A Mini Project with Seminar On

Plant Disease Detection Using Deep Learning

Submitted in partial fulfillment of the requirements for the award of the

Bachelor of Technology

in

Department of Computer Science and Engineering (Data Science)

by

Chalamalla Eshwar	**20241A6709**
Yamsani Sai Bharath	**20241A6760**
Rohan Daliyet	**20241A6747**

Under the Esteemed guidance of

Dr. G. Karuna

Professor & HOD

Department of Computer Science and Engineering (Data Science)

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, Autonomous under JNTUH, Hyderabad)

Bachupally, Kukatpally, Hyderabad-500090

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

Hyderabad-500090

CERTIFICATE

This is to certify that the mini project entitled "**Plant Disease Detection using Deep Learning**" is submitted by **Chalamalla Eshwar (20241A6709), Yamsani Sai Bharath (20241A6760), and Rohan Daliyet (20241A6747)** in partial fulfillment of the award of degree in BACHELOR OF TECHNOLOGY in Computer Science and Engineering [Data Science] during the Academic year 2022-2023.

.

Internal Guide	Head of the Department
Dr. G. Karuna	**Dr. G. Karuna**
Professor & HOD	**Professor & HOD**

External Examiner

ACKNOWLEDGEMENT

There are many people who helped us, directly and indirectly, to complete our project successfully. We would like to take this opportunity to thank one and all. First, we would like to express our deep gratitude towards our internal guide **Dr. G. Karuna,** Professor & HOD**,** Department of Computer Science Engineering (Data Science), for his support in the completion of our dissertation. We wish to express our sincere thanks to **Dr. G. Karuna,** Head of the Department**,** and to our principal **Dr. J. PRAVEEN,** for providing the facilities to complete the dissertation. We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

Chalamalla Eshwar(20241A6709)

Yamsani Sai Bharath(20241A6760)

Rohan Daliyet (20241A6747)

DECLARATION

We hereby declare that the mini project titled **"PLANT DISEASE DETECTION USING DEEP LEARNING"** is the work done during the period from **17th January 2023 to 12th June 2023** and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering (Data Science) from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad). The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

Chalamalla Eshwar(20241A6709)
Yamsani Sai Bharath(20241A6760)
Rohan Daliyet (20241A6747)

ABSTRACT

Crop diseases pose a significant threat to global food security, particularly in areas where infrastructure is limited. To address this challenge, a platform for accurate identification of plant diseases is needed. "Deep Learning Plant Disease Detection" is a platform that uses deep learning techniques to identify plant diseases. The platform uses the "CNN" algorithm, widely known for its high accuracy in image classification, enabling it to accurately identify plant diseases from images . Furthermore, the platform offers recommendations for preventing and provide supplements for that disease and managing the spread of crop diseases using its user-friendly web interface.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronyms	Description
CNN	Convolutional Neural Networks
ConvNet	Convolutional Neural Networks
CPU	Central Processing Unit
GPU	Graphical Processing Unit
NN	Neural Network
RNN	Recurrent Neural Network
UML	Unified Modelling Language
MVC	Model View Controller
DRY	Don’t Repeat Yourself
RAM	Random Access Memory
GUI	Graphical User Interface
API	Application Programmable Interface
IDE	Integrated Development Environment
ResNet	Residual Network
VGG	Visual Geometry Group

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

Agriculture plays a vital role in the economic development of countries worldwide. One of the key challenges in agriculture is the detection of plant diseases, which can lead to significant crop yield losses and economic losses for farmers. Traditional methods of plant disease detection involve manual inspection of plants, which is time-consuming, labor-intensive, and prone to errors. In recent years, there has been a growing interest in using deep learning techniques to automatically detect plant diseases, which has the potential to improve the efficiency and accuracy of plant disease detection.

The objective of our project is to build a model using deep learning techniques to detect plant diseases on crops like potatoes, tomatoes, and bell pepper with better accuracy and provide a user-friendly interface for farmers. Our motivation for this project comes from the need to address the challenges facing farmers in detecting plant diseases early and accurately, which can help them prevent crop losses and increase their yield.

In this project, we adopted a deep learning approach to train our model on a large dataset of images of healthy and diseased plants. We used convolutional neural networks (CNNs), which are a type of deep learning algorithm that can learn to extract features from images and classify them based on those features. We also used data augmentation techniques to increase the size of our dataset and prevent overfitting.

Our architecture diagram consists of three main components: the farmer interface, the deep learning model, and the resulting interface. The farmer interface allows farmers to upload images of their crops, which triggers the deep learning model to predict the disease. The deep learning model consists of a pre-trained CNN that can predict the disease based on the pre-processed image. The resulting interface displays the predicted result to the farmer.

Existing approaches to plant disease detection include traditional methods like visual inspection and chemical analysis, as well as machine learning-based methods like decision trees and support vector machines. However, these approaches have several limitations, such as low accuracy, time-consuming data collection, and the need for domain-specific knowledge. Deep learning techniques have the potential to overcome these limitations by leveraging large datasets to learn the underlying patterns in plant images and make accurate predictions.

Our model's performance is evaluated using several metrics like accuracy, precision, recall, and F1-score. We compared our model's performance with other existing approaches, and our results showed that our model outperforms existing methods in terms of accuracy and speed. Our system's accuracy is high enough to detect plant diseases accurately, and our model's user-friendly interface can help farmers detect plant diseases more quickly and efficiently.

The implications of our project for the agricultural industry are significant. By providing a more accurate and efficient way of detecting plant diseases, our system can help farmers prevent crop losses, reduce the use of pesticides, and increase their yield. The benefits of our system are particularly important for small-scale farmers who may not have access to specialized equipment and resources for disease detection.

In conclusion, our project's deep learning-based approach to plant disease detection shows promising results in terms of accuracy and speed. Our user-friendly interface can make plant disease detection more accessible to farmers, which can have significant economic and environmental benefits. Further research in this area can explore ways to expand our system to detect more types of diseases and improve the accuracy and efficiency of our approach.

1.1 HISTORY

The detection and management of plant diseases have been a significant concern for farmers and scientists for centuries. The earliest known record of plant disease dates back to ancient Egypt, where farmers used natural remedies to protect their crops. In the early 19th century, the Irish potato famine led to the death of over a million people due to potato blight, emphasizing the importance of plant disease detection and management.

With the advancement of technology, plant disease detection methods have evolved from traditional techniques like visual inspection and chemical analysis to more sophisticated methods like machine learning and deep learning. In the mid-20th century, scientists began using decision trees and other machine-learning techniques to classify plant diseases based on symptoms.

In the 1990s, researchers began exploring the use of neural networks, which are a type of machine learning algorithm that can learn to recognize patterns in data. However, these early neural network models were limited by the availability of data and computing power. In the early 2000s, the development of deep learning techniques revolutionized the field of image recognition, and scientists began exploring the use of deep learning for plant disease detection.

In recent years, there has been a surge of research interest in using deep learning techniques for plant disease detection. The availability of large datasets of plant images, coupled with advances in computing power, has enabled the development of deep learning models that can accurately detect plant diseases with high precision and recall. These models can also learn to detect plant diseases in real time, making them ideal for use in the field.

One of the earliest deep learning-based approaches to plant disease detection was proposed in 2016 by researchers from the University of Bonn in Germany. They developed a deep learning model using convolutional neural networks (CNNs) to classify images of healthy and diseased leaves. Their model achieved an accuracy of over 99%, demonstrating the potential of deep learning techniques for plant disease detection.

Since then, numerous researchers have developed deep-learning models for plant disease detection on a wide range of crops, including tomatoes, potatoes, grapes, and apples. In 2019, researchers from the Indian Institute of Technology Roorkee developed a deep learning model using CNNs and transfer learning to classify tomato leaf diseases. Their model achieved an accuracy of 98.3%, demonstrating the potential of deep learning techniques for plant disease detection on specific crops.

Another recent development in plant disease detection is the use of hyperspectral imaging. Hyperspectral imaging involves capturing images of plants at multiple wavelengths, allowing for more precise detection of plant diseases. In 2020, researchers from the University of California, Davis, developed a deep-learning model using hyperspectral imaging to detect powdery mildew in grapes. Their model achieved an accuracy of 98%, demonstrating the potential of hyperspectral imaging for plant disease detection.

1.2 DISTINCT APPLICATIONS OF CNN

CNNs (Convolutional Neural Networks) have become a popular choice for solving problems in various domains due to their ability to learn and extract meaningful features from complex data. Here are some distinct applications of CNNs:

1.2.1 Image Classification

CNNs are widely used for image classification tasks, such as identifying objects or animals in images. They can learn to recognize specific patterns or features in an image and classify the image into one of several categories.

1.2.2 Object Detection

Object detection is the process of identifying and localizing objects within an image or video. CNNs can be used to detect and identify objects in real time, making them useful for applications such as self-driving cars or security systems.

1.2.3 Facial Recognition

Facial recognition is a biometric technology that identifies or verifies the identity of an individual using their facial features. CNNs can be trained to recognize specific facial features and identify individuals with high accuracy.

1.2.4 Medical Diagnosis

CNNs have been used in medical diagnosis tasks, such as identifying cancerous cells in medical images or diagnosing diseases from medical data. They can learn to identify specific patterns or features in medical data, allowing for accurate diagnosis and treatment.

1.2.5 Autonomous Systems

CNNs are used in autonomous systems, such as autonomous drones or robots, to perform tasks such as object recognition and navigation. They can process sensor data in real-time, allowing for quick decision-making and precise control of the autonomous system.

Overall, CNNs are a powerful tool for solving complex problems in various domains due to their ability to learn and extract meaningful features from complex data.

1.3 SIGNIFICANCE

CNNs (Convolutional Neural Networks) are significant because they have revolutionized the field of computer vision by achieving state-of-the-art performance on a wide range of visual recognition tasks. Here are some of the significant benefits of CNNs:

1.3.1 High Accuracy

CNNs are highly accurate in recognizing and classifying objects in images or videos. They can learn to extract relevant features from images and use them to make predictions with high accuracy.

1.3.2 Efficient Learning

CNNs are efficient in learning complex patterns in data. They can automatically learn and extract relevant features from large datasets, reducing the need for manual feature engineering.

1.3.3 Flexibility

CNNs are highly flexible in their architecture and can be customized to suit different applications. They can be trained on various types of data, including images, audio, and text.

1.3.4 Robustness

CNNs are robust to noise and variations in input data, such as changes in lighting, color, or scale. They can learn to recognize objects under various conditions and generalize well to new data.

1.3.5 Real-time Processing

CNNs are capable of processing large amounts of data in real time, making them useful for applications such as autonomous driving, robotics, and video analysis. Overall, CNNs are significant because they have opened up new possibilities in computer vision, enabling machines to perceive and understand the world in a way that was previously impossible. They are powering a wide range of applications, from autonomous systems to medical diagnosis, and are likely to continue to play a significant role in advancing AI and machine learning in the years to come.

1.4 Challenges in Building CNN-based Systems

While CNNs (Convolutional Neural Networks) have shown great promise in a wide range of applications, there are several challenges in building CNN-based systems:

1.4.1 Data Availability and Quality

CNNs require large amounts of labeled data for training, which can be difficult and time-consuming to acquire. Additionally, the quality of the data can significantly affect the performance of the CNN, so it is crucial to ensure that the data is accurate and representative of the problem domain.

1.4.2 Overfitting

CNNs can easily overfit the training data, which means that they become too specialized in recognizing specific patterns in the training data and perform poorly on new data. Overfitting can be mitigated by techniques such as regularization, dropout, and early stopping.

1.4.3 Architecture Design

The design of the CNN architecture can significantly affect its performance. It is often challenging to choose the optimal number and size of layers, filters, and other hyperparameters.

1.4.4 Computational Resources

CNNs require significant computational resources, such as processing power, memory, and storage, especially for large datasets or complex architectures. This can limit the applicability of CNNs in resource-constrained environments.

1.4.5 Interpretability

CNNs are often considered "black boxes," meaning that it can be challenging to understand how the model makes its predictions. This lack of interpretability can limit their use in applications where the explanation of the decision-making process is crucial.

1.4.6 Domain-Specific Challenges

Building CNN-based systems for specific domains, such as medical diagnosis or agriculture, can present additional challenges such as data privacy, ethical considerations, or the need for domain-specific expertise.

Overall, building CNN-based systems requires careful consideration of these challenges, as well as ongoing efforts to improve the accuracy, efficiency, and interpretability of CNNs.

1.5 DRAWBACKS

While CNNs (Convolutional Neural Networks) have shown remarkable performance on a wide range of visual recognition tasks, there are some drawbacks to using CNNs:

1.5.1 Data Requirements

CNNs require a large amount of labeled training data to achieve high accuracy. Gathering and labeling data can be time-consuming and costly, especially for niche applications with limited datasets.

1.5.2 Computational Resources

CNNs require significant computational resources, such as processing power, memory, and storage. Training large models can take days or even weeks on high-end hardware, limiting their use in resource-constrained environments.

1.5.3 Interpretability

CNNs are often considered as "black boxes" due to their complex architecture and millions of parameters. It can be challenging to understand how the model makes its predictions, making it difficult to explain its decisions to stakeholders.

1.5.4 Overfitting

CNNs can easily overfit the training data, which means that they become too specialized in recognizing specific patterns in the training data and perform poorly on new data. Overfitting can be mitigated by techniques such as regularization, dropout, and early stopping.

1.5.5 Limited Robustness

CNNs are often sensitive to variations in input data, such as changes in lighting, color, or scale. They can also be vulnerable to adversarial attacks, where small, deliberate changes to the input can cause the model to misclassify the image.

1.5.6 Transferability

CNNs trained on one dataset or task may not generalize well to other datasets or tasks. This means that models trained on a specific task may not be useful for other tasks without significant retraining or fine-tuning.

Overall, while CNNs have many advantages and have achieved state-of-the-art performance on a wide range of tasks, it is essential to consider their limitations and trade-offs when designing and deploying CNN-based systems

CHAPTER 2
LITERATURE SURVEY

The paper “Potato Leaf Disease Classification using Deep Learning Approach” by Rizqi Amaliatus Sholihati, Indra Adji Sulistijono, Anhar Risnumawan, and Eny Kusumawati presents the usage of VGG16 and VGG19 deep learning models for classification of potato leaf disease along with data augmentation to solve the overfitting problem. The dataset used consists of 5100 images categorized into 5 categories. Both models achieved an accuracy of about 91%.

The paper “Plant Leaf Disease Classification using EfficientNet Deep Learning Model” by Ümit Atila, Murat Uçar, Kemal Akyol, and Emine Uçar presents the usage of the EfficientNet CNN model for the classification of Plant diseases. It compares the proposed method with other CNN models like AlexNet, VGG16, Inception V3, and ResNet50.The dataset used in this study is PlantVillage consists of 14 different plant species. The results show that EfficientNet used achieved the highest accuracy when compared with other models.

The paper “Deep Learning Utilization in Agriculture: Detection of Rice Plant Diseases Using an Improved CNN Model” by Ghazanfar Latif, Sherif E. Abdelhamid, Roxane Elias Mallouhy, Jaafar Alghazo, and Zafar Abbas Kazimi presents the usage of fine-tuned transfer learning VGG19 CNN model for classification of Rice Plant Diseases using images along with data augmentation to solve overfitting. The model proposed can identify 6 categories of rice plant diseases. The proposed method achieved an accuracy of 96.08%.

The paper “Classification of Beans Leaf Diseases using Fine Tuned CNN Model” by Vimal Singh, Anuradha Chug, and Amit Prakash Singh presents the usage of CNN models like MobileNetV2, EfficientB6, and NasNet pre-trained models along with optimizers like Adam, RMSProp, and Nadam for classification of Beans Leaf Diseases using TensorFlow and Keras. The results show that EfficientNetB6 showed the highest accuracy of 96.62% when the Adam optimizer was used.

The paper “Tomato Crop Disease Classification Using pre-trained deep learning algorithm” by Aravind Krishnaswamy Rangarajan, Raja Purushothaman, and Aniirudh Ramesh presents the usage of fine-tuned pre-trained AlexNet and VGG16 CNN models for classification of Tomato crop diseases into 6 different categories. The classification accuracy using 13,262 images was found to be 97.29% for VGG16 net and 97.49% for AlexNet. The dataset used is named “Plant Village.”

The paper “Plant Disease Classification using deep learning” by Akshai KP, and J. Anitha presents the usage of pre-trained VGG19, DenseNet, and ResNet CNN models with ImageNet weights for the classification of Plant Diseases of 14 different species. The results show that DenseNet achieved the highest accuracy of 98%.

The paper “Bell Pepper Leaf Disease Classification Using CNN” by Monu Bhagat, Rehan Mahmood, Monu Kumar, Dr. Dilip Kumar, and Bibhudhendra Pati presents the usage of the CNN model for the classification of Bell Pepper Leaf disease detection. This method achieved an accuracy of 96.78%.

The paper “A Comparative Analysis of Deep Learning Models Applied for Disease Classification in Bell Pepper” by Nidhi Kundu, Geeta Rani, and Vijaypal Singh Dhaka presents the comparative research on the accuracy of classification of Deep Learning algorithms namely VGG16, VGG19, ResNet50, ResNet152, InceptionResNetV2, and DenseNet121 for classification bell pepper plant diseases. The dataset used is a publicly available dataset of the bell pepper plant. The results show that DenseNet requires less training time and achieved a training accuracy of 97.49% and a testing accuracy of 96.87%.

The paper “Leaf Disease Detection using Deep Learning” by Utkarsha N. Fulari, Rajveer K. Shastri, and Anuj N. Fulari presents the usage of SVM and AlexNet CNN model for classification of plant diseases of 9 different species into either healthy or disease categories.

The dataset used consists of 12,949 images of healthy and unhealthy leaves such as apples, cherries, corn, grape, peach, etc. The results show that the SVM model acquired an accuracy of 80%, whereas the CNN model acquired an accuracy of 97.71%.

The paper “Tomato Leaf Disease Detection Using Deep Learning Techniques” by Surampalli Ashok, Gemini Kishore, Velpula Rajesh, S. Suchitra, S.G. Gino Sophia, and B. Pavithra presents using CNN model using image segmentation for the classification of Tomato Leaf Disease. The proposed method is compared with other Machine Learning models namely AlexNet and ANN and the proposed method acquired the highest accuracy of 98.12. The proposed model was built using OpenCV.

The paper “Potato Leaf Diseases Detection using Deep Learning” by Divyansh Tiwari, Mrityunjay Ashish, Nitish Gangwar, Abhishek Sharma, Suhanshu Patel, and Dr. Suyash Bharadwaj presents the usage of pre-trained CNN models namely VGG16, VGG19, and InceptionV3 for feature extraction and Classifiers namely SVM, Neural Network, KNN, and Logistic regression. The extracted features from the CNN model are given as input for classifiers for the classification of Potato Leaf Diseases into three categories. The proposed method achieved an accuracy of 97.8% when both VGG19 And Logistic Regression were used for classification.

The paper “Rice Plant Disease Classification using Transfer Learning of Deep Convolutional Neural Network” by V. K. Shrivastava, M. K. Pradhan, S. Minz, and M. P. Thakur presents the usage of the AlexNet CNN model for feature extraction and SVM as the classifier for classification Rice Plant diseases into four categories. The image samples used were collected from Indira Gandhi Agricultural University, Raipur. The dataset consists of 619 images. The proposed method is tested for many training-testing divisions. The proposed method achieved the highest accuracy of 91.37% when the training-testing division was 80-20.

2.1 SUMMARY

Table 2.1 Summary of Existing Approaches

Reference Number	Paper Name	Methodology	Results/Remarks
[1]	Potato Leaf Disease Classification using Deep Learning Approach	Deep Learning using VGG16 and VGG19 CNN architecture.	Achieved an accuracy of 91%.
[2]	Plant Leaf Disease Classification using EfficientNet deep learning model	EffiecientNet Deep Learning CNN architecture.	EffiecientNet architecture achieved the highest accuracy among other deep learning models.
[3]	Deep Learning Utilization in Agriculture: Detection of Rice Plant Diseases Using an Improved CNN Model	VGG19-based transfer learning Deep Learning method.	VGG19-based transfer learning achieved the highest average accuracy 96.08% using the non-normalized augmented dataset.
[4]	Classification of Beans Leaf Diseases using Fine Tuned CNN Model	MobileNetV2, EfficientNetB6, and NasNet.	Analysis of experimental results shows that EfficientNetB6 performs better with 91.74% accuracy than other models.

[5]	Tomato Crop Disease Classification using pre-trained deep learning algorithm	AlexNet And VGG16 CNN architectures.	Achieved an accuracy of 97.29% for VGG16 and 97.49% for AlexNet.
[6]	Plant Disease Classification using deep learning	CNN, RESNET, VGG, and DenseNet models.	Achieved an accuracy of 94.58% for CNN, 95.32% for VGG,97.04% for RESNET, and 98.27% for DENSENET.
[7]	Bell Pepper Leaf Disease Classification Using Deep Learning	CNN model.	Achieved an accuracy of 96.78%.
[8]	A Comparative Analysis of Deep Learning Models Applied for Disease Classification in Bell Pepper	VGG, ResNet, DenseNet, and InceptionResNet.	'DenseNet' requires less training time and gives the highest validation accuracy among all the above-stated models. It achieved an accuracy of 96.87%.
[9]	Leaf Disease Detection using Deep Learning	SVM and CNN model.	Achieved an accuracy of 80% with SVM and 97.71 % for the CNN model.
[10]	Tomato Leaf Disease Detection Using	CNN model.	Achieved an accuracy of 98%.

	Deep Learning Techniques		
[11]	Potato Leaf Diseases Detection using Deep Learning	Extracted features from pre-trained CNN models are passed to classifiers such as SVM, KNN, and Neural networks.	Achieved an accuracy of 97.8%.
[12]	Rice plant disease classification using Transfer Learning of Deep Learning Convolutional Neural Networks.	Extracted features from AlexNet are fed into SVM.	Achieved an accuracy of 91.37%.

2.2 Drawbacks of Existing Approaches

The proposed methods lack a user-friendly platform to utilize the built model. Furthermore, the dataset used in these methods was divided into two categories: training and testing, which may raise concerns regarding overfitting. Additionally, a significant number of these methods did not utilize data augmentation techniques, which may lead to overfitting.

2.3 Convolutional Neural Networks (CNN or ConvNet)

Convolutional Neural Networks (CNN or ConvNet) are a fascinating subset of machine learning that has revolutionized the field of computer vision. While machine learning techniques encompass a wide range of algorithms and models, CNNs are designed for tackling image-related tasks. Their architecture and working principles make them incredibly effective at processing visual data and extracting meaningful features.

At the core of a CNN lies its ability to uncover intricate patterns and details within images. By leveraging principles from linear algebra, CNNs analyze the binary representation of visual data to identify specific features that are relevant for classification and recognition. Interestingly, CNNs can also be employed to classify other types of data, such as time series, audio signals, and even text.

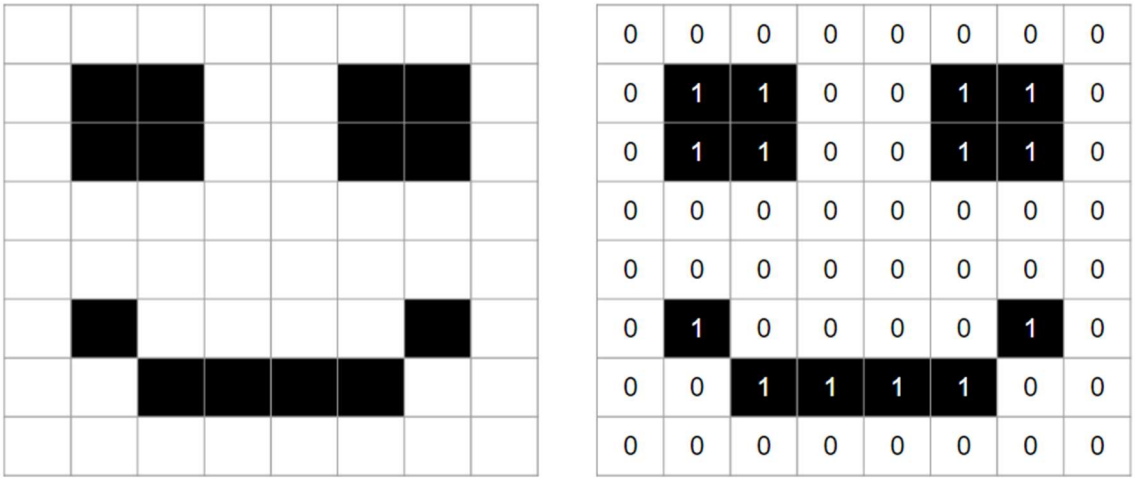


Fig 2.1 Binary Representation of a black and white image. (Courtesy: Source [13])

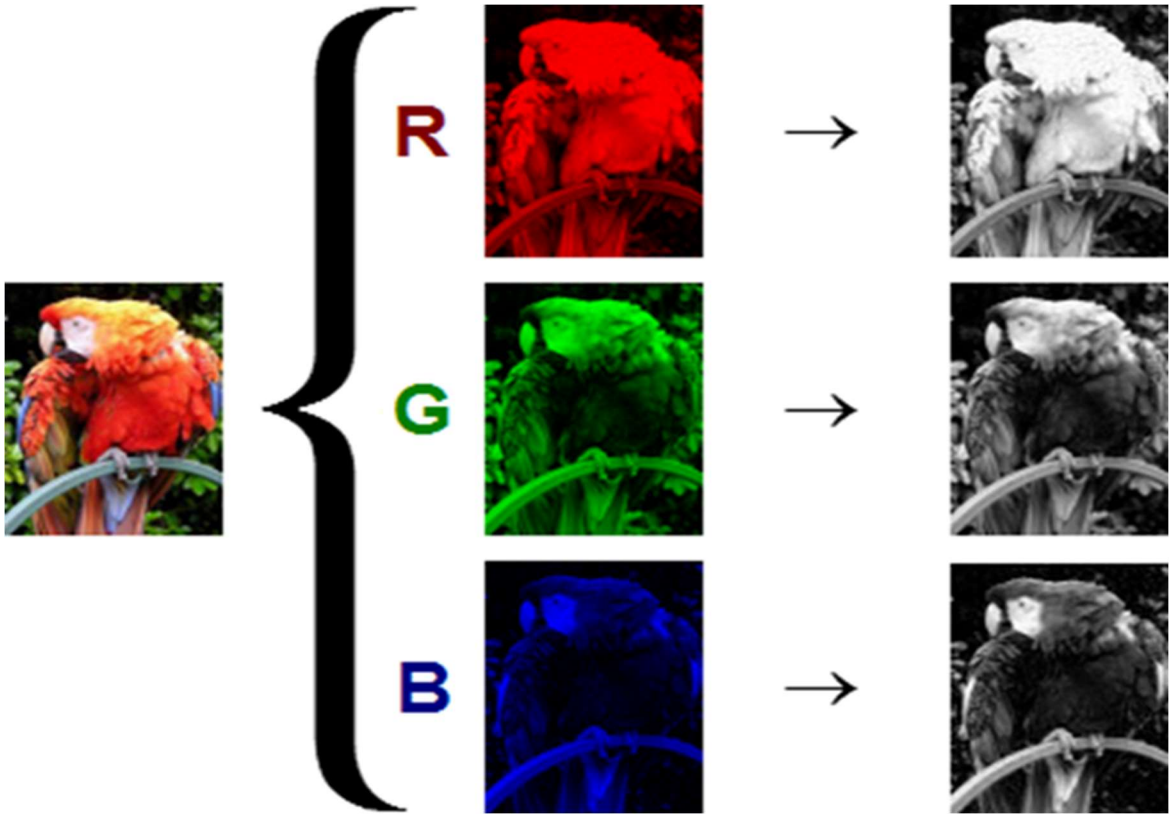


Fig 2.2 Representation of a color image in RGB format (Courtesy: Source[14])

One of the most remarkable advantages of CNNs is their capability to reduce human efforts when it comes to detecting features in images by autonomously learning and identifying the important features. That makes CNNs highly suitable for various applications where object recognition plays a vital role, ranging from self-driving cars and facial recognition systems to medical imaging and industrial quality control.

Conceptually, the architecture of a CNN resembles the intricate connectivity of neurons in the human brain. Just like our brain consists of billions of neurons, CNNs consist of numerous nodes that act as artificial neurons. These neurons are arranged in a specific way, mirroring the organization of the frontal lobe in the brain, which is responsible for processing visual stimuli. This structural similarity enables CNNs to capture and process visual information in a way that closely resembles how our brains perceive and interpret images.

CNN Architecture

A typical CNN model consists of 5 layers namely:

1. Input Layer
2. Convolutional Layer
3. Pooling Layer
4. Fully Connected Layer
5. Output Layer

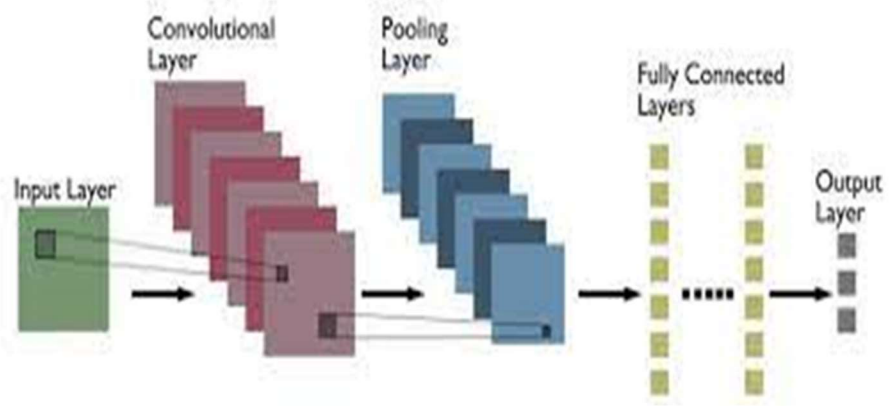


Fig 2.3 Convolutional Neural Network Architecture (Courtesy: Source[15])

From the convolutional layer to the fully connected layer the complexity of the CNN increases significantly, this increasing complexity of the CNN helps them in identifying the more complex features of the image.

1. Input Layer

Input layer is the most important layer in every neural network model. This is the layer that takes input.

2. Convolutional Layer

The convolutional layer serves as the fundamental building block of Convolutional Neural Networks (CNNs), bearing the crucial responsibility of handling the computational load within the network. This layer operates by performing a dot product operation between matrices, wherein one matrix comprises the set of learnable parameters commonly referred to as the kernel or filter. The other matrix corresponds to a restricted portion of the reception field, essentially a subsection of

the input image. It is important to note that the kernel is typically smaller than the input image but possesses a greater depth, extending up to three channels in the case of RGB images.

During the forward pass of the CNN, the kernel slides across the image representation, systematically examining each receptive field and generating a 2-D representation as a result. This 2-D representation is commonly known as an activation map, feature map, or convolved features. The size of the stride, which determines the amount of sliding that occurs between each kernel operation, plays a significant role in controlling the spatial dimensions of the resulting activation map.

The stride value allows for flexibility in adjusting the level of spatial information preservation in the generated feature map. The large stride value will result in reduced size of the output feature map, as the kernel skips over the great number of pixels during its sliding process. Conversely, a smaller stride value leads to a more detailed feature map with a higher spatial resolution, as the kernel analyzes each receptive field more comprehensively.

Overall, the convolutional layer plays a critical role in CNNs by facilitating the extraction of local features through the convolution operation. This process enables the network to learn and discern essential patterns and visual elements within the input data.

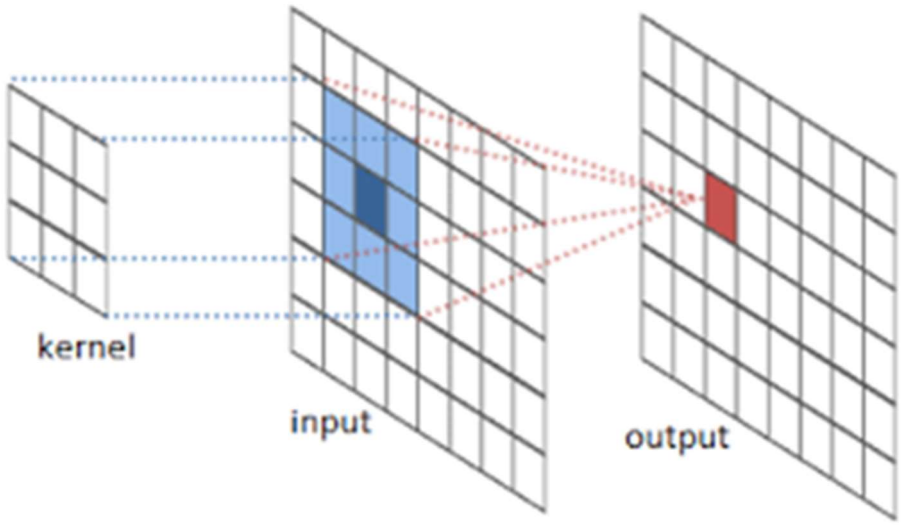


Fig 2.4 Working of convolutional layer (Courtesy: Source[15])

Wout=(W-F+2P)/S+1

where

1. Wout- output width of image
2. F-Spatial size
3. P-Amount of padding
4. S-Stride

If we have an input of size WxWxD and Dout number of kernels then the convolutional layer will yield an output of size Wout x Wout x Dout.

3. Pooling Layer

The pooling layer is another important component of CNN. It works with the convolutional layer to further enhance the network's capability to analyze images.

The main function of the pooling layer is to downsample the feature maps generated by reducing the spatial dimensions of the feature maps. The pooling layer help in retaining the important and relevant information of the images while reducing the computation complexity of the network.

To reduce the spatial dimensions of the feature maps we use a fixed-size window called as pooling window of the filter. The filter is applied to the feature map and within each window a value is selected by using the pooling operations and this process summarizes the presence of a specific feature in that region.

Pooling layers also contain a parameter known as the stride which is similar to the convolutional layer. The stride determines the amount of sliding that occurs in each pooling operation, enabling control over the output size.

There are two commonly used pooling operations:

1 **Max Pooling**

It takes the max value in the pooling window.

2 **Average pooling**

It takes the average of the pooling values in the pooling window.

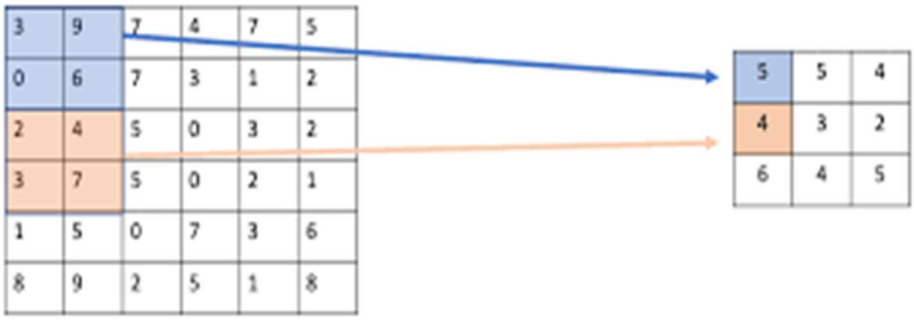


Fig 2.5 Representation of how the pooling layer works (Courtesy: Source[16])

If we have an activation map of size WxWxD and pooling kernel of size F and stride S Then the size of output volume is Wout x Wout x D, where Wout=(W-F)/S+1.

4. Fully Connected Layer

The fully connected layer, known as the dense layer, is an integral part of Convolutional Neural Networks (CNNs) that follows the convolutional and pooling layers. Its main purpose is to perform high-level reasoning and decision-making based on the features extracted by the preceding layers. Nodes in this layer have full connectivity with all nodes in the preceding and succeeding layers as seen in regular Fully Connected Neural Networks. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.

The fully connected layer typically takes the flattened output from the preceding layers as its input. This means that the 3D feature maps are reshaped into a 1D vector before being fed into the fully connected layer. This vector represents a compact representation of the extracted features, capturing their spatial relationships and patterns.

Each node in the fully connected layer receives input from all the nodes in the previous layer. The connections between nodes in the fully connected layer are accompanied by learnable weights and biases, which are adjusted during the training process to optimize the network's performance.

The output of the fully connected layer is usually passed through an activation function, such as the rectified linear unit (ReLU) or SoftMax function, to introduce non-linearities and enable the network to model complex relationships between the features.

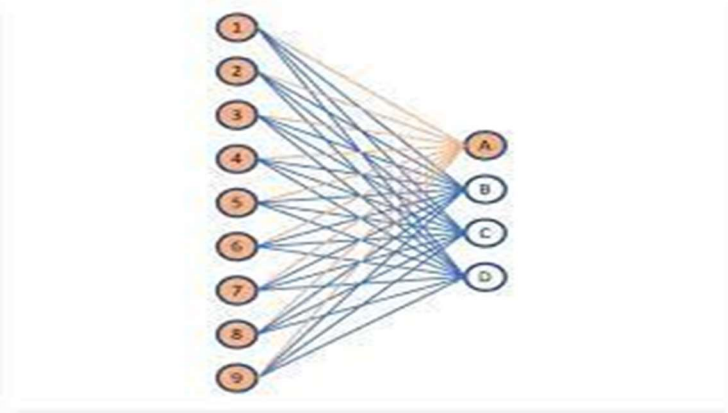


Fig 2.6 Representation of fully connected layers (Courtesy: Source[17])

5. Output Layer

This is the important layer in the network this layer is used for prediction or for getting the output. There are several types of CNN models, each with its own strengths and weaknesses. Here are some of the commonly used CNN models:

1. LeNet
2. GoogleNet
3. VGGNet
4. AlexNet
5. DenseNet
6. ResNet

There are many versions of the above mention CNN models. The newer versions have resulted in state-of-the-art accuracies.

CHAPTER 3
PROPOSED METHOD

3.1 PROBLEM STATEMENT

Plant diseases present a significant threat to food security and the agricultural industry globally. When a plant gets infected, it can cause a reduction in crop yield, ultimately leading to a shortage of food. These effects are particularly devastating for developing countries, where many people depend on agriculture for their livelihoods. Thus, it is crucial to detect plant diseases accurately and quickly to prevent the spread of the disease to other plants. Traditionally, the Detection and diagnosis of plant diseases are relied on visual inspection by trained experts. However, this approach is time-consuming and labor-intensive, making it difficult to scale up and deploy in large agricultural areas. The other way to detect is by using machine learning algorithms. Despite the availability of numerous classification models for detecting plant diseases, there are still limited online platforms available for utilizing these models to identify specific diseases. This lack of infrastructure can make it difficult for farmers and other stakeholders to access accurate information and make informed decisions about disease management.

While detecting plant diseases is important, it is not sufficient to prevent their spread and minimize the damage they cause. Applying the appropriate chemicals and treatments for each specific disease is equally important. Failure to do so can result in continued disease spread, crop damage, and financial losses for farmers. Additionally, incorrect chemicals can have unintended consequences, such as harming beneficial organisms or polluting nearby water sources. Therefore, it is crucial to ensure that farmers can access accurate information regarding the specific plant diseases affecting their crops and the chemicals or treatments required to prevent them.

To address these challenges, a platform for accurate identification of plant diseases is needed. "Deep Learning Plant Disease Detection" is a platform that uses deep learning techniques to identify plant diseases. The platform uses the "CNN" algorithm, widely known for its high accuracy in image classification, enabling it to accurately identify plant diseases from images.

Furthermore, the platform offers recommendations for preventing and managing the spread of crop diseases using its user-friendly web interface.

3.2 OBJECTIVE

a. To develop a robust CNN model that can accurately classify images. The model will be trained on a dataset containing images of different diseases. The CNN architecture will be designed to extract relevant features from the images and make predictions based on those features.
b. CNN model will be specifically designed to classify images and detect diseases . It will be trained on a diverse dataset that includes images representing various diseases. The objective is to achieve high accuracy in classifying and identifying the presence of a particular disease in an image.
c. Alongside the CNN model development, the project aims to create a user-friendly website for deploying the model. The website will provide an intuitive interface for users to upload their images for disease detection. It will have clear instructions on how to capture and upload images and ensure a smooth user experience
d. The website will integrate the trained CNN model to analyze the uploaded images and detect the presence of diseases. Based on the detection results, the system will provide recommendations for disease prevention and management. These recommendations may include treatment options, preventive measures.

3.3 EXPLANATION

This section includes an explanation of the proposed solution, which includes the following sections;

1. Software and Hardware Requirements.
2. CNN models.
3. Architecture Diagram.
4. Modules and their description

3.3.1 Software-Hardware Requirements

Software Requirements:

1. **Programming Languages:** Python.
2. **Scripting Languages** : HTML, CSS.
3. **IDE** : VSCode, Google Colab.
4. **Frameworks** : TensorFlow, Keras, and Bootstrap.
5. **API** : Flask
6. **Libraries** : Matplotlib pyplot, and NumPy.

Hardware Requirements:

1. **CPU** : Minimum intel i5 8th generation processor.
2. **GPU** : NVIDIA GeForce RTX or NVIDIA Tesla V100.
3. **RAM** : Minimum 16 GB.
4. **Storage** : As per the requirements.
5. Internet Connection

TensorFlow

Fig 3.1 TensorFlow (Courtesy: Source[18])

The word Tensor is made up of two words namely, Tensor and Flow. Tensor is a multidimensional array and flow is used to represent the flow of data. TensorFlow is one of the popular frameworks used in machine learning. It is a free and open-source framework released in 2015 by Google. It is used for numerical computation and data flow which helps in making machine learning easier and faster. TensorFlow is completely based on Python language.

TensorFlow is used to define the flow of data in operation on a Tensor. TensorFlow is popular because it is accessible to everyone and it can also integrate different APIs to create deep learning architectures.

Use cases of TensorFlow

TensorFlow is mainly used for deep learning or machine learning problems and here are some use cases of TensorFlow:

1. Voice Recognition.
2. Image Classification.
3. Image Segmentation.
4. Time Series data.
5. Video Detection.
6. Text-based applications like sentimental analysis, fraud detection, etc.

Features of TensorFlow

1. Flexible.
2. Easily Trainable.
3. Open Source.
4. Parallel training.
5. Large Community.
6. Layered Components.
7. Responsive Construct.

Keras

Fig 3.2 Keras (Courtesy: Source[19])

Keras is a deep learning API developed by Google for working with neural networks. Keras is completely written in Python and helps in making the implementation of neural networks easier. It can also support multiple backend neural network computations.

Keras runs on top of open-source machine learning libraries like Theano, TensorFlow, or Cognitive toolkit. Keras has many implementations of commonly used building blocks neural networks such as layers, activation functions, optimizers, and tools for working with image and text data.

Keras is one of the popular choices for deep learning projects, and it is used by many companies including Google, Facebook, and Microsoft. It can also be used for various applications like image classification, speech recognition, and natural language processing.

Features of Keras

Keras makes high-level neural network API easier and more performant by using various optimization techniques. Keras has the following features-

1. It is a consistent, simple, and extensible API.
2. It has minimal structure-it can easily achieve results.
3. It is a user-friendly framework, that will run on both GPU and CPU.
4. It can support multiple platforms and backends.
5. It has high scalability of computation.

Benefits of using Keras

Keras is a highly powerful and dynamic framework and has the following advantages-

1. It is easy to test using Keras.
2. It supports both CNN and RNN.
3. It is open-source and has a large community.
4. Keras neural networks are written in Python, so the code will be simpler.
5. It is easy to use and the API is simple and intuitive.
6. It is designed to be fast and can be used to build deep neural networks that work on large neural networks.

Numpy

Fig 3.3 NumPy (Courtesy: Source[20])

NumPy stands for Numerical Python. NumPy is a Python library created in 2005. It is an open-source and free-to-use library designed for working with arrays. It has various functions for working in the domain of linear algebra, and matrices.

In Python, there are no arrays and we use lists to serve the process of arrays, but they are slow to process.

Using lists in machine learning may increase the time of computation as they are slow to process. But, NumPy provides us with an array object which 50 times faster when compared to the general Python lists. The array object provided by NumPy is called ndarray, and it also has many supporting functions to make it easy to use.

Unlike lists, NumPy arrays are stored in a single, continuous block of memory. This makes them much faster to access and manipulate, as the CPU does not have to jump around in memory to find the data it needs. NumPy arrays are also optimized to work with the latest CPU architectures, which further improves their performance.

Benefits:

Benefits of using NumPy include-

1. NumPy arrays are faster compared to traditional lists, which can n crease the speed of the process.
2. It is easy to use and has tools for integrating with C, C++, and Fortran code.
3. NumPy arrays are very powerful and can be used for a variety of tasks such as machine learning and data science.

Matplotlib Pyplot:

Matplotlib is a library in Python used for creating static, animated, and interactive visualizations. It provides us with an API for embedding plots into applications and interactive environments. Pyplot is a sub-module of Matplotlib that provides us with an interface like MATLAB for creating graphs. Pyplot is easy to learn and use.

The pyplot API consists of a hierarchy of Python code objects and includes numerous functions topped by matplotlib.pyplot. here are some of the commonly used pyplot functions:

1. plot(): Plots a line or scatter plot.
2. bar(): Plots a bar chart.
3. hist(): Plots a histogram.
4. pie(): Plots a pie chart.
5. scatter(): Plots a scatter plot.
6. subplots(): Creates a subplot grid.
7. show(): Displays the current figure.

Bootstrap

Fig 3.4 Bootstrap(Courtesy: Source[21])

Bootstrap is a free and open-source CSS framework created by Mark Otto and Jacob Thornton in the year 2011. It is directed at responsive, mobile-first front-end development. It contains HTML, CSS, and JavaScript-based designs for typography, forms, buttons, and other interface components.

It was originally intended to be used at Twitter, but it quickly gained popularity and become one of the popular CSS frameworks. Bootstrap is easy to use, customizable, and includes many pre-built components which are useful in web development. Bootstrap also has a wide number of JavaScript plugins, which can be used to add functionality to web pages.

Benefits of Bootstrap:

1. It is easy to use.
2. It is customizable.
3. It is open-source and has a large community.
4. It is useful for creating responsive web pages.
5. It has many pre-built components that help us on more focusing on logic.

Flask API

Fig 3.5 Flask (Courtesy: Source[22])

A lightweight Python web framework with an emphasis on flexibility and simplicity is called Flask. It adopts a "micro-framework" strategy, offering just the necessities for creating web applications. Flask offers a variety of extensions for further functionality and lets developers organize their code as they like. It is simple to get started with and obtain support thanks to its extensive documentation and vibrant community. Flask allows for the separation of presentation logic via templates, and it is simple to deploy to a variety of hosting services or use as a standalone server. In general, Flask is a well-liked option for developers seeking a simple and adaptable framework for creating web apps with Python.

Django

Django is a high-level Python web framework that follows the model-view-controller (MVC) architectural pattern. It provides a set of tools and features to help developers build scalable and maintainable web applications quickly and efficiently. Here's some information about Django

1. **MVC Architecture** Django follows the MVC architectural pattern, although it refers to it as the Model-View-Template (MVT) pattern. The model represents the data structure, the view handles the logic for processing requests and rendering responses, and the template defines the presentation layer.
2. **Rapid Development** Django emphasizes rapid development by providing a lot of built-in features and functionality. It includes an object-relational mapper (ORM) that allows

developers to interact with databases using Python objects, form handling, authentication, session management, URL routing, and more.

3. **DRY Principle** Django promotes the "Don't Repeat Yourself" (DRY) principle, which means that developers should avoid duplicating code. It provides reusable components and encourages modular design to reduce code duplication and increase maintainability.
4. **Database Support** Django supports various databases, including PostgreSQL, MySQL, SQLite, Oracle, and others. It abstracts the database layer with its ORM, allowing developers to write database-agnostic code.
5. **Admin Interface** Django includes a powerful built-in admin interface that allows developers to create, update, and manage the content of the website without writing
6. much code. It automatically generates forms, lists, and detailed views based on the models defined in the application.
7. **Template Engine** Django comes with a template engine that enables developers to separate the design from the Python code. Templates are written using Django's template language, which provides control structures, variable substitution, and filters for manipulating data.
8. **Security** Django incorporates built-in security features to help developers prevent common web application vulnerabilities. It includes protection against cross-site scripting (XSS), cross-site request forgery (CSRF), SQL injection, and clickjacking.
9. **Scalability and Performance** Django is designed to handle high-traffic websites and can scale horizontally by employing load-balancing techniques and caching. It also provides tools like Django's database connection pooling to enhance performance.
10. **Third-Party Ecosystem** Django has a vibrant third-party ecosystem with numerous reusable applications and packages available. These packages can be easily integrated into Django projects to add additional functionality or solve specific problems.
11. **Community and Documentation** Django has a large and active community of developers, which means there are ample resources available for learning and troubleshooting. The official documentation is comprehensive and well-maintained, covering all aspects of Django development.

Django is widely used by developers around the world to build a wide range of web applications, from small personal projects to large-scale enterprise solutions. It provides a solid foundation for developing robust and secure web applications using Python

3.3.2 ConvNet models

3.2.2.1 VGG16

VGG-16 is a type of CNN model that was developed by Visual Studio Group.VGG16 is a very deep model with 16 layers. This model achieved n accuracy of 92.7% on the ImageNet dataset which consists of 14 million images belonging to 1000 different classes. It is one of the models that achieve the state-of-the-art results.

VGG16 is a powerful model for image classification, and it has been used for a variety of applications like image classification, image detection, image detection, and image retrieval.
VGG16 is also popular for using transfer learning, i.e., using a pre-trained model for training a new model for an entirely different type of application. VGG16 makes an improvement over AlexNet by decreasing the large kernel-sized filters to multiple smaller 3x3 filters one after another.

VGG16 Architecture

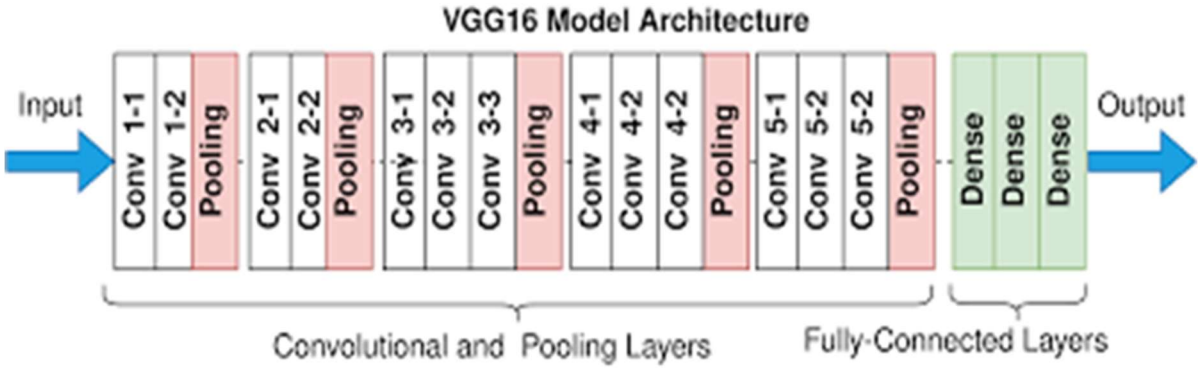


Fig 3.6 Representation of VGG16 Architecture. (Courtesy: Source[23])

The VGG16 network contains an input layer,13 convolutional layers, 5 pooling layers, 3 fully connected layers, and an output layer.

1. **Input layer**

   The input layer is the crucial layer in the VGG16 model. This is the layer where input is taken into the network. The input can be an image of size 224x224x3. That input is passed to the succeeding layers.

2. **Convolutional layer**

   The VGG 16 model has 13 convolutional layers that can extract features and are in with a combination of pooling layers as shown in fig 3.6. Each convolutional layer has 3x3 filters and stride as 1, The first convolutional layer has 64 filters, the next 2 convolutional layers have 128 filers, the next 3 succeeding convolutional layers have 256 filters and the last 5 convolutional layers have 512 filters.

3. **Pooling layers**

   Polling layers used in VGG16 are max-pooling layers, max-pooling layers take maximum values from the filter window. Pooling layers have a filter of size 2x2 ad a stride of 2.

4. **Fully Connected Layers**

   There are a total of 3 fully connected layers or dense layers in the VGG16 model. The first two dense layers have 4096 nodes, and the last fully connected layers have 1000 nodes.

5. **Output Layer**

The output layer has about 1000 nodes, one for each class in the ImageNet dataset, and this layer is responsible for the classification of images.

Advantages of using the VGG16 model:

1. It is effective and one the most powerful CNN models.
2. It can be used for a variety of tasks such as image classification, object detection, image retrieval, and image segmentation.
3. It can achieve state-of-the-art results.
4. It can be used for transfer learning.

Disadvantages of using the VGG16 model:

1. It is expensive to train the model.
2. It has a very large number of parameters which makes it difficult to train and deploy the model.
3. It is not efficient compared to the new CNN models.

3.2.2.2 VGG19

VGG19 is a type of CNN model that was developed by Visual Studio Group.VGG16 is a very deep model with 19 layers. This model achieved an accuracy of 93.3% on the ImageNet dataset which consists of 14 million images belonging to 100 different classes. It is one of the models that achieve the state-of-the-art results.

VGG19 is a powerful model for image classification, and it has been used for a variety of applications like image classification, image detection, image detection, and image retrieval.VGG19 is also popular for using transfer learning, i.e., using a pre-trained model for training a new model for an entirely different type of application.

The main difference between VGG16 and VGG19 is that vGG6 has 13 convolutional layers whereas VGG19 has 16 convolutional. Though it is complex when compared to VGG16, VGG19 can produce high accuracy.

VGG19 Architecture

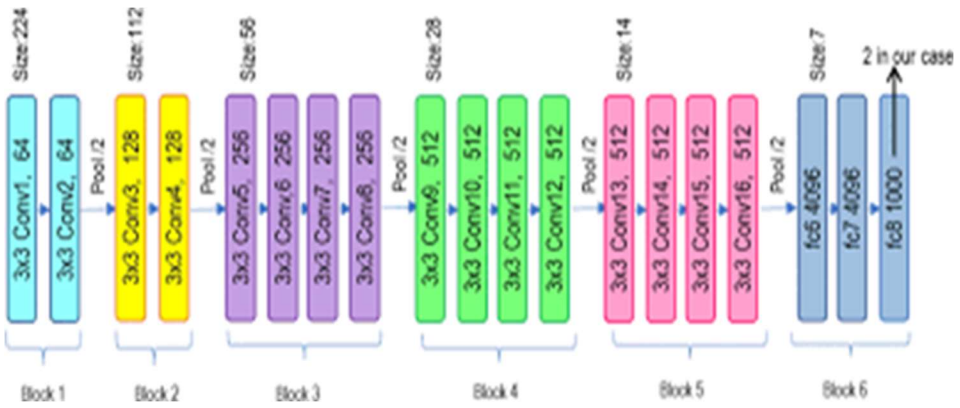


Fig 3.7 Representation of VGG19 Architecture(Courtesy: Source[24])

The VGG19 network contains an input layer,16 convolutional layers, 5 pooling layers, 3 fully connected layers, and an output layer.

1. **Input layer**

   The input layer is the crucial layer in the VGG19 model. This is the layer where input is taken into the network. The input can be an image of size 224x224x3. That input is passed to the succeeding layers.

2. **Convolutional layer**

   The VGG19 model has 16 convolutional layers that can extract features and are in combination with pooling layers as shown in fig 3.7. Each convolutional layer has 3x3 filters and stride as 1, The first two convolutional layers have 64 filters, the next 2 convolutional layers have 128 filers, the next 4 succeeding convolutional layers have 256 filters and the last 8 convolutional layers have 512 filters.

3. **Pooling layers**

   Polling layers used in VGG16 are max-pooling layers, max-pooling layers take maximum values from the filter window. There are 5 max-pooling layers in VGG19. Pooling layers have a filter of size 2x2 ad a stride of 2.

4. **Fully Connected layers**

   There are a total of 3 fully connected layers or dense layers in the VGG16 model. The first two dense layers have 4096 nodes, and the last fully connected layers have 1000 nodes.

5. **Output layer**

   The output layer has about 1000 nodes, one for each class in the ImageNet dataset, and this layer is responsible for the classification of images.

Advantages of using the VGG19 model

1. It is effective and one the most powerful CNN models.
2. It can be used for a variety of tasks such as image classification, object detection, image retrieval, and image segmentation.
3. It can be used for transfer learning.
4. It can achieve state-of-the-art results.

Disadvantages of using the VGG19 model

1. It is expensive to train the model.
2. It has a very large number of parameters which makes it difficult to train and deploy the model.
3. It is not efficient compared to the new CNN models.

3.2.2.3 InceptionV3

InceptionV3 is a convolutional neural network model that was developed by Google in 2015. It is a successor to InceptionV2. InceptionV3 is a deep model, with 299 layers. It has achieved state-of-the-art results on the ImageNet dataset, which consists of 14 million images belonging to 100 different classes.

InceptionV3 was developed using the inception architecture, which was first used in its predecessor InceptionV1. Inception architecture is designed to be efficient in terms of both computation and memory. Inception architecture uses a combination of different convolution operations, including 1x1, 3x3, and 5x5 convolutions.

InceptionV3 user various techniques like label smoothing, factorized convolutions, and auxiliary classifiers to improve its performance.

InceptionV3 Architecture

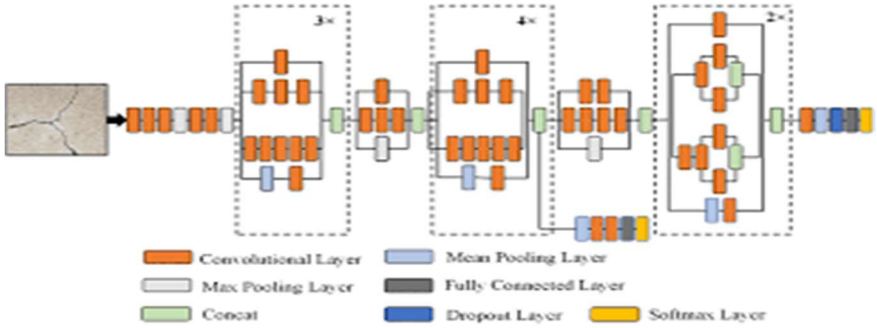


Fig 3.8 Representation of InceptionV3 Architecture (Courtesy: Source[25])

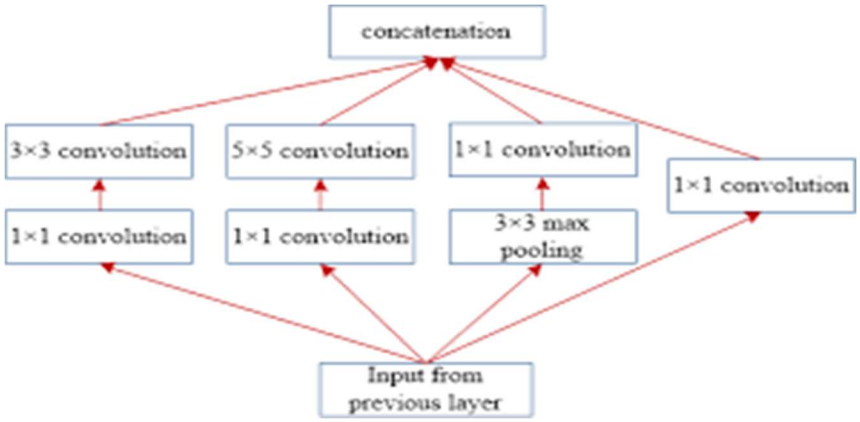


Fig 3.9 Representation of Inception Block(Courtesy: Source[26])

The Inception v3 architecture is divided into four stages

1. The first stage consists of two convolutional layers followed by a max pooling layer. The first convolutional layer has 64 filters of size 1x1, and the second convolutional layer has 128 filters of size 3x3. The max pooling layer has a kernel size of 2x2 and a stride of 2.
2. The second stage consists of two Inception modules followed by a max pooling layer. Each Inception module has a convolutional layer with 192 filters of size 1x1, followed by a convolutional layer with 128 filters of size 3x3, followed by a convolutional layer with 96 filters of size 3x3. The max pooling layer has a kernel size of 2x2 and a stride of 2.

3. The third stage consists of four Inception modules followed by a max pooling layer. Each Inception module has a convolutional layer with 256 filters of size 1x1, followed by a convolutional layer with 192 filters of size 3x3, followed by a convolutional layer with 160 filters of size 3x3. The max pooling layer has a kernel size of 2x2 and a stride of 2.
4. The final stage consists of three Inception modules followed by an average pooling layer and a fully connected layer. Each Inception module has a convolutional layer with 384 filters of size 1x1, followed by a convolutional layer with 192 filters of size 3x3, followed by a convolutional layer with 128 filters of size 3x3. The global average pooling layer reduces the dimensionality of the feature maps to 1024. The fully connected layer has 1000 neurons, and each node is associated with a label in the ImageNet dataset.

Advantages of using the InceptionV3 network

1. Error rate of InceptionV1 is less compared to the other models.
2. It is relatively efficient compared to the other CNN architectures.
3. It is flexible and can be used for other applications such as image classification, object detection, image retrieval, and image segmentation.

Disadvantages of using InceptionV3

1. It is expensive to train the model.
2. It has a very large number of parameters which makes it difficult to train and deploy the model.
3. It is not efficient compared to the new CNN models such as ResNets

3.2.2.4 DenseNet201

DenseNet201 is a type of CNN model that was proposed by Zhuang Liu, Gao Huang, and Kaiming He in the paper "Densely Connected Convolutional Networks" in 2016. This model achieved an error rate of 3.57% on the ImageNet dataset which consists of 14 million images belonging to 100 different classes.

DenseNet201 is an efficient and powerful CNN architecture that can be used to achieve state-of-the-art results on image classification tasks. DensNet201 is a powerful model which can be used for transfer learning.

DenseNet201 uses dense connectivity of layers. In a dense network, each layer is connected to both the preceding and succeeding layers. The dense connectivity allows the model to extract robust features from the input.

DenseNet201:

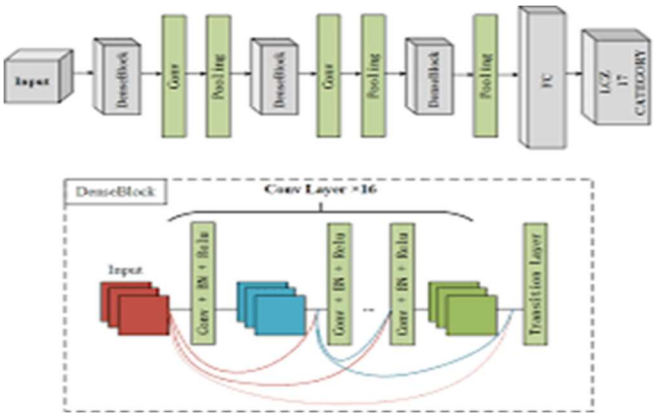


Fig 3.10 Representation of DenseNet201 Architecture (Courtesy: Source[27])

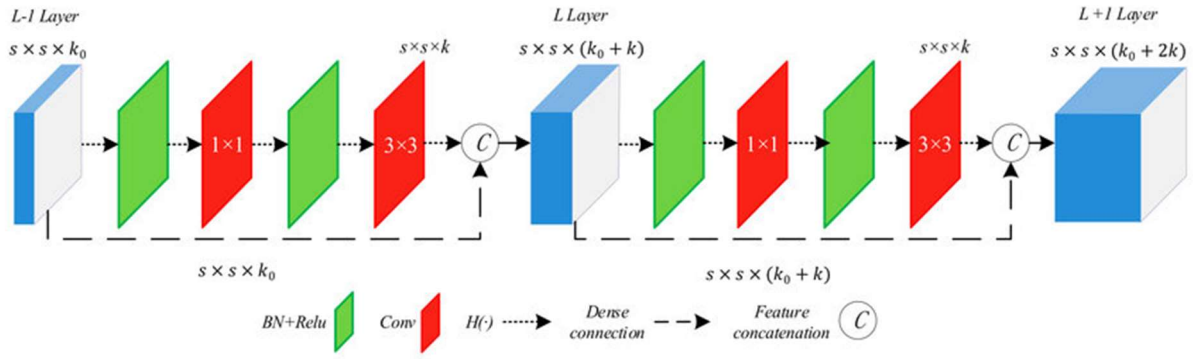


Fig 3.11 Representation of Dense Block (Courtesy: Source[28])

DenseNet consists of 201 layers, including 12 dense blocks. Here is the detailed description of DenseNet201 architecture

1. **Input Layer**

   The input layer is the crucial layer in the DenseNet model. This is the layer where input is taken into the network. The input can be an image of size 224x224. That input is passed to the succeeding layers. The input should in RGB format.

2. **Stem**

   Stem in DenseNet consists of two convolutional layers with 64 and 192 filters respectively. The first layer uses a kernel of size 7x7 and a stride of 2, whereas the second layer uses a kernel of size 3x3 and a stride of 1. It is responsible for extracting low-level features from the images.

3. **Dense Blocks**

   These are the core of the DenseNet architecture. Each dense block consists of a stack of convolutional layers which are densely connected. The network consists of 12 dense blocks. The number of convolutional layers in each dense block varies.

4. **Transition layers**

   They are responsible for reducing the dimensionality of feature maps generated by dense blocks. The transition layers use a combination of a convolutional layer and a max-pooling layer. Between each dense block, there is a transition layer. It is used to prevent the network from becoming too large and computationally expensive.

5. **Output Layer**

   The output layer has about 1000 nodes, one for each class in the ImageNet dataset. This layer is responsible for the classification of images.

Advantages of using DenseNet201

1. It can achieve state-of-the-art accuracy in image classification.
2. It is relatively efficient compared to other CNN architectures
3. It is robust to noise in the input data.
4. It can be used for transfer learning.

Disadvantages of using DenseNet201

1. It is expensive to train the model.
2. It has a very large number of parameters which makes it difficult to train and deploy the model.

3.3.1 Architecture Diagram

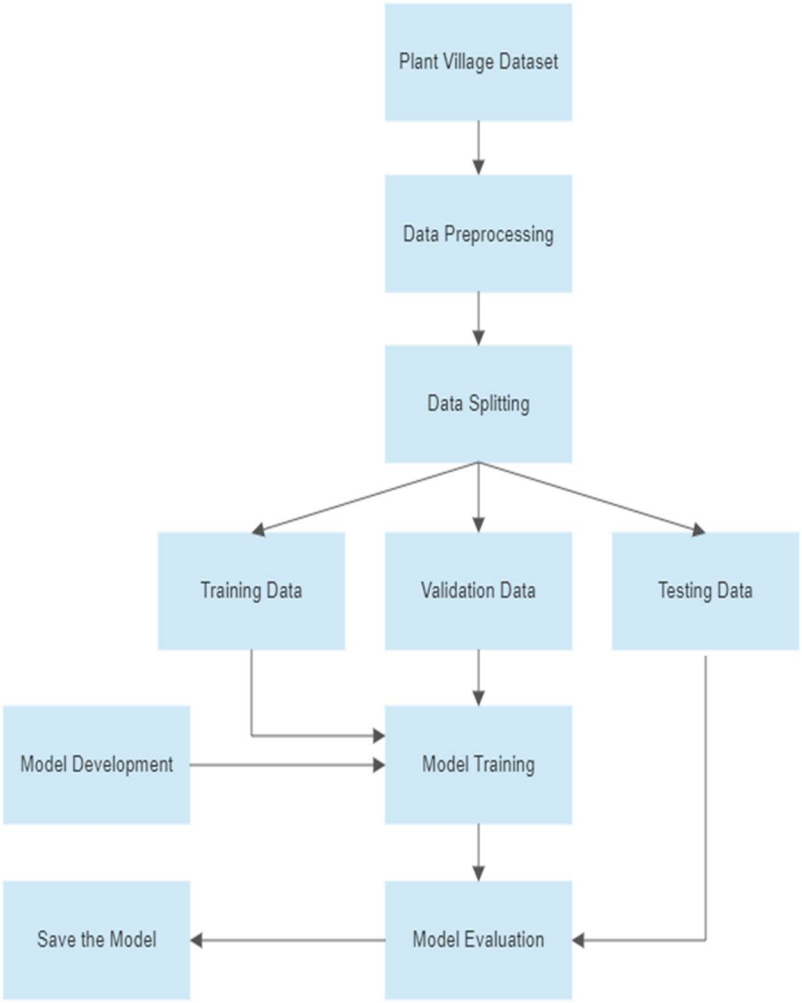


Fig 3.12 Architecture diagram of Building CNN model**.**

The Figure 3.12 represents the architecture diagram of building the CNN model. It shows the steps involved in building the CNN model. It also represents the flow in which data travels while building the model.

1. First, the dataset is pre-processed. The pre-processing includes a wide range of tasks like resizing images, converting images to tensors, and augmenting data using rotation, flipping, or zooming of images.
2. The pre-processed data is then split into 3 datasets- train, test, and validation data in the ratios 8:1:1 respectively.

3. The model is built on train data and validation data acts as the first test data which is used for overcoming the overfitting problem.
4. The model built is then evaluated using the test data and the evaluation metrics such as accuracy and loss are noted and the model is saved.
5. The model showing the best results is integrated with the web interface which utilizes this model to classify images.

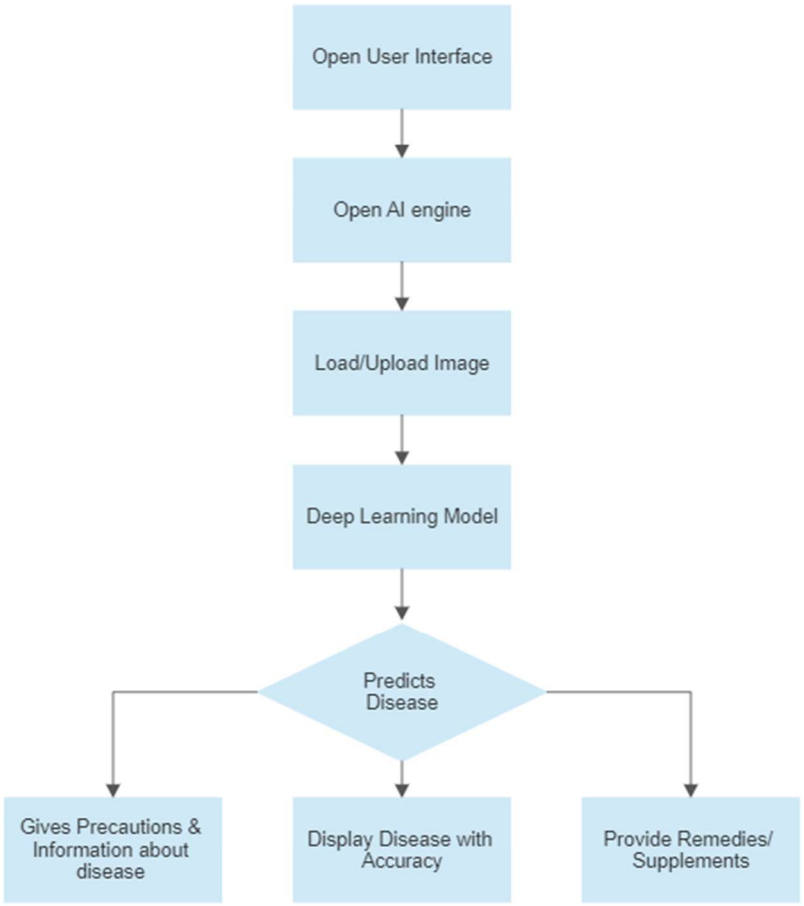


Fig 3.13 Architecture Diagram of User-Interface

represents the architecture diagram of user interface. It describes how user can interact with the user-interface and also the functions of the interface.

3.3.4 Modules and its Description

Module 1 Data Collection

The dataset used is from the Kaggle repository named “PlantVillage.” The dataset consists of 21,000 images of 3 different species named Tomato, Potato, and Bell Pepper and diseases of 15 categories.

Module 2 Data Pre-processing

Data pre-processing describes the process of preparing images for use in a Convolutional Neural Network (CNN) architecture. The first step is to resize the images so that they fit into the architecture's input layer. This resizing process involves adjusting the dimensions of the images while maintaining their aspect ratio. Once the images are resized, they are converted into tensors, which are multi-dimensional arrays that can be processed by the CNN.

Next, the tensor-formed images are further normalized. This normalization process involves scaling the pixel values of the images to a range that is suitable for the CNN's input layer. Normalization helps to improve the efficiency of the training process and ensures that the CNN can learn effectively from the data.

To prevent overfitting, which occurs when a model becomes too specialized to the training data and performs poorly on new data, data augmentation techniques are used. These techniques involve applying transformations such as rotation and flipping of the images. By applying these transformations, the model is exposed to a wider variety of images, and this helps to improve its ability to generalize to new data. In summary, preparing images for use in a CNN involves a series of steps, including resizing, conversion to tensors, normalization, and data augmentation.

Module 3 Data Splitting

Data splitting refers to dividing the dataset into different subsets for the development of the model. The dataset used here is split into three subsets- training data, testing data, and validation data of divisions of 80-10-10.

The training data is data used for the development of the model. The testing data is used for testing the accuracy of the model's predictions using measures like accuracy, f1-score, loss, etc.

Whereas the validation data is considered the first test in developing the model, it helps to prevent the overfitting problem. It evaluates the performance of the model in the training stage and allows for changes to be made to the model to improve its accuracy.

Module 4 Model Development

Model Development refers to the process of developing the model by designing its architecture, and selecting the optimizers to be used. Designing the model architecture is a critical stage in model development. It involves making decisions about the number of convolutional layers, pooling layers, dense layers, and activation functions used in the model. Additionally, selecting the type of optimization algorithm to be used and defining the input and output layers are also essential components of this process.

In this project, transfer learning is used as it takes to so much time to build and train the models. Transfer learning is used by leveraging the pre-trained models VGG16, VGG19, InceptionV1, and DenseNet201. The models are fine-tuned by removing the last layer of the pre-trained model and adding new layers to improve their performance. The weights used are based on the ImageNet dataset, which consists of 14 million images belonging to 1000 different classes. The description of these models is mentioned in section 3.3.2.

For VGG16, firstly, the input layer is added, and two sequential layers are added, the first layer for resizing and rescaling the images and the second layer for augmenting images. Later, VGG16 is added to the model followed by adding an average pooling layer, a dense layer, a

dropout layer, and finally the output layer is added to the model. Fig 3.14 represents the architecture of the model developed.

Fig 3.14 Architecture of Model built by using VGG16 network

For VGG19, firstly, the input layer is added to the model, followed by two sequential layers, the first layer for resizing and rescaling the images and the second layer for augmenting images. Later, the pre-trained VGG19 network is added followed by an average pooling layer, a dense layer, a normalization layer, a dropout layer, and finally output layer(i.e., a dense layer) is added to the model. Fig 3.15 represents the architecture of the model developed.

Fig 3.15 Architecture of Model built by using VGG19 network

For InceptionV3, firstly, the input layer is added to the model followed by two sequential layers, the first layer for resizing and rescaling the images and the second layer for augmenting images. Later, the InceptionV3 network is added to the model followed by an average pooling layer, a dense layer, a batch normalization layer, a dropout layer, and finally the output layer(ie., a dense layer) is added to the model. Fig 3.16 represents the architecture of the model developed.

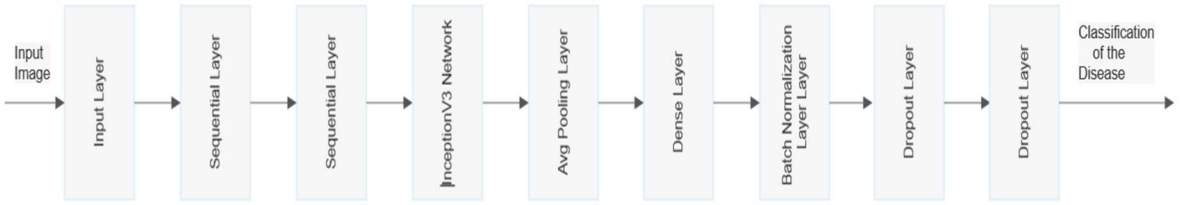


Fig 3.16 Architecture of Model built by using InceptionV3 network

For DenseNet201, firstly, the input layer is added to the model, followed by two sequential layers, the first layer for resizing and rescaling the images and the second layer for augmenting images. Later, the DenseNet201 network is added to the model followed by an average pooling layer, a dense layer, a batch normalization layer, a dropout layer, and finally the output layer(ie., a dense layer) is added to the model. Fig 3.17 represents the architecture of the model developed.

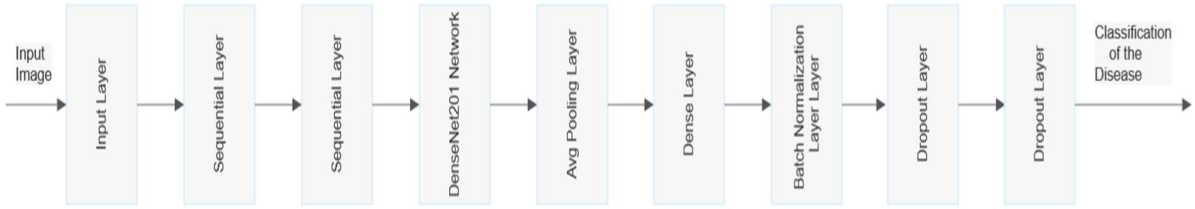


Fig 3.17 Architecture of Model built by using DenseNet201 network

Module 5 Model Training

Model training is the process of training the model developed with the help of training data. The model training process includes selecting the hyperparameters like batch size, and epochs. An epoch is a complete iteration through a dataset during the model training phase. It is the number of times the algorithm sees the entire dataset during the training process.

During an epoch, the model makes predictions on the training dataset, calculates the error or loss between the predicted and actual values, and updates the model's parameters based on the

gradient of the loss function concerning the model parameters. The goal is to minimize the loss function and improve the model's accuracy on the training data.

The number of epochs to train a model is a hyperparameter that must be tuned to achieve optimal performance. Setting the number of epochs too low may result in underfitting, while setting it too high may result in overfitting. Therefore, it is important to monitor the model's performance on a validation dataset and stop training when the validation loss stops improving, to prevent overfitting.

Batch size is a hyperparameter that must be set before the training of the model. It simply refers to the number of training examples used in a single iteration of the model during the training phase. During training, the model makes predictions on the input examples in a batch, calculates the loss between the predicted and actual values, and updates the model's parameters based on the gradient of the loss function for the model parameters.

The batch size can affect both the speed and quality of the training. The larger batch size can improve the training speed but also requires a lot of memory, whereas a smaller batch size can increase the generalization of the model and may prevent overfitting but may result in slow training and high variance in the loss function.

The epoch value taken while training the model is 50 and the batch size was taken as 32. The models are trained using the Adam optimizer, which is known for showing the start-of-the-art results.

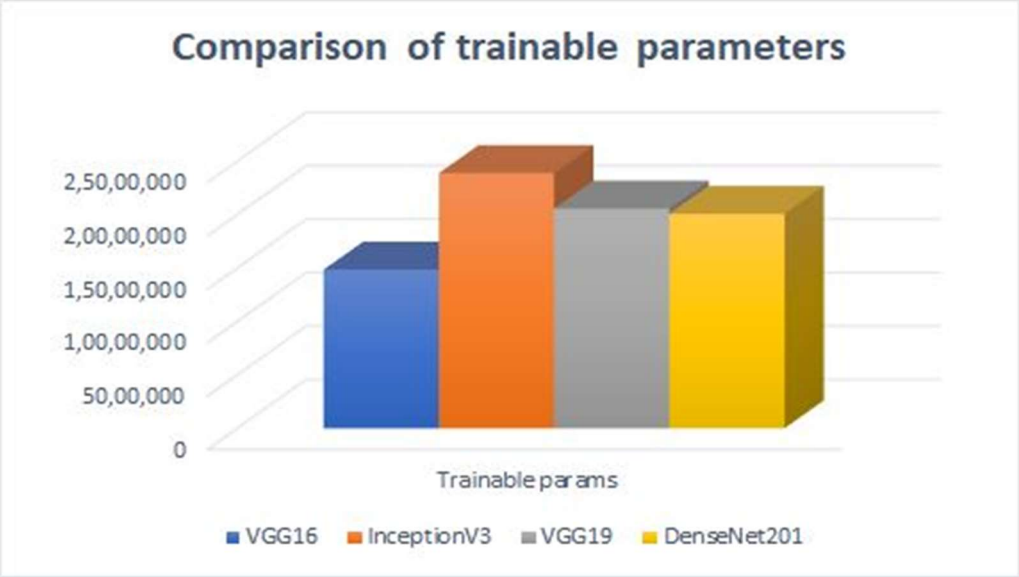


Fig 3.18 Comparison of trainable parameters for models used

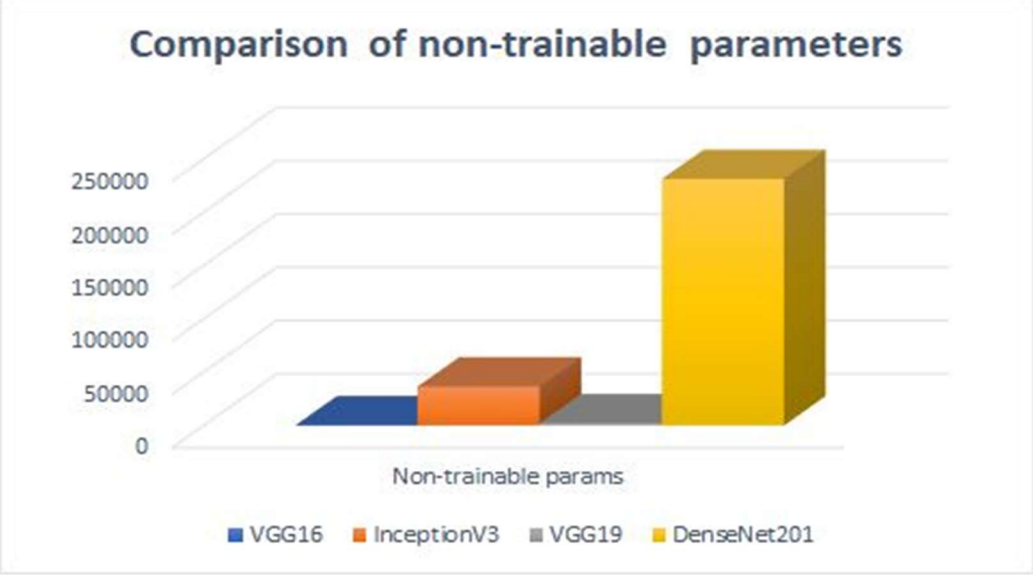


Fig 3.19 Comparison of non-trainable parameters for models used

Fig 3.18 and **Fig 3.19** show the comparison between the number of trainable and non-trainable parameters of the models used. Using the number of parameters we can compare the computation

time and complexity of the models. the model with more trainable parameters takes more time to train the model.

Module 6 Model Evaluation

Model evaluation is the phase where the generated model is evaluated for its performance on the predictions using the measures like accuracy, and loss. The model evaluation phase uses testing data to evaluate the model generated. The model with the most accuracy is selected.
Accuracy is a common evaluation metric used in the evaluation of the performance of a model. It is defined as the percentage of correct predictions made over the total number of predictions made.

accuracy = (TP + TN) / (TP + TN+ FP + FN)
TP-number of true positives
TN-number of true negatives
FP-number of false positives
FN-number of false negatives.

Loss is a commonly used metric for model evaluation. Loss refers to the variation in the predicted output and the actual output. However, the loss does not indicate the model's performance on unseen data. So may not be considered the best choice of using only loss as the evaluation metric. Our main objective is to minimize the loss of the model by using optimizers or by changing the architecture of the model.

Here, the evaluation of the model is done based on two evaluation metrics -accuracy and loss. The accuracy and loss are calculated for all three datasets- train, test, and validation.
Train accuracy represents the accuracy achieved by the trained model and similarly, train loss represents the loss achieved by the trained model.

Validation accuracy represents the accuracy achieved when the trained model is evaluated using the test data. The validation data is evaluated for every epoch value. Similarly, validation loss represents the loss acquired by the model when tested with validation data.

Train and validation evaluation metrics are updated for each epoch value. But only the last values are used for comparison. Test accuracy and loss represent the accuracy and loss acquired by the trained model when it is evaluated using the test data. The model is evaluated with test data only when it completes all the iterations defined by using epoch.

The accuracies and losses of all the datasets are compared and the best model had been for integrating with the website.

Module 7 User Interface

User Interface phase is the process of developing a user interface for integrating the model to use the model easily. The user interface here is developed using HTML, CSS, and frameworks like Bootstrap.

The user interface contains a section where the user can upload an image of a plant for classification. It also contains sections where the evaluation metrics such as accuracy are displayed and sections for displaying the preventive measures and management once the Image is classified.

Fig 3.20 Index of the user-interface

The Fig 3.20 shows the index page of the user interface. This page contains information about the application, how it works, and what type of plant diseases it can detect. When the "AI Engine" button is selected a new page opens where the user can upload the image of the plant to be detected as shown in the fig.3.21.

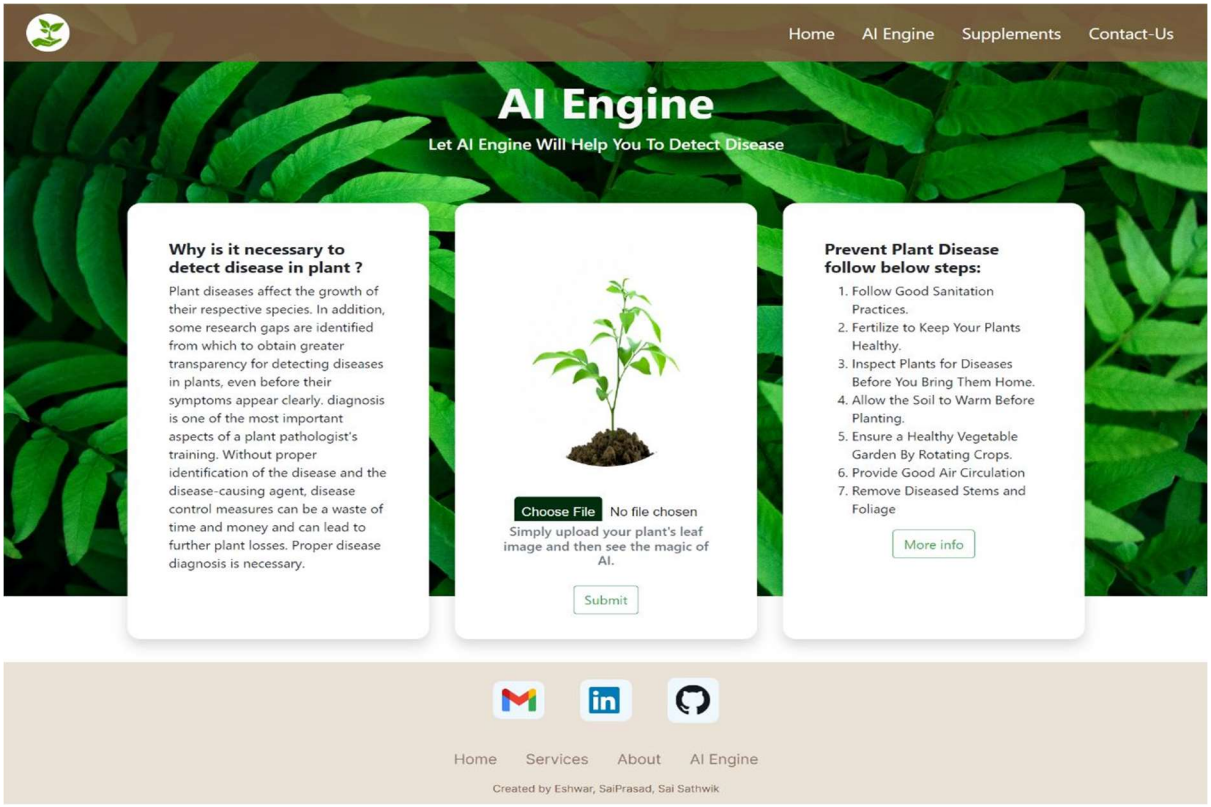


Fig 3.21 AI Engine

The user can upload the image in the AI engine and click on the "Submit" button to display results. The results are displayed as shown in Fig 3.23.

The interface can also be used to find and purchase the supplements used for agriculture. To see the supplements we just need to click the "Supplements" button in the navigation bar. This page shows all the supplements present as shown in Fig 3.22. Using this interface one can not purchase any product, but the page is redirected to the e-commerce website where the product chosen is available.

Module 8 Model Deployment

Model Deployment phase is the crucial phase, where the developed model is integrated with a web user interface using Flask API along with the information related to the diseases like preventive measures and management of specific diseases.

Here, the datasets of supplements_info and disease_info are integrated with the user interface using the Django framework. The suuplements_infro dataset contains all the details about supplements and where to find them and the disease_info dataset contains the details about the preventive measures of the diseases and also the description of the diseases.

When the model and datasets are integrated with the user interface, the interface can display the Supplements page by using the supplements_info dataset as shown in Fig 3.22 and the final results are displayed as shown in Fig 3.23.

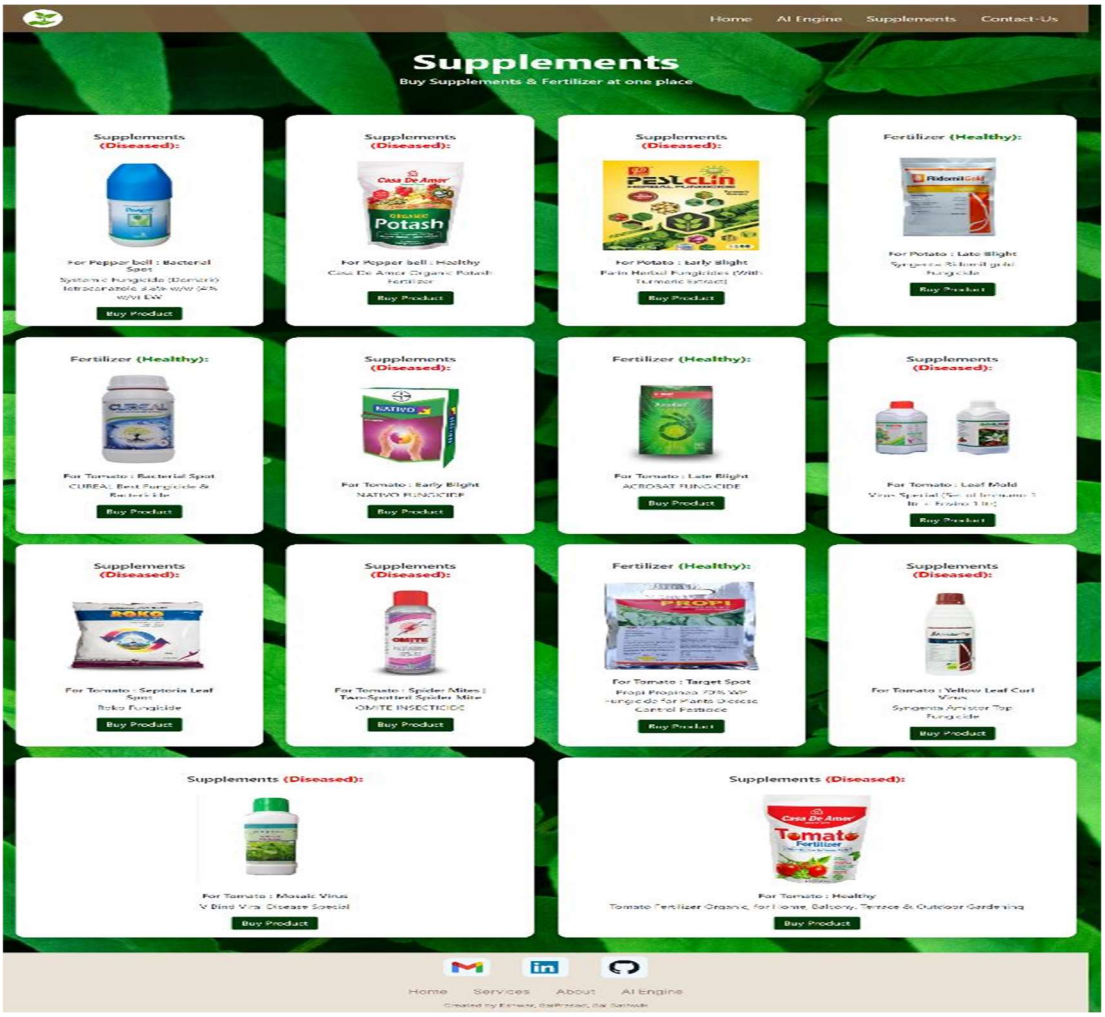


Fig 3.22 Supplements Page

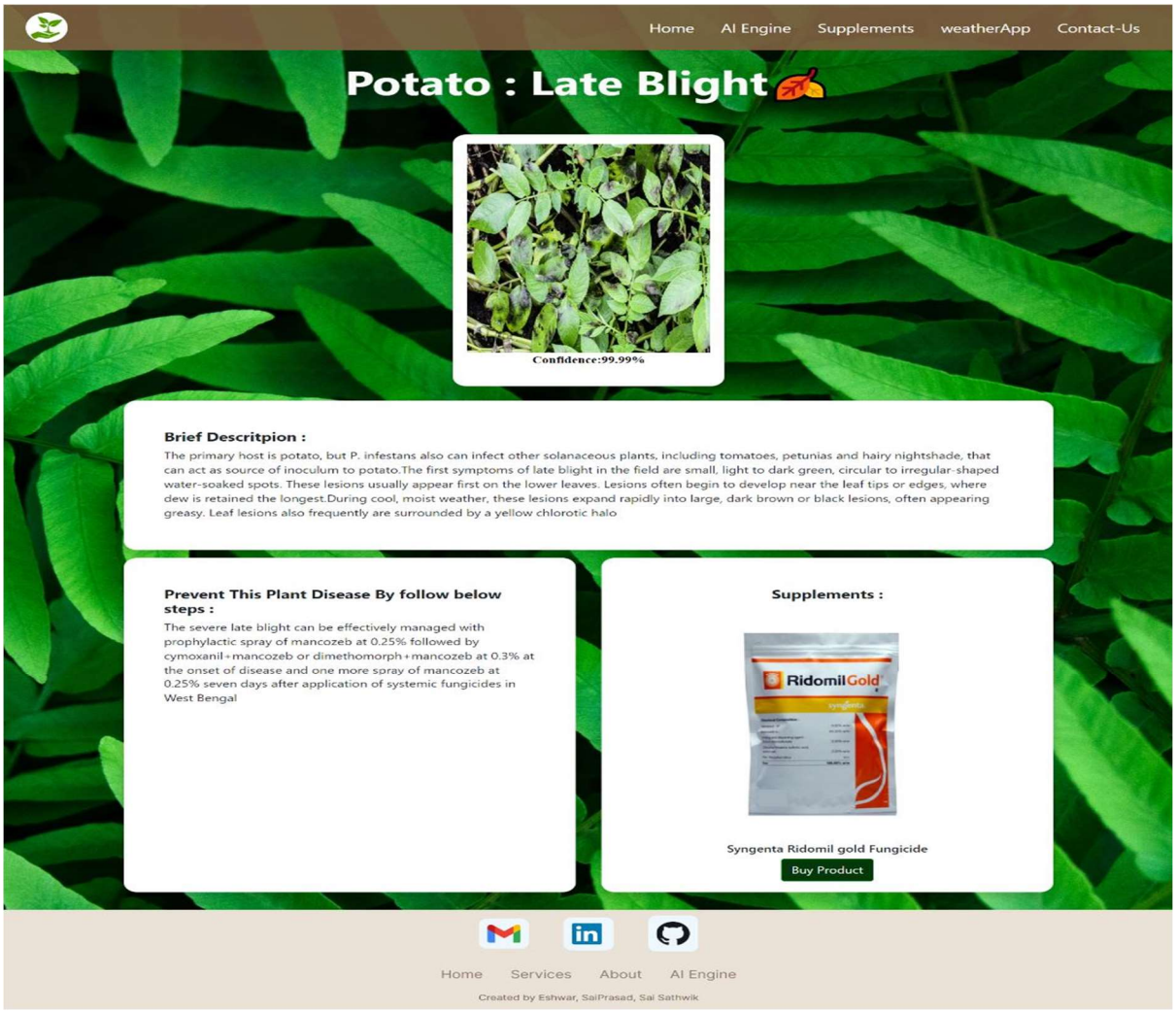


Fig 3.23 Results Page

When the submit button is clicked, the image is pre-processed and sent as the input to the model and the results are shown in Fig 3.23. The model is then updated by the input and the updated model is sent to Flask API and it replaces the updated model with the previous model. The description of the disease and preventive measures are taken from the disease_info based on the classification of the image and also the details about supplements are taken from the supplements_info dataset using the Django framework and are displayed as shown in fig 3.23.

When the supplements button is clicked the user interface leverages the supplements_info dataset using Django and displays results as shown in fig 3.22.

3.4 Requirements Engineering

3.4.1 Functional Requirements:

3.4.1.1 Uploading Image: The system should allow users to upload images.

3.4.1.2 Disease classification: The system should be able to classify plant diseases based on the uploaded images.

3.4.1.3 Display Results: The system should display the predicted disease, prevention measures, and supplements for the identified disease.

3.4.2 Non-Functional Requirements:

3.4.2.1 Performance: The system should provide fast and responsive disease classification within a reasonable time.

3.4.2.2 Usability: The system should have a user-friendly interface that is easy to navigate and understand.

3.4.2.3 Maintainability: The system should be designed in a modular and maintainable way to facilitate future enhancements or updates.

3.5 Analysis and Design through UML

3.5.1 Class Diagram

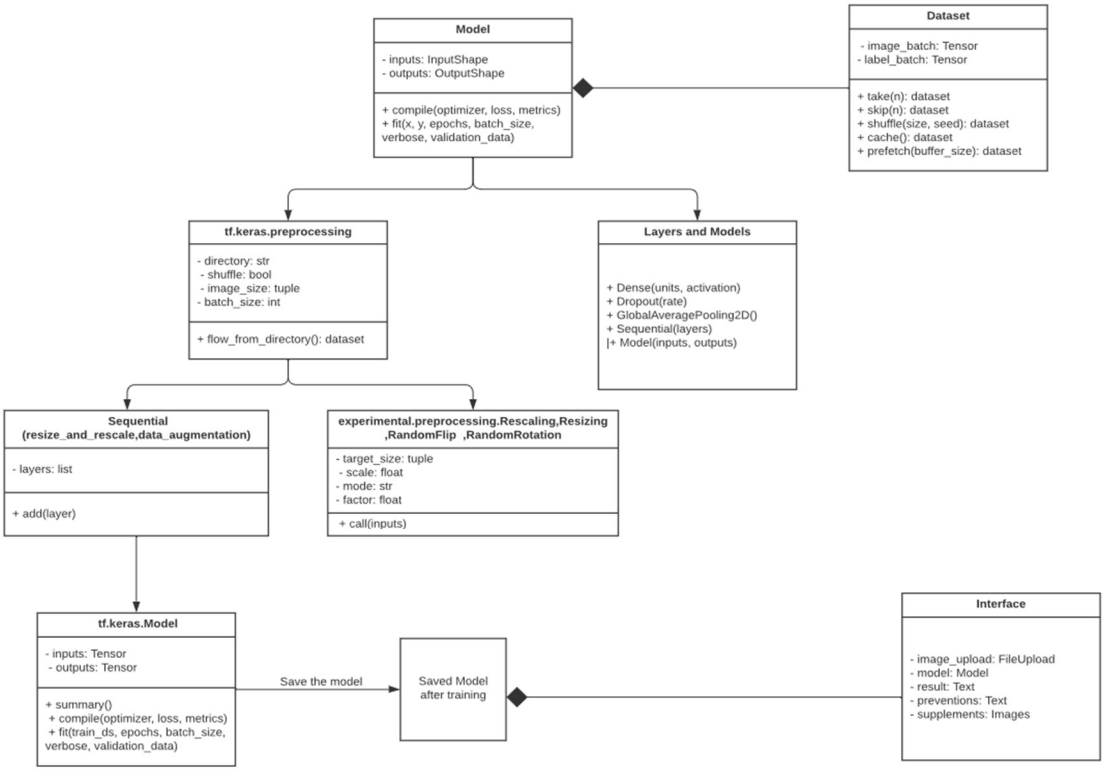


Fig 3.24 Class Diagram

3.5.2 Sequence Diagram

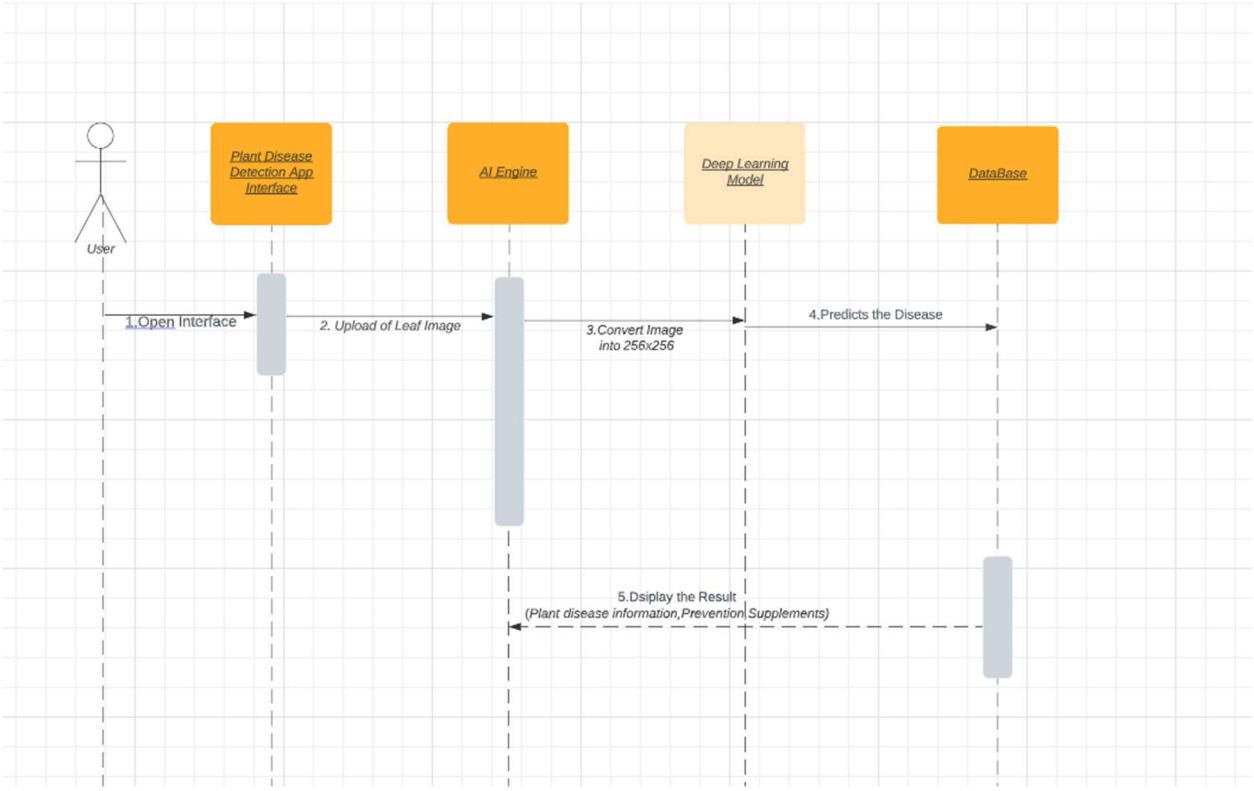


Fig 3.25 Sequence Diagram

The above sequence diagram represents the interactions and flow of control in the Plant Disease Detection App,

1. The user interacts with the User Interface (UI) of the app.
2. The user opens the AI Engine, where the user can upload the Plant leaf Image.
3. The UI sends image by clicking Upload button after uploading the image is converted into 256x256 image.
4. Then the image is passed through the CNN Pre-trained model. In this step the model predicts the disease and map the disease to the database where database contains disease description, preventive measures, and supplements.
5. The information sends to the user through the interface where it contains disease, confidence, disease description, preventive measures, and supplements.

3.5.3 Use Case Diagram

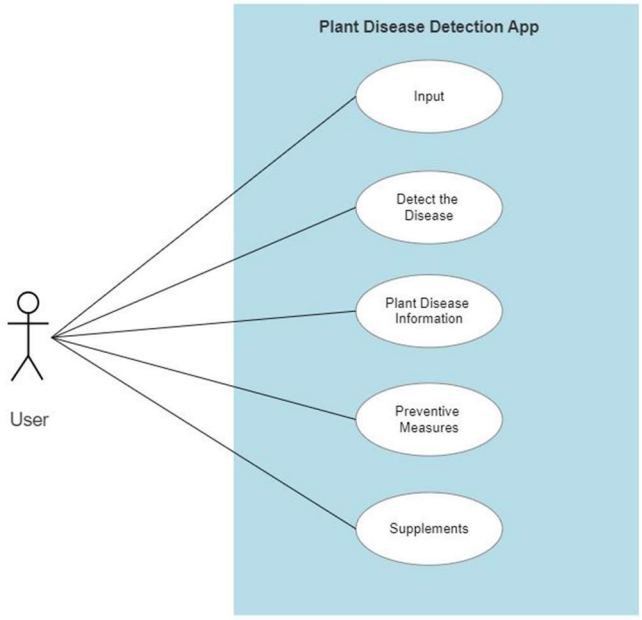


Fig 3.26 Use Case Diagram

The above figure Use Case Diagram represents the Image Deblurring App system. It outlines the various interactions between the users and the app through different use cases

1. **Plant Disease Detection App:** The main system being represented, which is the Plant Disease Detection App .
2. **User:** Represents the user or users interacting with the app.
3. **Use Cases:** Use cases represent the specific actions or tasks that users can perform within the app. The diagram includes the following use cases:
   a. **Upload the Image:** The user can upload an image within the app. This use case involves selecting a image from the user's device.
   b. **Detect the Disease:** The Pre-trained model can detect the image which is uploaded by the user.
   c. **Plant Disease Information:** The user can access the plant disease information which the model detected .on the user uploaded image.

d. **Preventive Measures:** The app displays Preventive measures to that particular disease.

e. **Supplements:** This app displays the Supplements on the user interface that can be used to treat the plant disease.

The Use Case diagram provides an overview of the different action's users can perform within the Plant disease Detection App and illustrates the interactions between users and the app through these use cases.

3.5.4 Activity Diagram

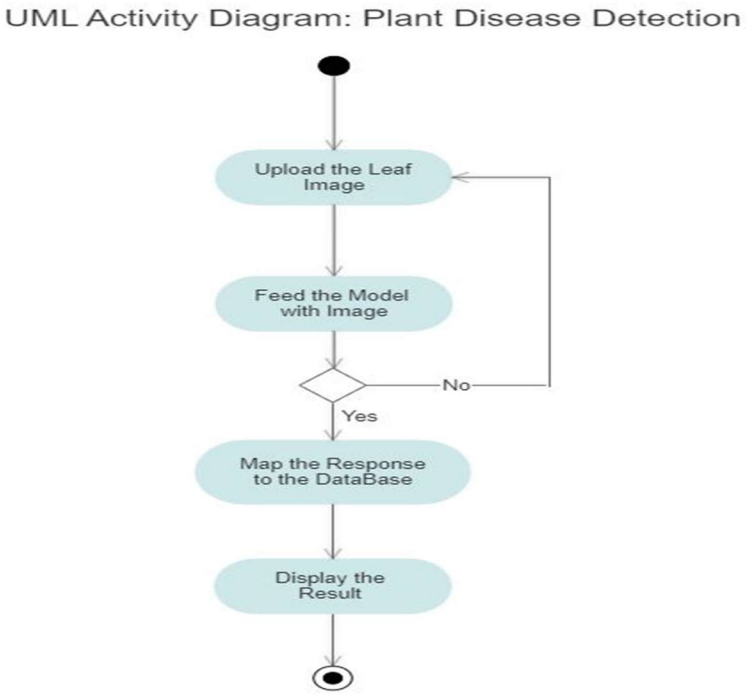


Fig 3.27 Activity Diagram

CHAPTER 4
RESULTS AND DISCUSSION

4.1 DESCRIPTION OF DATASET

The dataset used for building model is named as the Plant Village. It is sourced from the Kaggle Repository. It encompasses a diverse collection of 20,680 images, all uniformly sized at 256x256 pixels. These images are categorized into three plant species: Potato, Tomato, and Bell Pepper. The primary focus of this dataset revolves around the study of plant diseases, with images capturing various afflictions across 15 distinct categories.

Within the Tomato category, nine specific disease categories include Bacterial Spot, Early Blight, Late Blight, Leaf Mold, Septoria Leaf Spot, Spider Mites, Target Spot, Yellow Leaf Curl Virus, and Mosaic Virus. In addition to the range of tomato diseases, the dataset also comprises images of healthy tomato plants. Moving the Bell Pepper category consists of images showcasing the Bacterial Spot disease affecting the bell pepper and pictures of healthy bell pepper.

Furthermore, the dataset incorporates two disease categories of potatoes, namely Early Blight and Late Blight. Alongside these are the images of healthy potato plants, serving as a reference for disease-free samples.

Table 4.1 Data Description

Disease	Species	Number of Images
Bacterial Spot	Tomato	2127
Early Blight	Tomato	1000
Late Blight	Tomato	1909
Leaf Mold	Tomato	952
Septoria Leaf Spot	Tomato	1771
Spider Mites	Tomato	1676
Target Spot	Tomato	1404

Yellow Leaf Curl Virus	Tomato	3209
Mosaic Virus	Tomato	373
Healthy	Tomato	1591
Bacterial Spot	Bell Pepper	997
Healthy	Bell Pepper	1478
Early Blight	Potato	1000
Late Blight	Potato	1000
Healthy	Potato	152

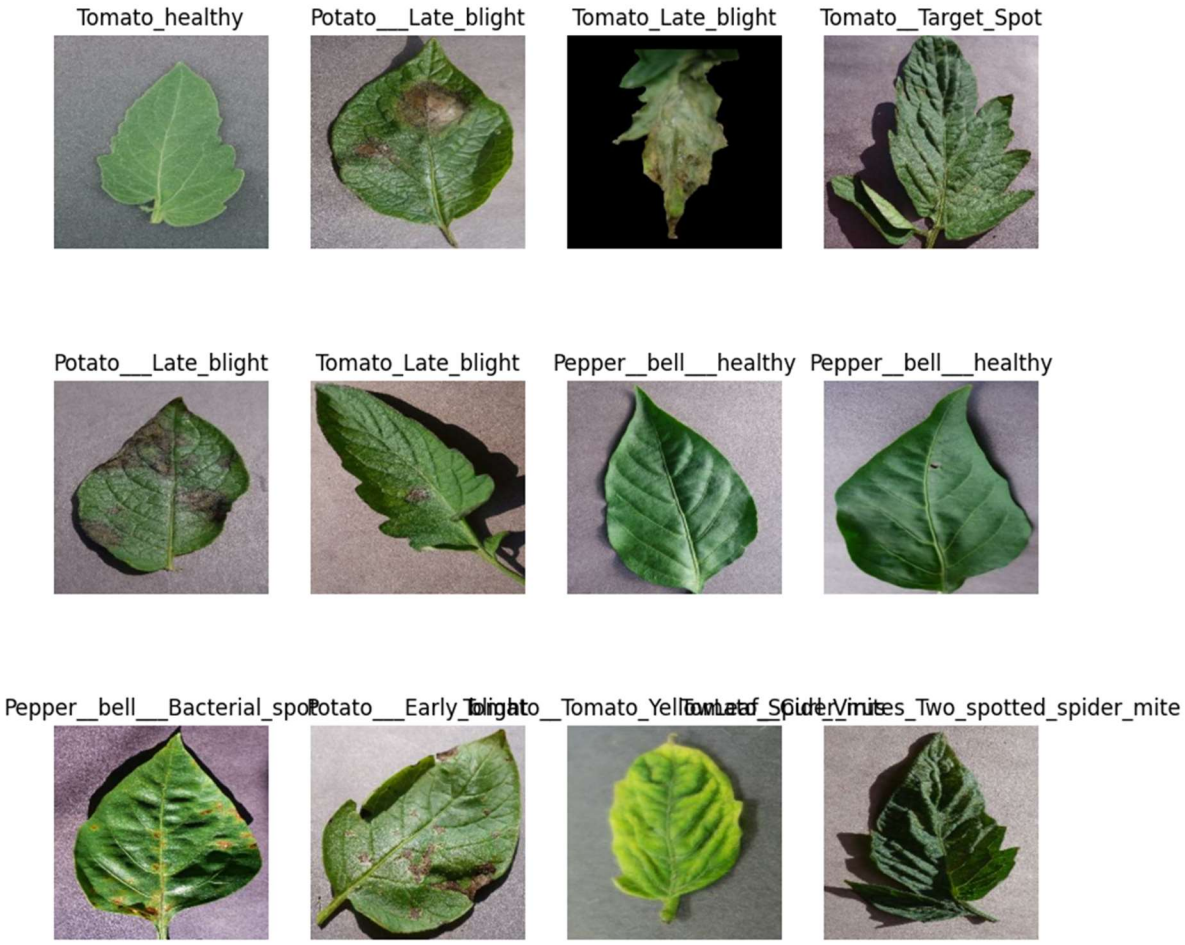


Fig 4.1 Some Images from Plant Village dataset

Apart from the Plant Village dataset, two datasets named supplements_info.csv and disease_info.csv are also used. The two datasets are not from any repositories but were created for this project.

The Supplements_info dataset contains information about the supplements and the links to purchase the supplements. The description of the supplements_info is shown below.

A	B	C	D	E
index	disease_name	supplement name	supplement image	buy link
1	Pepper,_bell___Bacterial_spot	Systemic Fungicide (Domark) T...	https://encrypted-tbn0.gstatic.com.	https://agribegri.com/products
2	Pepper,_bell___healthy	Casa De Amor Organic Potash	https://cdn.shopify.com/s/files/1/14	https://www.casagardenshop.
3	Potato___Early_blight	Parin Herbal Fungicides (With T...	https://encrypted-tbn3.gstatic.com.	https://agribegri.com/products
4	Potato___Late_blight	Syngenta Ridomil gold Fungi...	https://krushikendra.com/im...	https://krushikendra.com/Buy
5	Potato___healthy	Saosis Fertilizer for potato Fertili...	https://rukminim1.flixcart.com/im...	https://www.flipkart.com/saos
6	Tomato___Bacterial_spot	CUREAL Best Fungicide & Bac...	https://encrypted-tbn1.gstatic.co...	https://agribegri.com/products
7	Tomato___Early_blight	NATIVO FUNGICIDE	https://5.imimg.com/data5/SELL...	https://www.bighaat.com/proc
8	Tomato___Late_blight	ACROBAT FUNGICIDE	https://encrypted-tbn3.gstatic.c...	https://www.bighaat.com/proc
9	Tomato___Leaf_Mold	Virus Special (Set of Immuno 1	https://encrypted-tbn1.gstatic.co...	https://agribegri.com/products
10	Tomato___Septoria_leaf_spot	Roko Fungicide	https://cdn.shopify.com/s/files/1/07	https://www.bighaat.com/proc
11	Tomato___Spider_mites Two-sp...	OMITE INSECTICIDE	https://encrypted-tbn1.gstatic.com.	https://www.bighaat.com/proc
12	Tomato___Target_Spot	Propi Propineb 70% WP Fungi...	https://rukminim1.flixcart.com/im...	https://www.flipkart.com/propi
13	Tomato___Tomato_Yellow_Leaf_..	Syngenta Amistor Top Fungi...	https://krushikendra.com/im...	https://krushikendra.com/Buy
14	Tomato___Tomato_mosaic_virus	V Bind Viral Disease Special	https://encrypted-tbn1.gstatic.c...	https://agribegri.com/products
15	Tomato___healthy	Tomato Fertilizer Organic, for H...	https://cdn.shopify.com/s/files/1/...	https://www.casagardenshop.

Fig 4.2 Supplements_info.csv

The supplement_info.csv has four attributes namely disease name, supplement name, supplement image, and buy link. The other dataset used is diseases_info.csv. This dataset has the complete information of the diseases, their description, and their preventive measure. The below fig 4.2 represents the diseases_info.csv dataset.

A	B	C	D	E
index	disease_name	description	Possible Steps	image_url
1	Pepper bell : Bacterial Spot	Leaf spots that appear on the low..	Select resistant varieties Purcha...	https://backbonevalleynursery
2	Pepper bell : Healthy	Keep bell peppers well-wate...	Red, Orange, and Yellow Bell P...	https://www.healthbenefitstim
3	Potato : Early Blight	In most production areas, early	Treatment of early blight includ...	https://www.ag.ndsu.edu/pub
4	Potato : Late Blight	The primary host is potato, but ...	The severe late blight can be eff...	https://www.ag.ndsu.edu/pub
5	Potato : Healthy	Many potatoes need consistent	Packed With Nutrients. Shar...	https://www.garden.eco/wp-c
6	Tomato : Bacterial Spot	Bacterial spot of tomato is a pot...	Plant pathogen-free seed or tran...	https://www.growingproduce.
7	Tomato : Early Blight	Early blight is one of the most c...	Prune or stake plants to improv...	https://extension.umn.edu/site
8	Tomato : Late Blight	Late blight is caused by the oo...	Sanitation is the first step in cont...	https://www.canr.msu.edu/cor
9	Tomato : Leaf Mold	Tomato leaf mold is a fungal dis...	When treating tomato plants wit...	https://www.gardeningknowh
10	Tomato : Septoria Leaf Spot	Septoria leaf spot is caused by	Remove diseased leaves. Impr...	https://www.missouribotanica
11	Tomato : Spider Mites | Two-Spo...	The two-spotted spider mite is t...	Avoid weedy fields and do not p...	https://entomology.ca.uky.edu
12	Tomato : Target Spot	The disease starts on the older	Cultural control is important. T...	https://barmac.com.au/wp-co
13	Tomato : Yellow Leaf Curl Virus	Tomato yellow leaf curl virus is...	Use only virus-and whitefly-free	https://gd.eppo.int/media/data
14	Tomato : Mosaic Virus	Tomato mosaic virus (ToMV)...	Use certified disease-free seed	http://ucanr.edu/blogs/dirt/blo
15	Tomato : Healthy	Fertilize one week before as wel...	The tomato (Solanum lycopersic...	https://cdn8.dissolve.com/p...

Fig 4.3 Diseases_info.csv

The supplement_info.csv has four attributes namely disease_name,description, possible steps, and image url.

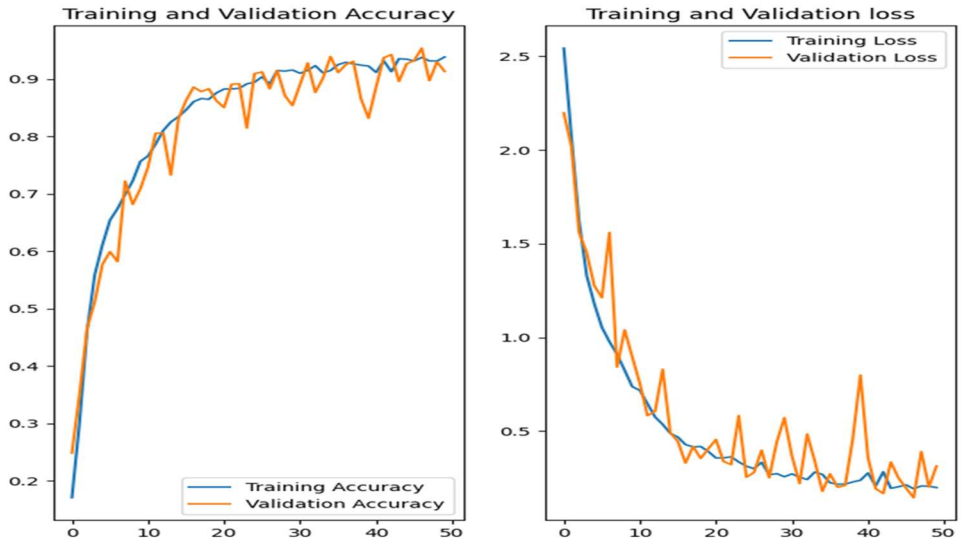
4.2 Detailed Explanation of Experimental Results

Four deep-learning convolutional neural networks (CNN) algorithms are used for detecting plant diseases. These algorithms include VGG16, VGG19, GoogleNet (InceptionV1), and DenseNet201. The performance of these models is calculated using accuracy and loss metrics to determine the best-performing algorithm.

The evaluation process involved monitoring the accuracy of each algorithm throughout the training epochs. Accuracy refers to the model's ability to classify plant disease images accurately. By observing how the accuracy of each algorithm evolves with each epoch value, important insights can be gained about their learning capabilities and performance trends.

The following graphs display the changes in accuracy for each algorithm across different epochs. These visualizations provide a comprehensive understanding of how the models' accuracy improves or fluctuates during the training. Analyzing these graphs will help identify patterns, trends, and potential areas for optimization in each algorithm's learning process.

VGG16



Accuracy vs Epochs Loss vs Epochs

Fig 4.4 Comparison of evaluation metrics over epochs for VGG16

This graph represents the relationship between the number of epochs and the Accuracy of a model during the training and testing process on a dataset. The x-axis indicates the number of epochs, which refers to the complete passes made by the model through the training data. The y-axis represents the Accuracy, which measures how effectively the model reduces the disparity between its predicted outputs and the actual targets.

This graph illustrates the relationship between the loss and the number of epochs during the training and testing of a model on a dataset. The x-axis represents the number of epochs, which corresponds to the number of complete passes through the training data during the training process. The y-axis represents the loss, which is a measure of how well the model is performing in terms of its ability to minimize the difference between the predicted outputs and the actual targets.

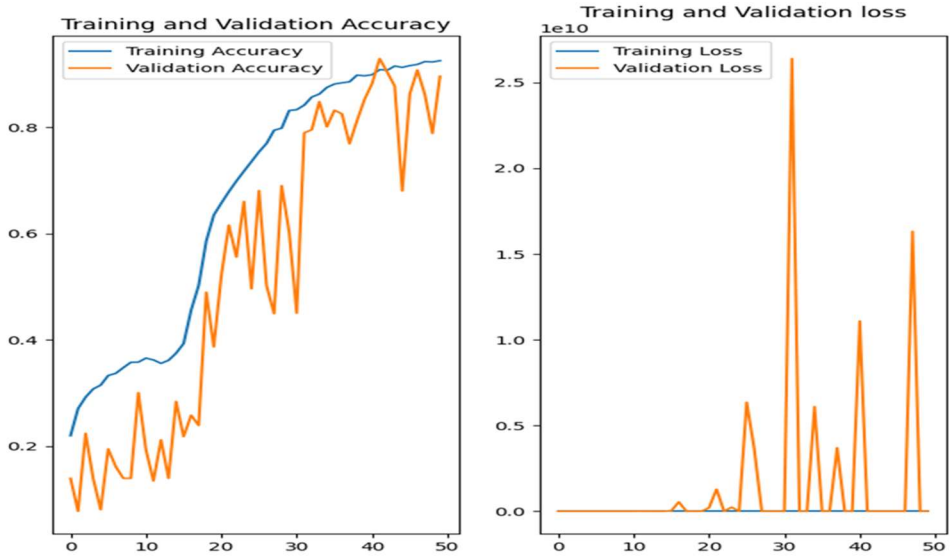
In this specific case, the training and testing of the model were conducted with a learning rate of 0.1. The learning rate is a hyperparameter that determines the step size taken by the optimization algorithm during each iteration. It influences the rate at which the model updates its parameters to minimize the difference between predicted outputs and actual targets.

The training process involved a total of 50 epochs, where each epoch represents a complete pass through the training data. In each epoch, the model performs a forward pass, where the input data propagates through the neural network, and a backward pass, where the model adjusts its parameters based on the computed gradients to optimize the loss function. The purpose of multiple epochs is to allow the model to learn progressively from the data and refine its predictions over time.

During training, a batch size of 32 was utilized. The batch size refers to the number of samples from the training data that are processed together in one iteration. In this case, during each epoch, the model processed the data in batches of 32. Batch processing offers computational efficiency and helps in generalizing the learned representations across the data by updating the parameters based on a mini-batch rather than individual samples.

Overall, with a learning rate of 0.1, training for 50 epochs, and a batch size of 32, the model underwent iterative updates and optimization to improve its performance in accurately predicting the targets based on the given dataset.

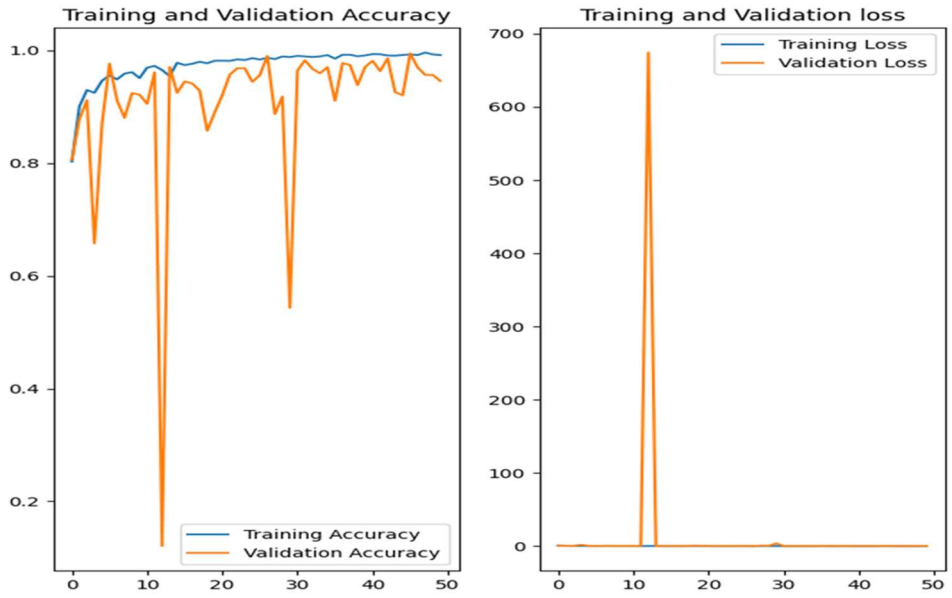
VGG19



Accuracy vs Epochs Loss vs Epochs

Fig 4.5 Comparison of evaluation metrics over epochs for VGG19

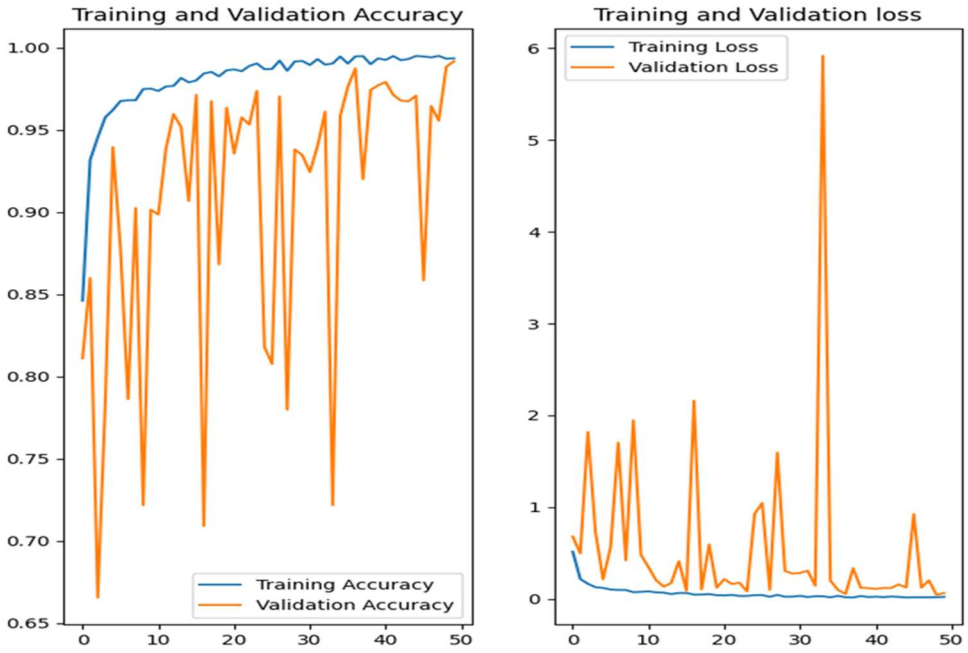
InceptionV3



Accuracy vs Epochs Loss vs Epochs

Fig 4.6 Comparison of evaluation metrics over epochs for InceptionV3

DenseNet201



Accuracy vs Epochs Loss vs Epochs

Fig 4.7 Comparison of evaluation metrics over epochs for DenseNet201

Table 4.2 Evaluation metrics of the models used

Model	**Optimizer**	**Accuracy**	**Loss**	**Validation Accuracy**	**Validation Loss**	**Test Accuracy**	**Test Loss**
VGG16	Adam	93.86%	26.01%	90.82%	27.97%	90.86%	27.54%

Google Net	Adam	99.18%	2.52%	94.58	24.47%	94.32%	26.57%
VGG19	Adam	92.53%	21.84%	89.60%	34.16%	36.49%	89.91%
DENSE NET201	Adam	99.35%	2.22%	99.17%	6.28%	98.82%	6.93%

The above table contains detailed information on the evaluation metrics such as the accuracy and loss of the models which are used.

By comparing the above results, it is clear that DenseNett201 has shown the maximum accuracy for all types of data. It has achieved an accuracy of 99.17% for the validation dataset and 98.82% for the test dataset.

4.3 Significance of the Proposed Work

Plant diseases are a major threat to food security and the agricultural industry. They can cause crop losses, which can lead to food shortages and economic hardship. Early detection and treatment of plant diseases is essential to prevent their spread and minimize damage.

Traditionally, plant diseases have been detected by visual inspection by trained experts. However, this approach is time-consuming and labor-intensive, and it can be difficult to scale up to large agricultural areas.

Machine learning algorithms can also be used to detect plant diseases more quickly and accurately than visual inspection. However, there are still limited online platforms available for

farmers to use these algorithms. This lack of infrastructure can make it difficult for farmers to access accurate information and make informed decisions about disease management.

This project uses a convolutional neural network (CNN) model to identify plant diseases from images. The CNN model was trained on a large dataset of images of healthy and diseased plants. The model has been shown to be accurate in identifying plant diseases, with a testing accuracy of 98.82%.

The project also provides a user-friendly interface that allows farmers to upload images of their plants and receive a diagnosis within seconds. The interface also provides farmers with information about the disease, supplements needed for treatment, and also preventive measures.

This project has the potential to revolutionize plant disease detection. It is a fast, accurate, and easy-to-use tool that can help farmers to protect their crops and ensure a sustainable food supply.

4.3.1 Here are some benefits of using this project

1. It can help farmers to save time and money.
2. It can help farmers to improve the quality of their crops.
3. It can help farmers to reduce the risk of crop loss.
4. It can help farmers to increase their yields.

Overall, this project is a valuable tool for farmers who are looking to protect their crops from disease. It is a fast, accurate, and easy-to-use tool that can help farmers to improve the quality and yield of their crops.

CHAPTER 5
CONCLUSION AND FUTURE ENHANCEMENTS

A major global worry is crop failure and how it affects the availability of food. Numerous obstacles face farmers today, such as the usage of harsh chemicals and climate change, both of which reduce yields. Deep learning in particular, a development in technology, offers encouraging methods to lessen these issues.

In the suggested study, authors created a deep learning model with an astonishing accuracy of 98.82% for identifying various plant diseases. Although this accomplishment is admirable, there are still opportunities for advancement and innovation.

To enhance the model's accuracy and real-world applicability, it is crucial to expand the dataset used for training. Including a diverse range of real-world images, encompassing different lighting conditions, backgrounds, and plant growth stages, would provide the model with a more comprehensive understanding of disease patterns. This expansion would help the model generalize better and improve its performance when exposed to various environmental conditions.

Furthermore, it would be advantageous to develop the model to not only detect diseases but also identify the stage and severity of a particular disease. By incorporating additional data and leveraging deep learning techniques, the model could learn to differentiate between early-stage infections and advanced disease progression. This information would enable farmers to take timely and targeted actions, potentially saving more crops and improving overall yield.

Additionally, integrating this technology into a surveillance system would further automate the disease detection process. By employing computer vision and deep learning algorithms, the system could continuously monitor large agricultural areas, detecting diseases in real-time. This proactive approach would enable farmers to respond swiftly, implementing precise interventions to prevent the spread of diseases and minimize crop losses.

Furthermore, the authors' development of a web application for accessing the disease detection features demonstrates the potential for user-friendly interfaces. Further advancements in user experience and accessibility can be explored, such as mobile applications or integration with existing farm management systems. Streamlining the process and making it readily available to farmers would encourage widespread adoption and maximize the positive impact of the technology.

CHAPTER 6
APPENDICES

```
import tensorflow as tf
from tensorflow.keras import models, layers
from tensorflow.keras.applications.vgg16 import VGG16
import matplotlib.pyplot as plt

IMAGE_SIZE= 256
BATCH_SIZE=32

dataset=tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/PlantDisease/PlantVillage",
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

class_names=dataset.class_names

class_names

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

for image_batch, label_batch in dataset.take(1):
    print(image_batch[0])

plt.figure(figsize=(10,10))
for image_batch, label_batch in dataset.take(2):
    for i in range(12):
        ax=plt.subplot(3,4,i+1)
```

```
        plt.imshow(image_batch[i].numpy().astype("uint8")) #convert into the integers as it is the float
        plt.title(class_names[label_batch[i]])
        plt.axis("off")


def                dataset_partitioning_tf(dataset,train_split=0.8,val_split=0.1,test_split=0.1,shuffle=True,
shuffle_size=1000):
    ds_size=len(dataset)

    if shuffle:
        dataset=dataset.shuffle(shuffle_size,seed=12)#seed is for just predictability same seed every time it
will give you same result

    train_size=int(ds_size*train_split)
    val_size=int(ds_size*val_split)
    test_size=int(ds_size*test_split)

    train_ds=dataset.take(train_size)
    rem_ds=dataset.skip(train_size)
    val_ds=rem_ds.take(val_size)
    test_ds=rem_ds.skip(val_size)

    return train_ds, val_ds, test_ds
train_ds, val_ds, test_ds=dataset_partitioning_tf(dataset)

print(len(train_ds))
print(len(val_ds))
print(len(test_ds))

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)
#this improve the preformance of pipeline as it stores the images in cache
#if we are using GPU or CPU if GPU is busy taining prefetch will load the next set of batch from your disk
```

that will improve the performance

```
#scaling the numpy array between 0-1 by dviding it by 255 (rgb)
resize_and_rescale=tf.keras.Sequential([
   layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
   layers.experimental.preprocessing.Rescaling(1.0/255)
])



# data augmentaion
data_augmentaion=tf.keras.Sequential([
   layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
   layers.experimental.preprocessing.RandomRotation(0.2)
])

CHANNELS = 3
input_shape=(IMAGE_SIZE,IMAGE_SIZE,CHANNELS)
n_classes=15
# base_model = VGG16(weights="imagenet", include_top=False, input_shape=input_shape)
# base_model.summary()
inputs = tf.keras.Input(shape=input_shape)
x = resize_and_rescale(inputs)
x = data_augmentaion(x)
base_model = VGG16(weights="imagenet", include_top=False, input_shape=input_shape)
x = base_model(x)
x = layers.GlobalAveragePooling2D()(x)

# Add additional layers on top of the base model
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(n_classes, activation='softmax')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()

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

EPOCHS=50
history = model.fit(
    train_ds,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    verbose=1,
    validation_data=val_ds



)
model_vesion=4
model.save(f"/content/drive/MyDrive/PlantDisease/models/{model_vesion}")

import numpy as np
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)
    return predicted_class, confidence
import matplotlib.pyplot as plt
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")


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()
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()
plt.title("Training and Validation loss")

import numpy as np
for images_batch, labels_batch in test_ds.take(1):
```

```
first_image = images_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])])
```

REFERENCES

[1] Rizqi Amaliatus Sholihati; Indra Adji Sulistijono; Anhar Risnumawan; Eny Kusumawati, "Potato Leaf Disease Classification Using Deep Learning Approach" 2020 International Electronics Symposium (IES), Surabaya, Indonesia, doi: 10.1109/IES50839.2020.9231784

[2] Umi Atilaa, Murat Uçarb, Kemal Akyolc, and Emine Uçar, "Plant leaf disease classification using EfficientNet deep learning model," Ecological Informatics, vol.61, Article 101182, March 2021, doi: 10.1016/j.ecoinf.2020.101182

[3] Latif, G.; Abdelhamid, S.E.; Mallouhy, R.E.; Alghazo, J.; Kazimi, Z.A. "Deep Learning Utilization in Agriculture: Detection of Rice Plant Diseases Using an Improved CNN Model". Plants 2022, 11, 2230, doi:10.3390/plants11172230

[4] Vimal Singha, Anuradha Chug, Amit Prakash Singh, "Classification of Beans Leaf Diseases using Fine Tuned CNN Model", International Conference on Machine Learning and Data Engineering, doi: 10.1016/j.procs.2023.01.017

[5] Rangarajan, A. K., Purushothaman, R., & Ramesh, A. (2018). Tomato crop disease classification using a pre-trained deep learning algorithm. Procedia Computer Science, 133, 1040–1047. doi:10.1016/j.procs.2018.07.070

[6] KP, A., & Anitha, J. (2021). Plant disease classification using deep learning. 2021 3rd International Conference on Signal Processing and Communication(ICPSC). doi:10.1109/icspc51351.2021.9451696

[7] Bhagat, M., Kumar, D., Mahmood, R., Pati, B., & Kumar, M. (2020). Bell Pepper Leaf Disease Classification Using CNN. 2nd International Conference on Data, Engineering, and Applications (IDEA). doi:10.1109/idea49133.2020.9170728

[8] Kundu, N., Rani, G., & Dhaka, V. S. (2020). A Comparative Analysis of Deep Learning Models Applied for Disease Classification in Bell Pepper. 2020 Sixth International Conference on Parallel, Distributed, and Grid Computing (PDGC). doi:10.1109/pdgc50313.2020.9315821

[9] Utkarsha N. Fulari, Rajveer K. Shastri, Anuj N. Fulari, "Leaf Disease Detection using Deep Learning" (2020), Journal of Seybold Report, ISSN NO: 1533-9211

[10] Ashok, S., Kishore, G., Rajesh, V., Suchitra, S., Sophia, S. G. G., & Pavithra, B. (2020). Tomato Leaf Disease Detection Using Deep Learning Techniques. 2020 5th International Conference on Communication and Electronics Systems (ICCES). doi:10.1109/icces48766.2020.9137986

[11] Tiwari, D., Ashish, M., Gangwar, N., Sharma, A., Patel, S., & Bhardwaj, S. (2020). Potato Leaf Diseases Detection Using Deep Learning. 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS). doi:10.1109/iciccs48265.2020.9121067

[12] V. K. Shrivastava, M. K. Pradhan, S. Minz, and M. P. Thakur, "RICE PLANT DISEASE CLASSIFICATION USING TRANSFER LEARNING OF DEEP CONVOLUTION NEURAL NETWORK",doi:10.5194/isprs-archives-XLII-3-W6-631-2019-corrigendum

[13] Binary Representation of a black and white image [Online]
Available:https://learnlearn.uk/binary/wp-content/uploads/sites/11/2017/01/step-1-binary-bitmap-bw.png

[14] Representation of colour image in RGB format [Online]
Available:https://savan77.github.io/assets/img/3channles.png

[15] Working of convolutional Layer [Online]
Available:https://vitalflux.com/wp-content/uploads/2022/04/Typical-CNN-architecture-768x250.png

[16] Representation of how the pooling layer works [Online]
Available: http://intellabs.github.io/RiverTrail/tutorial/images/convolution2.png

[17] Representation of fully connected layers [Online]
Available:https://builtin.com/sites/www.builtin.com/files/styles/ckeditor_optimize/public/inline-images/3_fully-connected-layer_0.jpg

[18] TensorFlow [Online]
Available: https://i.ytimg.com/vi/yjprpOoH5c8/maxresdefault.jpg

[19] Keras [Online]
Available: https://keras.io/img/logo.png

[20] NumPy [Online]
Available: https://numpy.org/images/twitter-image.jpg

[21] Bootstrap [Online]
Available:https://upload.wikimedia.org/wikipedia/commons/thumb/b/b2/Bootstrap_logo.svg/1200px-Bootstrap_logo.svg.png

[22] Flask API [Online]
Available:https://nordicapis.com/wp-content/uploads/How-to-Create-an-API-Using-The-Flask-Framework.png

[23] Representation of VGG16 Architecture [Online]
Available: https://miro.medium.com/v2/resize:fit:1100/0*6VP81rFoLWp10FcG

[24] Representation of VGG19 Architecture [Online]
Available:https://www.researchgate.net/publication/359771670/figure/fig5/AS:11431281079634597@1660789329088/VGG-19-Architecture-39-VGG-19-has-16-convolution-layers-grouped-into-5-blocks-After.png

[25] Representation of InceptionV3 Architecture [Online]
Available:https://www.researchgate.net/publication/349717475/figure/fig5/AS:996933934014464@1614698980419/The-architecture-of-Inception-V3-model.ppm

[26] Representation of Inception Block [Online]
Available:https://www.researchgate.net/profile/Anabia-Sohail-2/publication/330511306/figure/fig4/AS:717351204958208@1548041263007/Basic-architecture-of-inception-block.ppm

[27] Representation of DenseNet201 Architecture [Online]
Available:https://www.researchgate.net/profile/Hemant-Kumar-84/publication/354450739/figure/fig3/AS:1068321533349890@1631719110936/DenseNet-201-Architecture.ppm

[28] Representation of Dense Block [Online]
Available:https://www.researchgate.net/publication/337642659/figure/fig1/AS:830749703950336@1575077572918/Structure-of-dense-block-There-are-two-dense-connections-in-this-dense-block-BN.jpg