet were still used to distinguish three different soybean bugs from pictures of healthy

leaves. Six of the pre-trained networks that were used were VGG16, AlexNet, GoogleNet, ResNet, and DenseNet.

[15] Here, the author identifies three different kinds of bugs and two different forms of pest damage in cassava shrubberies using a transfer-learning approach. The scientists subsequently carried out more research on the identification of cassava plant diseases, obtaining an accuracy of 80.6% by employing a CNN model based on a smartphone.

III. Methodology

A sizable set of plant leaf photos is required to classify Potato plant disease. The images are from a collection called PlantVillage. In this field, the approach taken is carefully considered.

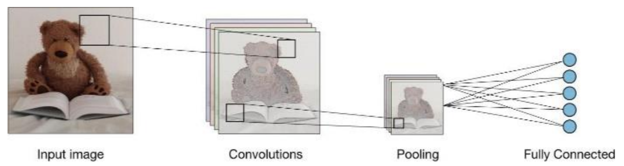


Fig. 1. Convolutional layer, pooling layer and fully connected layer.

In Fig. 1, we demonstrate a convolutional network and how it works. Here we can see how the convolutional layer, pooling layer and fully connected layer are connected.

A. Dataset

For the purposes of training and testing classification, a suitable and large dataset is necessary. The expert-it's dataset, known as the PlantVillage Dataset [15], was sourced from the open-access Kaggle platform. It includes several images and tags of plant leaves. It is a dataset of 54,305 photographs of healthy and diseased leaves that were shot in various settings. Images of both healthy and diseased leaves from 14 other plant types are contained in 38 distinct classes within the dataset. In order to detect potato diseases, we have chosen 2,152 photos of three different types of potato leaves: healthy leaves, late blight, and early blight. The samples in the Dataset are categorized in Table I. A few instances from the database are shown in Fig. 2.

B. Dataset Preprocessing

In order to get the Dataset ready for training, different resolution pictures are shrunk to 128 by 128 pixels. The neural network may be biased by the training photos' varying lighting and backgrounds because they were taken in an abandoned setting. To test this, the segmented database and greyscale versions of the tests were also used. Next, we divided the dataset into three categories: training (80%), validation (10%), and testing (10%).

TABLE I

Dataset Use for the Classification

No.	Type	Number
1	Healthy Leaf	152
2	Early Blight Leaf	1,000
3	Late Blight Leaf	1,000

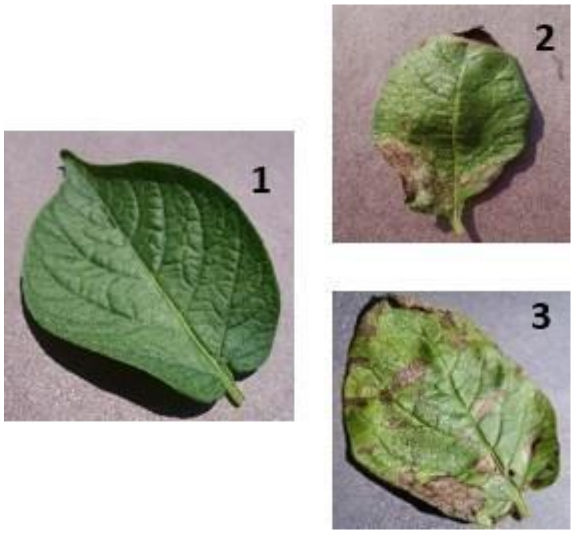
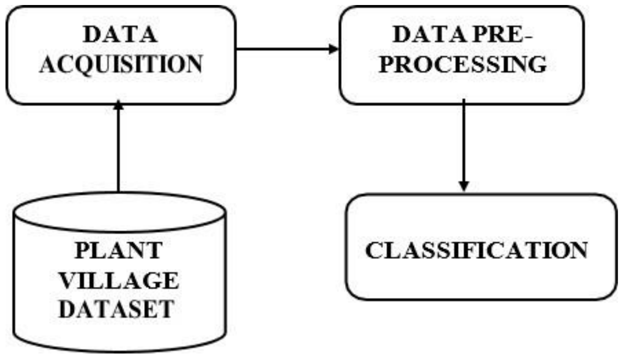


Fig. 2. Sample images from the Dataset, (1) Healthy Potato Leaf, (2) Early blight Potato Leaf, and (3) Late blight Potato leaf.

.

C. Segmentation

The primary goal of image segmentation is to extract the region of interest (ROI) of the tomato leaf from the image. In this case, the algorithm will be evaluating the plant leaf in the foreground, while the



grey backdrop of the Dataset contains no information. Consequently, the backdrop of the image must be removed using a mask so that only the leaf-containing pixels and a black background remain. After masking the surrounding area of the leaf image, the healthy portion will be further veiled in order to calculate the proportion of the leaf affected by the disease

D. Design

The proposed system architecture, as seen in Fig. 3, comprises gathering information from a large dataset, processing several convolution layers, and classifying plant diseases to ascertain whether the plant image is healthy or diseased.

Fig. 3. Working sequences of the proposed plant disease detection system.

E. Proposed CNN Model

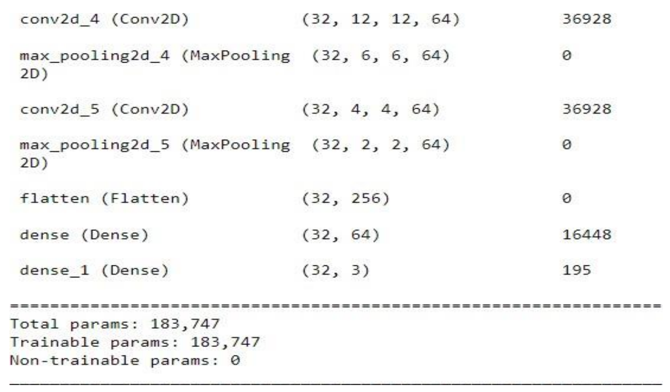
Convolutional neural networks are a kind of deep neural networks. A CNN is better suited for processing 2D data, such photos, by combining input data with well-read features prior to employing 2D convolutional layers. CNNs are capable of classifying pictures without the need for human feature extraction or removal. Features are directly extracted from images by the CNN model. In contrast to the extracted features, which are well-read and not pre-trained, the network is trained on several image groups. The Convolutional Neural Network (CNN) model processes images across several layers. The Convo Layer, Fully, Soft-max Layer, Connected Layer, Input Layer, Output Layer, and Pooling Layer are a few of them. [16] Depending on the issue at hand, several CNN architectures are used. The suggested model uses three convolutional layers, with a max-pooling layer in between. MPL, the last layer, has full connectivity. The ReLu activation function is applied to the output of each fully connected and convolutional layer.

TABLE II

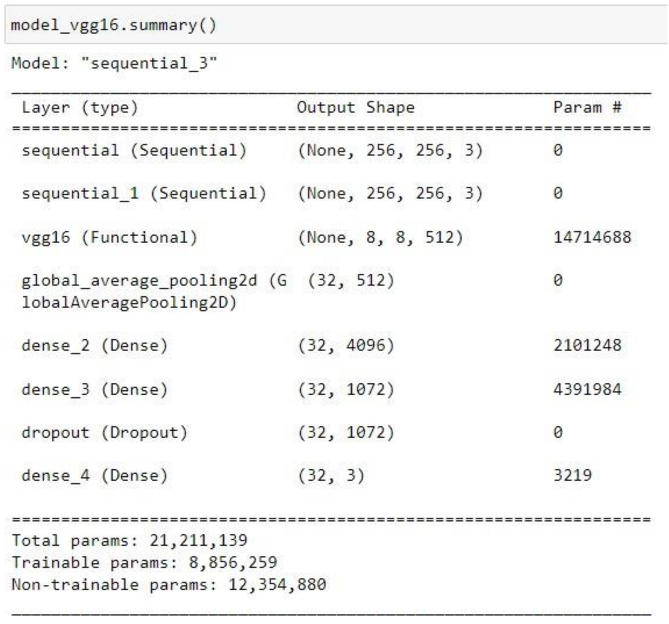
ARCHITECTURE OF THE PROPOSED CNN MODEL

Layer	Type	Filter Size	Stride	Output Size
L1	Conv	3 × 3	1	128 × 128 × 32
	Pool	2 × 2	2	64 × 64 × 32
L2	Conv	4 × 4	1	61 × 61 × 32
	Pool	2 × 2	2	64 × 64 × 32
L3	Conv	1 × 1	1	30 × 30 × 28
	Pool	2 × 2	2	15 × 15 × 128

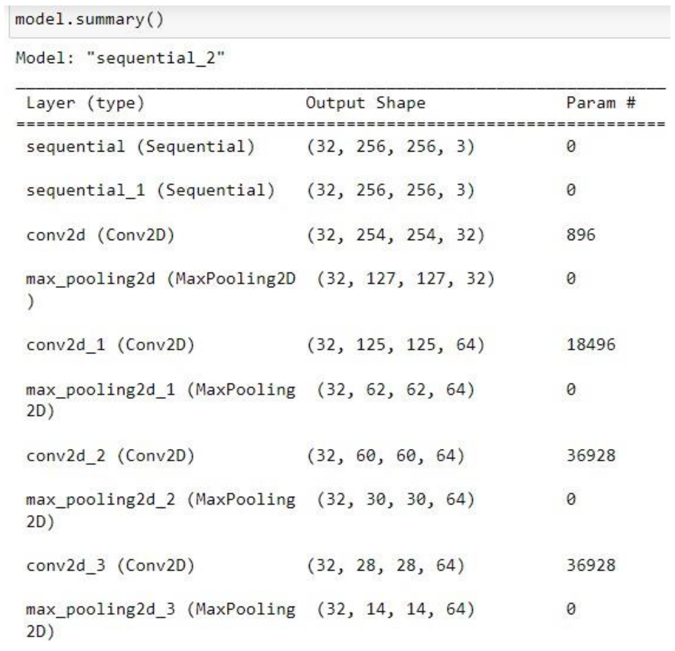
For three channels and fifty epochs with a 128 by 128 image size, we use a batch size of 32 in this illustration. The input image is filtered using 32 3 x 3 kernels in the first convolutional layer. Later, the second convolution layer—which has 6464 kernels—is fed the output obtained from max pooling. Finally, there is a final convolutional layer with 256 size 11 kernels and 512 neurons that are completely coupled. After this layer's output is received, the softmax function generates a probability distribution for each of the four output classes. [17]. The proposed model (CNN) is presented in Table II. Training the model is done by adaptive moment training.



```
 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0
 2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0
 2D)

 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448

 dense_1 (Dense)             (32, 3)                   195

=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```



```
model_vgg16.summary()

Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0

 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 global_average_pooling2d (G  (32, 512)                0
 lobalAveragePooling2D)

 dense_2 (Dense)             (32, 4096)                2101248

 dense_3 (Dense)             (32, 1072)                4391984

 dropout (Dropout)           (32, 1072)                0

 dense_4 (Dense)             (32, 3)                   3219

=================================================================
Total params: 21,211,139
Trainable params: 8,856,259
Non-trainable params: 12,354,880
_________________________________________________________________
```



```
model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0
 )

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0
 2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0
 2D)
```

Fig. 4. Various layers, output shape and number of parameters of the CNN Model.

As shown in Fig. 4, estimation (Adam) was performed using a batch size of 32 for 50 epochs.

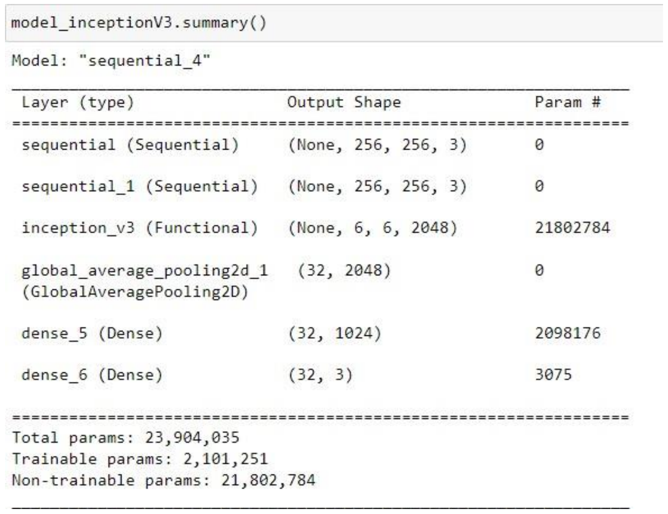
F. VGG 16 Model

VGG 16 is a CNN model designed for large-scale picture analysis. Completing two tasks is necessary for the best identification of plant diseases. Object localization, which finds objects from various classes inside an image, is the initial phase. Image categorization, the second stage, entails grouping photos into various categories. The CNN model consists of seven tiers. Every layer processes information in a unique way. The following are the seven tiers: Convolutional Layer, Input Layer, Output Layer, Pooling Layer, Fully Convolutional Layer, Fully Connected Layer, and Soft-max Layer [18]. An overview of the VGG16 model in practice is presented in Fig. 5.

Fig. 5. Various layers, output shape and number of parameters of the VGG16 model.

G. Inception V3

Compared to its predecessors, the 2015-released Inception v3 model has 42 layers and a lower error rate. When a model had several deep layers of convolutions, the data were over-fit. To avoid this, the concept V3 model makes use of the notion of using many filters on the same level, each with a different size. Consequently, our model is broader than deeper since the inception models have parallel layers instead of deep layers. The Inception model consists of many Inception modules. [19]. Figure 6 presents the model summary.



```
model_inceptionV3.summary()

Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0

 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 global_average_pooling2d_1   (32, 2048)               0
 (GlobalAveragePooling2D)

 dense_5 (Dense)             (32, 1024)                2098176

 dense_6 (Dense)             (32, 3)                   3075

=================================================================
Total params: 23,904,035
Trainable params: 2,101,251
Non-trainable params: 21,802,784
_________________________________________________________________
```

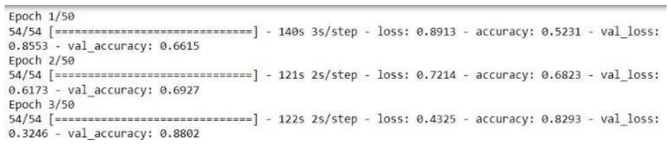
Fig. 6. Various layers, output shape and number of parameters of the Inception V3 model

IV. Results and Discussion

Ten percent is utilized for testing, ten percent for validation, and eighty percent of the dataset is used for training. Numerous models with different topologies and learning rates are tested. The learning parameter, kernel size, and filter size were among the network parameters that were chosen via trial and error. Research has shown that training is more effective when the ReLu activation function is used.

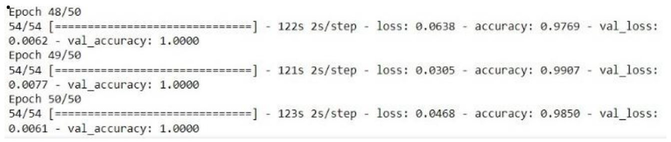
.

A. CNN Model



```
Epoch 1/50
54/54 [==============================] - 140s 3s/step - loss: 0.8913 - accuracy: 0.5231 - val_loss:
0.8553 - val_accuracy: 0.6615
Epoch 2/50
54/54 [==============================] - 121s 2s/step - loss: 0.7214 - accuracy: 0.6823 - val_loss:
0.6173 - val_accuracy: 0.6927
Epoch 3/50
54/54 [==============================] - 122s 2s/step - loss: 0.4325 - accuracy: 0.8293 - val_loss:
0.3246 - val_accuracy: 0.8802
```

.



```
Epoch 48/50
54/54 [==============================] - 122s 2s/step - loss: 0.0638 - accuracy: 0.9769 - val_loss:
0.0062 - val_accuracy: 1.0000
Epoch 49/50
54/54 [==============================] - 121s 2s/step - loss: 0.0305 - accuracy: 0.9907 - val_loss:
0.0077 - val_accuracy: 1.0000
Epoch 50/50
54/54 [==============================] - 123s 2s/step - loss: 0.0468 - accuracy: 0.9850 - val_loss:
0.0061 - val_accuracy: 1.0000
```

Fig. 7. Accuracy results of the CNN model.

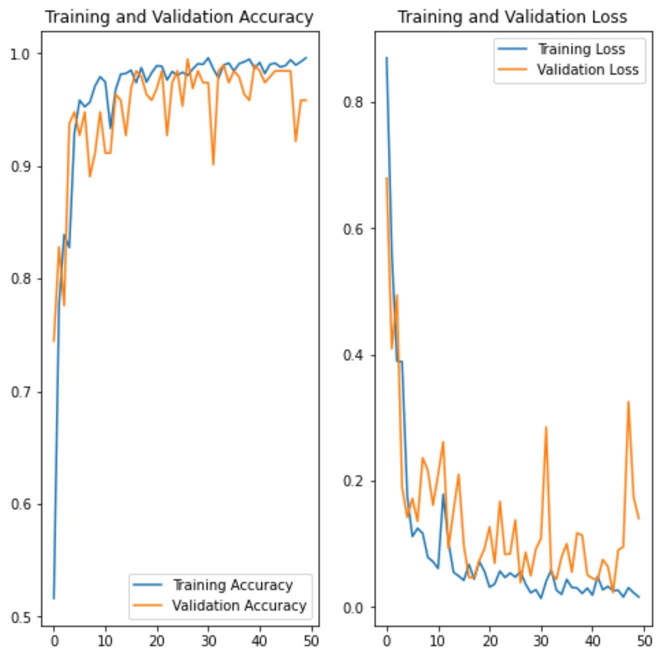


Fig. 8. Training vs validation accuracy and loss of the CNN model.

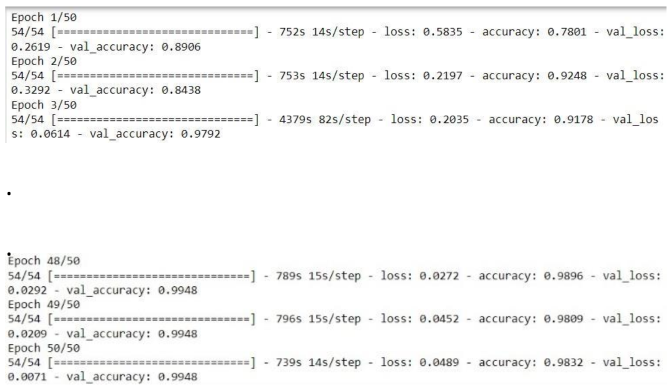
The classification accuracy is 66.15% in the first epoch and rises to 99.61% after 50 epochs, as Fig. 7 illustrates. The model's three convolution layers are followed by a max-pooling layer, which improves classification accuracy. The ReLu activation function is applied to each

layer. The model's training accuracy vs validation graphs are shown in Fig. 8.

The over-fitting of the model is demonstrated by the graphs. Over-fitting happens when the model fits the training set too closely. Because of this, the model finds it difficult to generalize to new examples that do not belong in the training set. A number of strategies have been devised to avoid over-fitting. Weight penalties like regularization, dropout, and data augmentation are used in these strategies. Trials were carried out to ascertain the effect of every strategy on the model's functionality. The first experiment we conducted included improving the training data by rotating, flipping, and rescaling the images because the dataset was too small in comparison to the total number of trainable parameters in the model. Only the training data are used for data augmentation. By itself, data augmentation significantly lessens over-fitting.

It also raises the accuracy of validation. Dropping out and regularization are also employed. In terms of performance, both models marginally outperformed the model. Consequently, we add a dropout layer with a probability of 0.5 when the MLP yields high classification accuracy.

VGG 16 Model



```
Epoch 1/50
54/54 [==============================] - 752s 14s/step - loss: 0.5835 - accuracy: 0.7801 - val_loss:
0.2619 - val_accuracy: 0.8906
Epoch 2/50
54/54 [==============================] - 753s 14s/step - loss: 0.2197 - accuracy: 0.9248 - val_loss:
0.3292 - val_accuracy: 0.8438
Epoch 3/50
54/54 [==============================] - 4379s 82s/step - loss: 0.2035 - accuracy: 0.9178 - val_los
s: 0.0614 - val_accuracy: 0.9792

.

.
Epoch 48/50
54/54 [==============================] - 789s 15s/step - loss: 0.0272 - accuracy: 0.9896 - val_loss:
0.0292 - val_accuracy: 0.9948
Epoch 49/50
54/54 [==============================] - 796s 15s/step - loss: 0.0452 - accuracy: 0.9809 - val_loss:
0.0209 - val_accuracy: 0.9948
Epoch 50/50
54/54 [==============================] - 739s 14s/step - loss: 0.0489 - accuracy: 0.9832 - val_loss:
0.0071 - val_accuracy: 0.9948
```

Fig. 9. Accuracy results of the VGG 16 model.

Fig. 9 shows the accuracy and loss results for the VGG16 model. In the first epoch, it acquired 78% accuracy and performed well in the 50^{th} of epoch with 100% accuracy. Fig.10 demonstrate the Training VS Validation Accuracy and loss curve.

B.Inception V3 Model

Fig. 11 shows the accuracy and loss results for the Inception V3 model. In the first epoch, it acquires 77% accuracy and performs well in the 50^{th} of epoch with 97% accuracy. Fig. 12 demonstrate the Training VS Validation Accuracy and loss curve.

TABLE III Accuracy of each model

	Training Accuracy	Test Accuracy
CNN	97.50%	99.61%
VGG16	98.32%	100%
INCEPTION V3	97.97%	98.83%

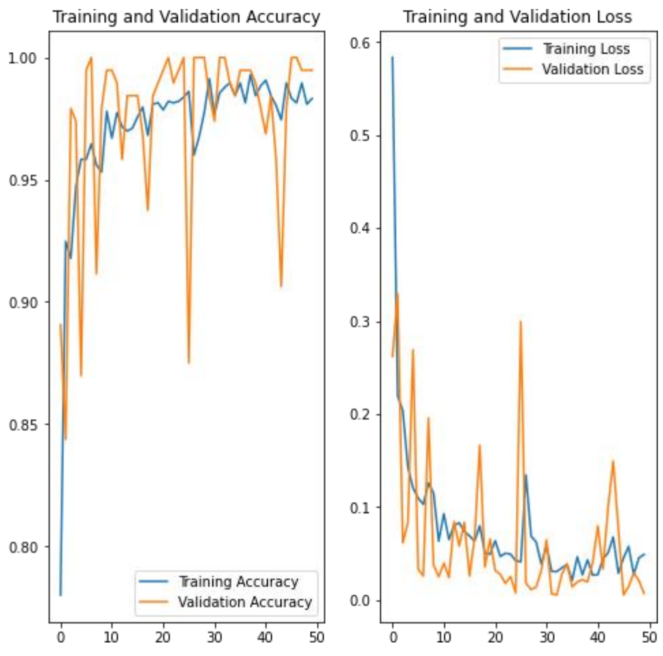
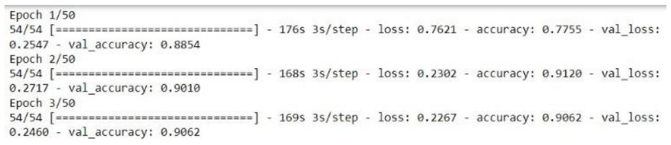


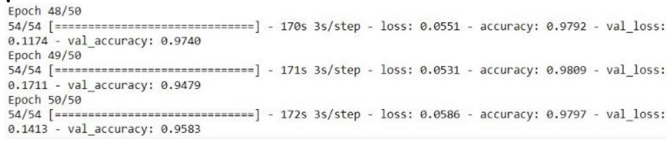
Fig. 10. Training vs validation accuracy and loss of the VGG 16 model.



```
Epoch 1/50
54/54 [==============================] - 176s 3s/step - loss: 0.7621 - accuracy: 0.7755 - val_loss:
0.2547 - val_accuracy: 0.8854
Epoch 2/50
54/54 [==============================] - 168s 3s/step - loss: 0.2302 - accuracy: 0.9120 - val_loss:
0.2717 - val_accuracy: 0.9010
Epoch 3/50
54/54 [==============================] - 169s 3s/step - loss: 0.2267 - accuracy: 0.9062 - val_loss:
0.2460 - val_accuracy: 0.9062
```

.

.



```
Epoch 48/50
54/54 [==============================] - 170s 3s/step - loss: 0.0551 - accuracy: 0.9792 - val_loss:
0.1174 - val_accuracy: 0.9740
Epoch 49/50
54/54 [==============================] - 171s 3s/step - loss: 0.0531 - accuracy: 0.9809 - val_loss:
0.1711 - val_accuracy: 0.9479
Epoch 50/50
54/54 [==============================] - 172s 3s/step - loss: 0.0586 - accuracy: 0.9797 - val_loss:
0.1413 - val_accuracy: 0.9583
```

Fig. 11. Accuracy results of the Inception V3 model.

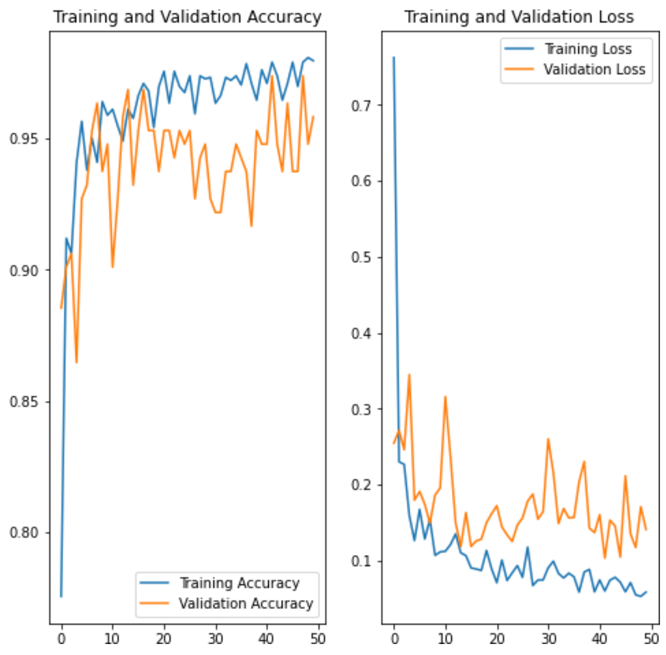


Fig. 12. Training vs validation accuracy and loss of the Inception V3 model.

Table III shows All models with their average accuracy and chooses the best fit model for this Dataset. Fig. 13 demonstrate the confidence of the best-fit model VGG16.

TABLE IV Comparison Table

Reference	Applied Model	Dataset	Accuracy
[10]	AlexNet and GoogleNet	PlantVillage Dataset	99.27% in AlexNet
[12]	Finetuned CNN	Stanford Background	96.3%
[13]	Inception V3 Based	Cassava Leaf Disease	93%
[15]	Nine-layer Deep CNN	Plant Leaf Disease	96.46%
This Work	**VGG 16**	**PlantVillage Dataset**	**100%**

Table IV compares different models applied in different Dataset chosen from several research papers. We compare it with their accuracy. And Finlay shows our proposed model in the PlantVillage Data-set with 100% accuracy.

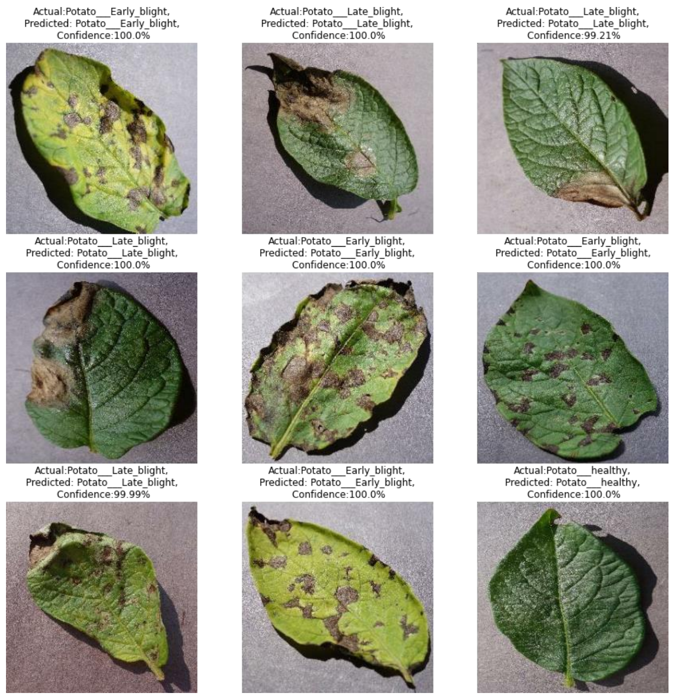

Actual:Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence:100.0%
Actual:Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence:100.0%
Actual:Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence:99.21%
Actual:Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence:100.0%
Actual:Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence:100.0%
Actual:Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence:100.0%
Actual:Potato___Late_blight,
Predicted: Potato___Late_blight,
Confidence:99.99%
Actual:Potato___Early_blight,
Predicted: Potato___Early_blight,
Confidence:100.0%
Actual:Potato___healthy,
Predicted: Potato___healthy,
Confidence:100.0%

Fig. 13. Predicted class and the confidence of using the VGG16 model.

V. CONCLUSION

This training uses CNN, VGG16, and Inception V3 to identify and categorize Potato Plant bugs. Using images gathered from the natural world, the system was trained to attain a 100% classification accuracy. This demonstrates how VGG 16 can gather vital data from the natural world that is required to identify plant diseases.

According to our friends, this was the first time that pictures shot in a natural setting have been used to get meaningful results. Experiments also show that when the dataset is limited, adding additional data to the training set improves network learning. Regularization and dropout were also shown to have an influence on overcoming over-fitting. A First API server based on the concept may likewise be developed. After that, a Google Cloud website and deployment model may be created. At last, we are able to create an application that will support farming.

REFERENCES

[20] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, "Deep learning for image-based cassava disease detection," *Frontiers in Plant Science*, vol. 8, pp. 1–7, 2017.

[21] P. K. Sethy, N. K. Barpanda, A. K. Rath, and S. K. Behera, "Deep feature-based rice leaf disease identification using support vector machine," *Computers and Electronics in Agriculture*, vol. 175, p. 105527, 2020.

[22] H. M. Alexander, K. E. Mauck, A. E. Whitfield, K. A. Garrett, and C. M.

Malmstrom, "Plant-virus interactions and the agro-ecological interface,"

European Journal of plant pathology, vol. 138, pp. 529–547, 2014.

[23] R. M. Prakash, G. Saraswathy, G. Ramalakshmi, K. Mangaleswari, and T. Kaviya, "Detection of leaf diseases and classification using digital image processing," in *International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1–

4, 2017.

[24] A. Devaraj, K. Rathan, S. Jaahnavi, and K. Indira, "Identification of plant disease using image processing technique," in *International Conference on Communication and Signal Processing (ICCSP)*, pp. 0749–0753,

2019.

[25] S. P. Mohanty, D. P. Hughes, and M. Salathe', "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, p. 1419, 2016.

[26] D. G. Lowe, "Distinctive image features from scale-invariant keypoints,"

International Journal of Computer Vision, vol. 60, pp. 91–110, 2004.

[27] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 886–893, 2005.

[28] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, pp. 346–359, 2008.

[29] S. P. Mohanty, D. P. Hughes, and M. Salathe', "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, p. 1419, 2016.

[30] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and electronics in agriculture*, vol. 145, pp. 311–

318, 2018.

[31] 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 and neuroscience*, vol. 2016,

2016.

[32] G. Geetharamani and A. Pandian, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers & Electrical Engineering*, vol. 76, pp. 323–338, 2019.

[33] S. B. Jadhav, V. R. Udupi, and S. B. Patil, "Identification of plant diseases using convolutional neural networks," *International Journal of Information Technology*, vol. 13, no. 6, pp. 2461–2470, 2021.

[34] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, "Deep learning for image-based cassava disease detection," *Frontiers in plant science*, vol. 8, p. 1852, 2017

and D. P. Hughes, "Deep learning for image-based cassava disease detection," *Frontiers in plant science*, vol. 8, p. 1852, 2017

[35] S. Y. Yadhav, T. Senthilkumar, S. Jayanthy, and J. J. A. Kovilpil- lai, "Plant disease detection and classification using cnn model with optimized activation function," in *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pp. 564– 569, IEEE, 2020.

[36] M. Akila and P. Deepan, "Detection and classification of plant leaf diseases by using deep learning algorithm," *International Journal of Engineering Research & Technology (IJERT)*, vol. 6, pp. 1–5, 2018.

[37] R. R, India., M. S H, and India., "Plant disease detection and classifi- cation using cnn," vol. 10, Sep 2021.

[38] C. Wang, D. Chen, H. Lin, B. Liu, C. Zeng, D. Chen, and G. Zhang, "Pulmonary image classification based on inception-v3 transfer learning model," *IEEE Access*, vol. PP, pp. 1–1, 10 2019.