PLANT DISEASE DETECTION SYSTEM FOR SUSTAINABLE AGRICULTURE

A Project Report

submitted in partial fulfillment of the requirements

of

AICTE Internship on AI: Transformative Learning
with
TechSaksham – A joint CSR initiative of Microsoft & SAP

by

Gursirat Kaur, siratjawandha.004@gmail.com

Under the Guidance of

P.Raja, Master Trainer Edunet Foundation

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Mr. P. Raja, Master Trainer at Edunet Foundation, for his invaluable guidance and support throughout the development of the Plant Disease Detection System for Sustainable Agriculture project. His expertise, dedication, and constructive feedback were instrumental in shaping the direction of this project.

I also appreciate the opportunity provided by the AICTE Internship Program, which enabled me to work under his mentorship and learn about cutting-edge technologies in the field of agriculture and machine learning. Mr. P. Raja's insights not only helped in overcoming technical challenges but also inspired me to think innovatively and approach problems systematically.

I am deeply grateful for his time, effort, and encouragement, which greatly contributed to the successful completion of this project.

ABSTRACT

The increasing prevalence of plant diseases poses a significant threat to agricultural productivity, food security, and environmental sustainability. Traditional methods of disease detection are often time-consuming, labor-intensive, and prone to errors, leading to delayed interventions and excessive use of chemical pesticides. To address these challenges, this project proposes a Plant Disease Detection System (PDDS) to enable early, accurate, and efficient disease identification and management, contributing to sustainable agriculture.

The primary objective of the system is to assist farmers in detecting diseases promptly and taking informed actions to minimize crop losses. The methodology integrates advanced technologies such as Convolutional Neural Networks (CNNs) for image-based disease identification and Internet of Things (IoT) devices for real-time environmental monitoring. CNNs analyze plant leaf images to classify diseases with high accuracy, while IoT sensors collect data on environmental factors like temperature, humidity, and soil conditions to predict disease risks.

Key results demonstrate that the system can accurately detect diseases like blight, rust, and mildew, enabling timely and targeted interventions. The inclusion of a mobile application ensures accessibility by providing farmers with instant disease diagnosis, treatment recommendations, and preventive measures. By reducing reliance on chemical pesticides and optimizing resource usage, the PDDS promotes sustainable farming practices.

In conclusion, the Plant Disease Detection System empowers farmers with real-time, actionable insights, reducing crop losses and supporting eco-friendly agriculture. This project highlights the potential of integrating artificial intelligence and IoT in addressing critical challenges in agriculture, fostering a resilient and sustainable food production ecosystem.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER1

Introduction

1.1Problem Statement:

The problem also lies in difficulty of identifying plant diseases early, which often leads to overuse of chemicals and inefficient farming practices. To solve this, there is a need for an intelligent and automated system that uses computer vision and machine learning to detect diseases in real-time. This system would help farmers take quick and targeted actions, promoting sustainable farming by reducing chemical usage and protecting crops effectively.

In early agricultural civilizations, farmers relied on their observation skills to detect plant diseases. They learned to recognize visual symptoms such as wilting, discoloration, lesions, and abnormal growth patterns in plants. Farmers shared this knowledge orally from one generation to another.

With the emergence of artificial intelligence (AI) and machine learning (ML), automated and intelligent systems for plant disease detection have been developed. These systems use algorithms to analyze large datasets, including images, to identify and classify plant diseases accurately.

Now we will discuss about the basics that we need to go ahead with this project.

1.1.1 BASICS OF MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Intelligence is the ability to learn, understand and apply knowledge and skills. It involves reasoning, problem solving and adapting to new situations.
Intelligence includes various cognitive abilities:

1. Learning
   - The ability to acquire, retain, and apply knowledge or skills over time.

2. Reasoning
   - The capacity to think logically, analyze information, and make informed decisions.

3. Problem-Solving
   - The ability to identify challenges and develop effective solutions.

4. Adaptability
   - The flexibility to adjust behavior and thinking in response to changing environments or new information.

5. Creativity
   - The ability to generate novel and useful ideas, solutions, or approaches.

6. Emotional Intelligence

- The capacity to recognize, understand, and manage one’s own emotions, as well as empathize with and influence others.

7. Abstract Thinking

- The ability to understand complex concepts and relationships that go beyond concrete experiences.

8. Social Intelligence

- The skill to interact effectively with others, navigate social situations, and build meaningful relationships.

1.1.2 FIRST STEP TOWARDS AI BACK IN 1940s:

Wartime Contributions (1939–1945): Code Breaking during the second world war.

- Context: During World War II, Turing worked for the British government at the center for Allied code breaking efforts.
- Primary Achievement:
  - Turing developed techniques to crack the German Enigma machine, a sophisticated encryption device used by Nazi forces to encode military communications.
  - He played a critical role in designing the Bombe machine, an electromechanical device that significantly automated the decryption of Enigma messages.

What actually is the Turing test?

The Turing Test is a measure proposed by a British mathematician and computer scientist Alan Turing in 1950 to determine a machine's ability to exhibit intelligent behavior indistinguishable from that of a human.

It was introduced in his seminal paper titled **"Computing Machinery and Intelligence"** published in the journal *Mind*.

The test involves a human evaluator engaging in natural language conversations with a machine and another human through a text-based interface.

The evaluator doesn't know which participant is the machine and which is the human.

- If the evaluator **cannot reliably distinguish** the machine from the human based on their responses, the machine is said to have passed the Turing Test.

- The test focuses on **behavior** rather than internal thought processes, meaning it doesn't require the machine to "think" like a human, only to behave like one in conversation.

Key Elements of the Turing Test

1. Human Interaction: The machine must communicate with humans naturally, often via text.
2. Indistinguishability: The evaluator must not distinguish the machine's responses from a human.
3. No Task Restriction: The conversation can cover any topic.

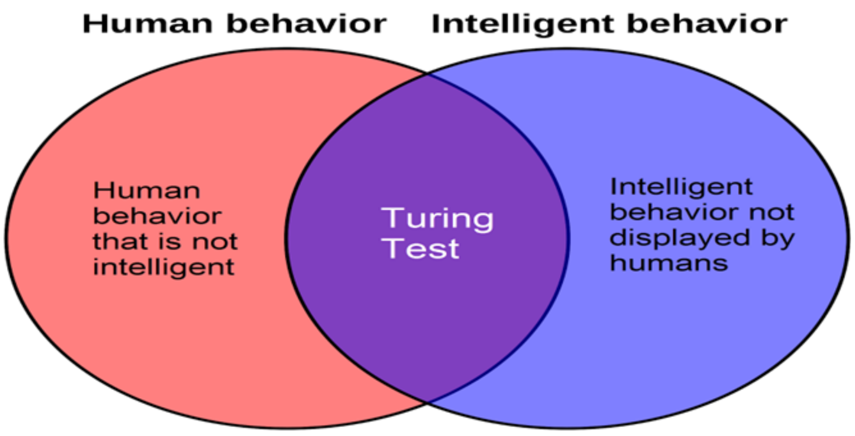

Human behavior
Intelligent behavior
Human behavior that is not intelligent
Turing Test
Intelligent behavior not displayed by humans

Figure1

1.1.3 WHAT IS ARTIFICIAL INTELLIGENCE?

Artificial Intelligence (AI) refers to the field of computer science focused on creating systems or machines that can perform tasks typically requiring human intelligence. These tasks include learning, reasoning, problem-solving, understanding natural language, perception, and decision-making.

Categories of artificial intelligence

#Types of AI Based on Capability

1. Narrow AI (Weak AI)
   a. Definition: AI systems designed to perform a single specific task with proficiency.
   b. Examples:
      i. Virtual assistants (e.g., Siri, Alexa).
      ii. Recommendation systems (e.g., Netflix, Amazon).
      iii. Image recognition software.
   c. Characteristics:

        i. Task-specific.
        ii. Lacks generalization and human-like adaptability.

2. General AI (Strong AI)
    a. Definition: Hypothetical AI that possesses the ability to perform any intellectual task a human can do.
    b. Examples: No real-world implementations yet.
    c. Characteristics:
        i. Can think, reason, and learn like a human.
        ii. Able to apply knowledge across diverse domains.

3. Superintelligent AI
    a. Definition: A theoretical AI that surpasses human intelligence in virtually all fields, including creativity, decision-making, and problem-solving.
    b. Examples: Only a concept for now; often discussed in speculative or futuristic contexts.
    c. Characteristics:
        i. Far exceeds human capabilities.
        ii. Potential to revolutionize or pose risks to humanity (e.g., ethical concerns).

#Types of AI Based on Functionality

1. Reactive Machines
    a. Definition: AI systems that react to current scenarios without relying on past experiences or memory.
    b. Examples:
        i. IBM’s Deep Blue, the chess-playing AI.
    c. Characteristics:
        i. No memory or learning capability.
        ii. Performs tasks based on real-time input.

2. Limited Memory
    a. Definition: AI systems that can use past data to inform current decisions and improve performance.
    b. Examples:
        i. Autonomous vehicles using sensor data for navigation.
    c. Characteristics:
        i. Retains short-term memory for specific tasks.
        ii. Most modern AI systems fall into this category.

3. Theory of Mind
    a. Definition: AI capable of understanding emotions, beliefs, and intentions, and interacting socially.

b. Examples: Still in research phases, with developments in emotionally intelligent systems.
c. Characteristics:
    i. Can interpret human emotions and respond accordingly.
    ii. Crucial for advanced human-AI interaction.

4. Self-Aware AI
    a. Definition: Theoretical AI with self-consciousness and awareness of its own existence.
    b. Examples: Does not currently exist; considered a distant goal of AI research.
    c. Characteristics:
        i. Possesses self-awareness and subjective understanding.

RELATIONSHIP BETWEEN AI VS ML VS DL:

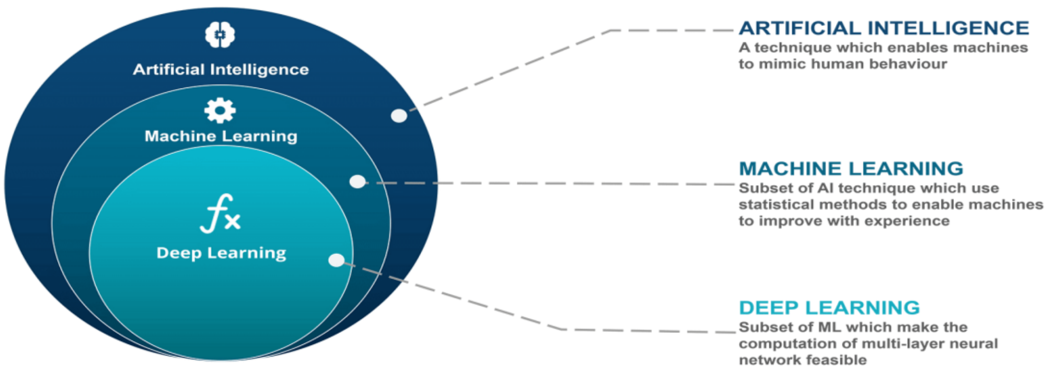

Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Figure 2

APPLICATIONS OF ARTIFICIAL INTELLIGENCE

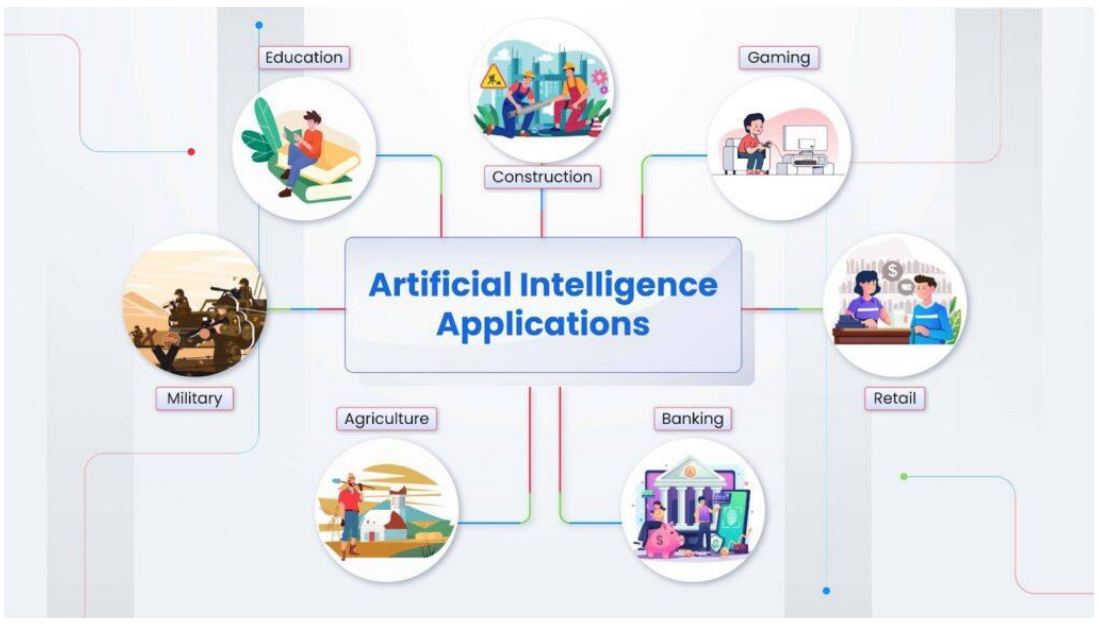


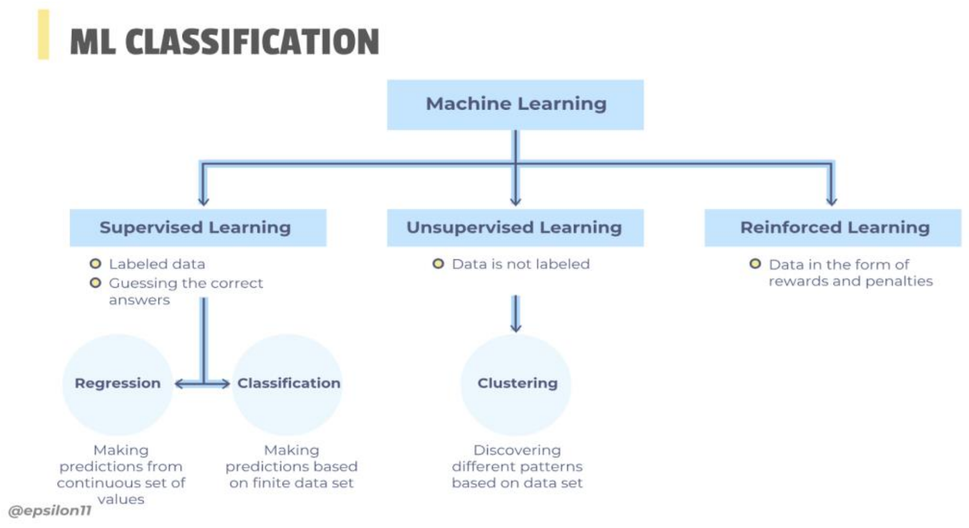
Figure 3

1.1.4 MACHINE LEARNING

Machine Learning (ML) is a branch of artificial intelligence (AI) that focuses on building systems capable of learning and improving their performance automatically from data without being explicitly programmed for every specific task.

TYPES OF MACHINE LEARNING

Figure 4


ML CLASSIFICATION
Machine Learning
Supervised Learning
Unsupervised Learning
Reinforced Learning
Labeled data
Guessing the correct answers
Data is not labeled
Data in the form of rewards and penalties
Regression
Classification
Clustering
Making predictions from continuous set of values
Making predictions based on finite data set
Discovering different patterns based on data set
@epsilon11

1.SUPERVISED LEARNING:

Supervised learning is defined as when a model gets trained on a “Labelled dataset”. Labelled datasets have both input and output parameters. Supervised learning algorithms learn to map points between inputs and correct outputs. It has both training and validation datasets labelled.

Let’s understand it with the help of an example.

Example: Consider a scenario where you have to build an image classifier to differentiate between cats and dogs. If you feed the datasets of dogs and cats labelled images to the algorithm, the machine will learn to classify between a dog or a cat from these labeled images. When we input new dog or cat images that it has never seen before, it will use the learned algorithms and predict whether it is a dog or a cat. This is how supervised learning works, and this is particularly an image classification.

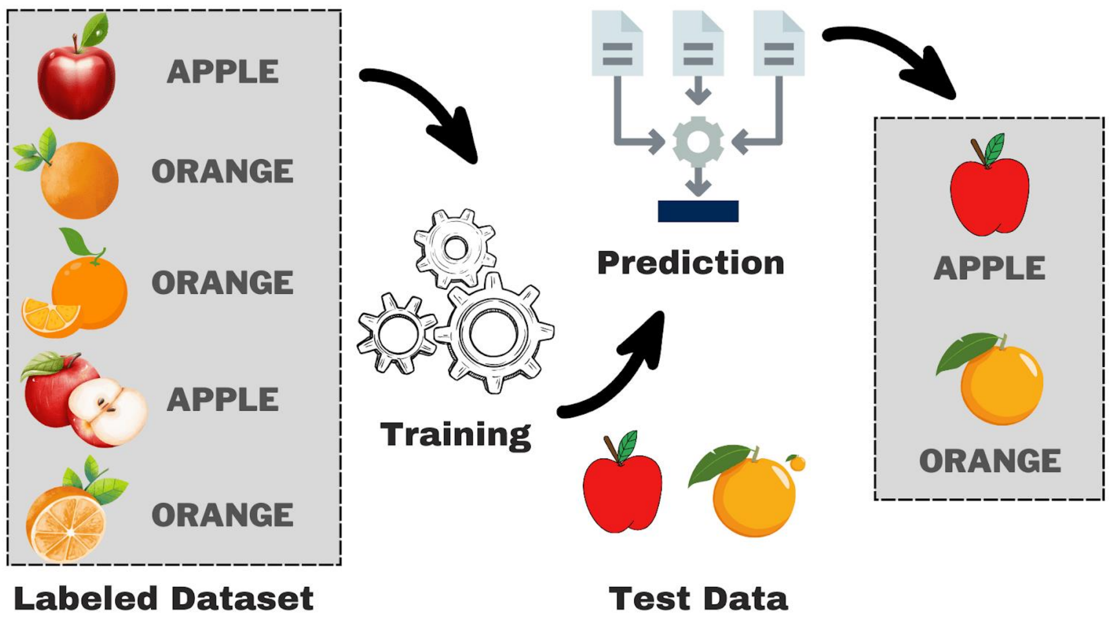


Figure 5

There are two main categories of supervised learning:

1. Classification
2. Regression

a. Classification
Definition: Classification is a task where the goal is to predict a discrete label or category for an input based on its features.

Key Characteristics:

- Outputs are categorical or discrete values.
- Answers questions like "Which category does this belong to?" or "Is this yes or no?"

b. Regression
Definition: Regression is a task where the goal is to predict a continuous numerical value based on input features.
Key Characteristics:

- Outputs are continuous or real-valued numbers.
- Answer questions like "How much?" or "What is the value?"

Applications of Supervised Learning
Supervised learning is used in a wide variety of applications, including:

- Image classification: Identify objects, faces, and other features in images.
- Natural language processing: Extract information from text, such as sentiment, entities, and relationships.
- Speech recognition: Convert spoken language into text.
- Recommendation systems: Make personalized recommendations to users.
- Predictive analytics: Predict outcomes, such as sales, customer churn, and stock prices.
- Medical diagnosis: Detects diseases and other medical conditions.
- Fraud detection: Identify fraudulent transactions.
- Autonomous vehicles: Recognize and respond to objects in the environment.
- Email spam detection: Classify emails as spam or not spam.

2. UNSUPERVISED LEARNING:

Unsupervised learning is a type of machine learning technique in which an algorithm discovers patterns and relationships using unlabeled data. Unlike supervised learning, unsupervised learning doesn't involve providing the algorithm with labeled target outputs. The primary goal of Unsupervised learning is often to discover hidden patterns, similarities, or clusters within the data, which can then be used for various purposes, such as data exploration, visualization, dimensionality reduction, and more.

Let's understand it with the help of an example.

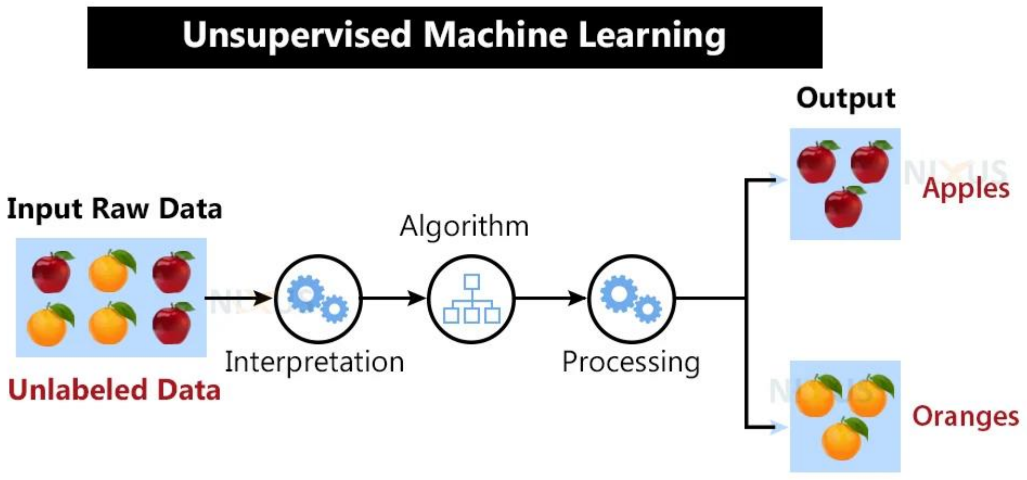


Figure 6

Example: Consider that you have a dataset that contains information about the purchases you made from the shop. Through clustering, the algorithm can group the same purchasing behavior among you and other customers, which reveals potential customers without predefined labels. This type of information can help businesses get target customers as well as identify outliers.

Main category of unsupervised learning-Clustering

Clustering is an unsupervised learning technique used to divide a dataset into groups (called clusters) such that data points within the same cluster are more similar to each other than to those in other clusters. It helps uncover hidden patterns or structures in data without using predefined labels.
Imagine a box full of mixed marbles of different colors: red, blue, and green. Without any prior knowledge about the marbles, your task is to sort them into groups based on their colors.

- Input: A collection of mixed marbles.
- Process: Group marbles based on their color similarity (you might visually notice clusters of red, blue, and green marbles forming).
- Output: Three clusters—one for red, one for blue, and one for green marbles.

Here, clustering identifies groups (clusters) of similar items (marbles) based on their shared characteristic (color), even though no labels (e.g., "Red," "Blue," "Green") are provided initially.
Applications of Unsupervised Learning

Here are some common applications of unsupervised learning:

- Clustering: Group similar data points into clusters.
- Anomaly detection: Identify outliers or anomalies in data.
- Dimensionality reduction: Reduce the dimensionality of data while preserving its essential information.
- Recommendation systems: Suggest products, movies, or content to users based on their historical behavior or preferences.

3. REINFORCEMENT LEARNING:

Reinforcement Learning is a type of machine learning where an agent learns to make decisions by interacting with an environment to maximize a reward. Unlike supervised learning, RL doesn't rely on labeled data; instead, the agent learns through trial and error using feedback in the form of rewards or penalties.

How Reinforcement Learning Works

1. The agent starts in an initial state.
2. It chooses an action based on its policy.

3. The environment transitions to a new state and provides a reward.
4. The agent updates its policy to maximize cumulative rewards over time.

This cycle repeats until the agent learns an optimal strategy.
Example: Teaching a robot to Walk

1. Environment: A simulated room.
2. Agent: The robot.
3. State: The robot's current position and posture.
4. Action: Moving its legs (forward, backward, lift, etc.).
5. Reward:
    a. +1: For walking a step without falling.
    b. -1: For falling.

Initially, the robot may fall often, receiving penalties. Over time, it learns which leg movements keep it stable and help it walk further.

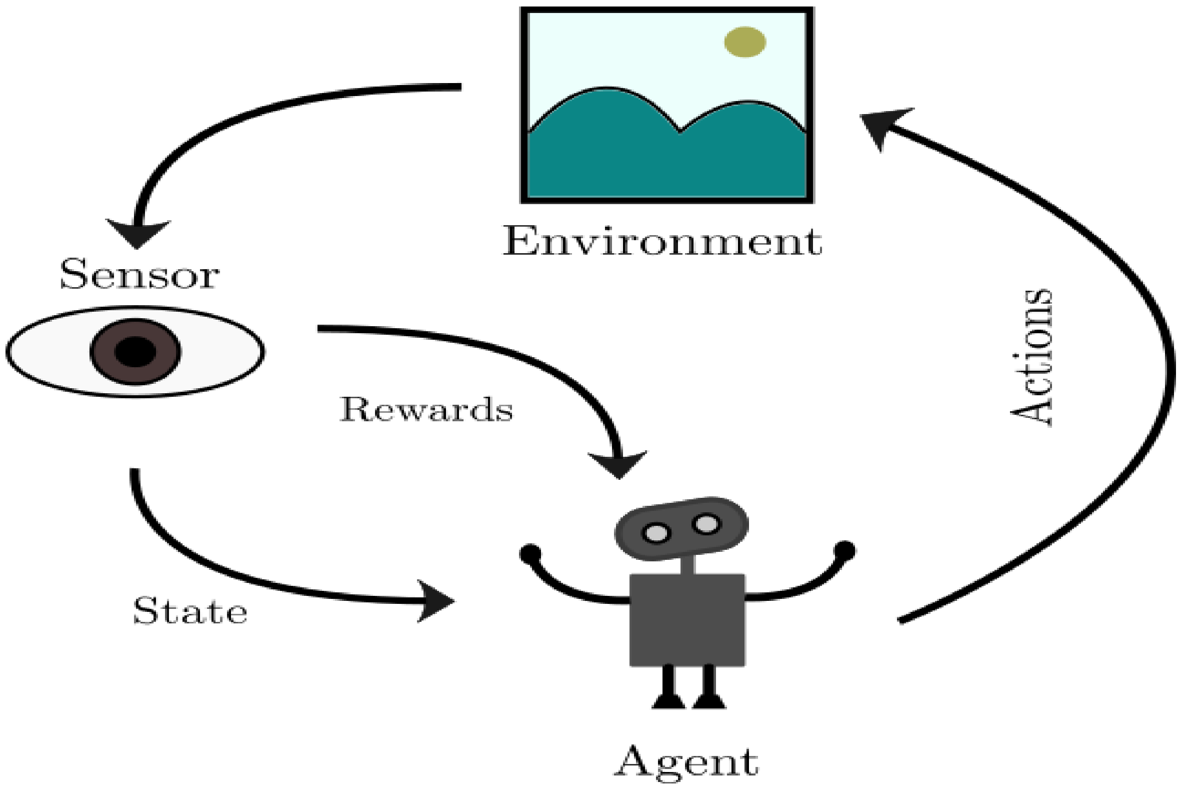


Figure 7

WORKFLOW OF MACHINE LEARNING MODEL:

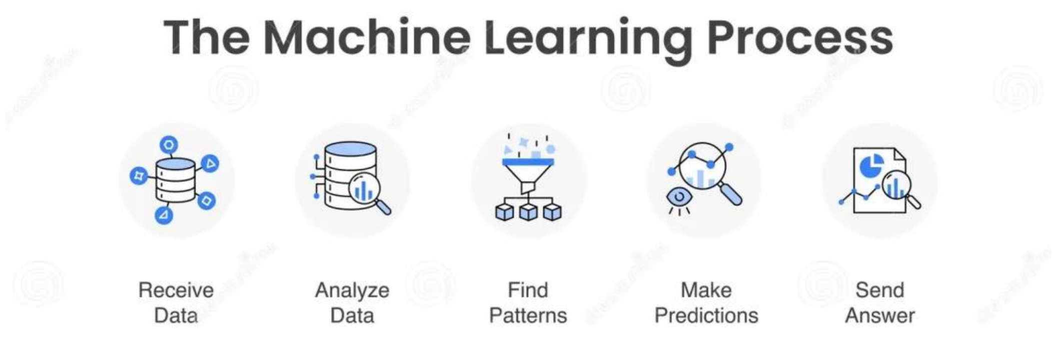


Figure 8

1.2 Motivation:

The motivation for developing the Plant Disease Detection System for Sustainable Agriculture stems from the critical challenges faced by the agricultural sector due to plant diseases. These diseases significantly reduce crop yields, leading to economic losses for farmers and threatening global food security. Traditional methods of disease detection are often manual, time-consuming, and prone to errors, resulting in delayed interventions and the excessive use of chemical pesticides, which harm both the environment and human health.

With the increasing global population, the demand for sustainable and efficient farming practices has never been more pressing. Early and accurate detection of plant diseases can significantly reduce crop losses, minimize the environmental impact of chemical usage, and enhance overall agricultural productivity. Additionally, advancements in artificial intelligence, computer vision, and IoT provide an opportunity to revolutionize agriculture by enabling precise, real-time monitoring and diagnosis of plant health.

The project is motivated by the potential to empower farmers with accessible, technology-driven tools that can enhance decision-making, promote eco-friendly farming practices, and contribute to sustainable development goals. By integrating cutting-edge technologies into agriculture, this system aims to address critical issues like food security, environmental sustainability, and economic resilience for farming communities.

Potential Applications and Impact

Applications:

1. Early Disease Detection:

   a. Rapid and accurate identification of plant diseases from leaf images, allowing farmers to take preventive actions before the disease spreads.

2. Precision Agriculture:
   a. Integration with IoT sensors to monitor environmental factors like temperature, humidity, and soil conditions, enabling precise disease management strategies tailored to specific crops and regions.
3. Farmer Assistance Tools:
   a. Mobile and web-based applications to provide farmers with real-time disease diagnostics, treatment recommendations, and preventive measures.

4. Agricultural Research:
   a. Use in agricultural research for studying disease patterns, identifying high-risk zones, and developing resistant crop varieties.

5. Sustainable Farming Practices:
   a. Encourages reduced use of chemical pesticides by promoting targeted interventions, supporting organic farming initiatives.

6. Government and Policy Support:
   a. Helping policymakers track disease outbreaks across regions and make informed decisions about resource allocation and disaster management.

Impact:

1. Increased Crop Yields:
   a. Early and accurate detection minimizes crop loss, ensuring higher productivity and profitability for farmers.

2. Reduced Environmental Impact:
   a. By minimizing excessive pesticide usage, the system promotes eco-friendly farming, reducing soil and water contamination.

3. Improved Food Security:
   a. Addressing plant diseases effectively helps ensure a stable food supply, especially in regions dependent on agriculture.

4. Economic Growth:
   a. Enhancing farmers' income through better crop health and reducing wastage contributes to the overall economic development of agricultural communities.

5. Accessible Technology:

a. Providing user-friendly tools empowers small-scale farmers with limited resources, bridging the gap in technology adoption.

1.3 Objective:

1. Early Detection of Plant Diseases: The primary aim of the project is to develop a system that can detect plant diseases at an early stage. By using image processing techniques and machine learning algorithms

2. Preventing Crop Loss: By detecting plant diseases early, farmers can take immediate action to prevent further spread and minimize crop loss. Early intervention can help in implementing targeted treatment strategies, thereby improving the overall health and yield of the crops.

3. Accuracy and Reliability: The project aims to achieve a high level of accuracy in disease detection by utilizing advanced image processing techniques and machine learning algorithms. The system should be reliable and provide consistent results, minimizing false positives and negatives to ensure effective disease management.

4. Promote Sustainable Agriculture: Minimize the excessive use of chemical pesticides by enabling precise and targeted interventions, thereby supporting eco-friendly and sustainable farming practices.

5.Contribute to Food Security: Address the global challenge of food security by ensuring healthier crops, reducing wastage, and increasing the availability of agricultural produce.

6. Empower Small Scale Farmers: Make the technology accessible and affordable for small and marginal farmers, bridging the gap between modern technology and traditional farming practices.

7. Support Data-Driven Decision Making: Leverage data analytics to identify patterns, high-risk zones, and disease outbreaks, providing valuable insights for researchers, policymakers, and farmers.

8. Scalability and Adaptability: Design the system to be scalable and adaptable for different crops, regions, and farming conditions, ensuring broad applicability and impact.

1.4 Scope of the Project:

1. Disease Detection and Diagnosis:
    a. The system focuses on detecting visible plant diseases from leaf images, providing early identification to prevent disease spread.

2. Farmer Assistance:
    a. A user-friendly mobile and web application offers real-time disease diagnosis, treatment suggestions, and preventive measures, empowering farmers with actionable insights.

3. Targeted Crop Protection:
    a. Reduces reliance on chemical pesticides by recommending targeted interventions, promoting eco-friendly farming.

4. Data Analysis and Reporting:
    a. Provides analytics on disease patterns, high-risk zones, and environmental factors to aid researchers and policymakers in decision-making.

5. Global Adaptability:
    a. The system is designed to be scalable and adaptable for various crops, farming conditions, and geographical regions.

6. Support for Sustainable Agriculture:
    a. Contributes to sustainable practices by reducing environmental impact and promoting efficient resource use.

LIMITATIONS:

Reliance on Visible Symptoms:

- The system may fail to detect diseases that do not show visible symptoms on leaves or that affect other parts of the plant.

Limited Crop Coverage:

- Initial development might focus on a few specific crops, requiring further expansion for broader applicability.

Accuracy Dependence on Image Quality:

- The system's effectiveness depends on high-quality images; poor lighting or unclear images may reduce accuracy.

CHAPTER 2
Literature Survey

2.1 LITERATURE REVIEW

1. Parez, S., Dilshad, N., Alanazi, T. M., & Lee, J. W. (2023). Towards Sustainable Agricultural Systems: A Lightweight Deep Learning Model for Plant Disease Detection. *Comput. Syst. Sci. Eng.*, *47*(1), 515-536.- The study presents **E-Green Net**, a lightweight deep learning framework utilizing MobileNetV3Small for efficient plant disease detection. Trained on diverse datasets, it achieves high accuracy (up to 1.00%) and outperforms existing methods in speed and performance. While promising, field testing, integration with IoT, and scalability for real-world adoption remain areas for improvement.

2. Ngongoma, M. S., Kabeya, M., & Moloi, K. (2023). A review of plant disease detection systems for farming applications. *Applied Sciences*, *13*(10), 5982.- This abstract highlights the need for digitizing agriculture to address global food insecurity and improve yields, particularly in developing regions like Africa and India. While reviewing plant disease detection models, it identifies gaps in real-time monitoring, early mitigation measures, and integrating multi-tasking models, emphasizing opportunities for further research and development.

3. Poonia, R. C., Singh, V., & Nayak, S. R. (Eds.). (2022). *Deep learning for sustainable agriculture*. Academic Press.

4. Elsayed, M. Z., Hasoon, A., Zidan, M. K., & Ayyad, S. M. (2024, July). Role of AI for plant disease detection and pest detection. In *2024 International Telecommunications Conference (ITC-Egypt)* (pp. 824-829). IEEE.- This literature review emphasizes the importance of artificial intelligence (AI) in automating pest and plant disease detection, which can reduce labor, time, and inaccuracies. It highlights challenges in the field and proposes solutions. The study provides valuable insights for researchers, making AI a crucial tool for improving agricultural disease management.

5. Shoaib, M., Shah, B., Ei-Sappagh, S., Ali, A., Ullah, A., Alenezi, F., ... & Ali, F. (2023). An advanced deep learning models-based plant disease detection: A review of recent research. *Frontiers in Plant Science*, *14*, 1158933.

2.2 TECHNIQUES AND METHODOLOGIES USED

2.2.1 DEEP LEARNING

Deep Learning is a subset of machine learning that uses neural networks with many layers (hence "deep") to automatically extract and learn complex patterns from data. It is particularly effective for tasks involving unstructured data, such as images, audio, and text. Deep learning comes into existence when we have to work with large unstructured data because when training with simple machine learning algorithms the accuracy of the model decreases due to large amounts of data.

In this automatically the deep learning architecture will extract features and there is no need to define it explicitly by the programmer.

Steps in Deep Learning

1. Define the Problem:
    a. Identify the task (e.g., image classification, text generation).
    b. Specify the input and output.
2. Collect and Preprocess Data:
    a. Gather a dataset relevant to the problem.
    b. Preprocess data:
        i. Normalize numerical data (e.g., scaling pixel values to [0, 1]).

ii. Augment data (e.g., rotate or flip images for more variety).
iii. Tokenize and pad sequences for text data.

3. Choose a Deep Learning Framework:
    a. Use libraries such as TensorFlow, PyTorch, or Keras to build and train models.
4. Design the Neural Network:
    a. Decide the architecture:
        i. Number of layers (e.g., convolutional, recurrent, dense).
        ii. Number of neurons in each layer.
        iii. Activation functions (e.g., ReLU, sigmoid, softmax).
    b. Select a loss function (e.g., cross-entropy, mean squared error).
5. Split Data:
    a. Split the dataset into training, validation, and test sets.
6. Train the Model:
    a. Feed training data into the model in batches.
    b. Optimize the model using an optimizer (e.g., Adam, SGD).
    c. Adjust weights and biases to minimize the loss.
7. Validate the Model:
    a. Evaluate performance on the validation set.
    b. Tune hyperparameters (e.g., learning rate, batch size).
8. Test the Model:
    a. Use the test set to measure final accuracy, precision, recall, etc.

MACHINE LEARNING VS DEEP LEARNING:

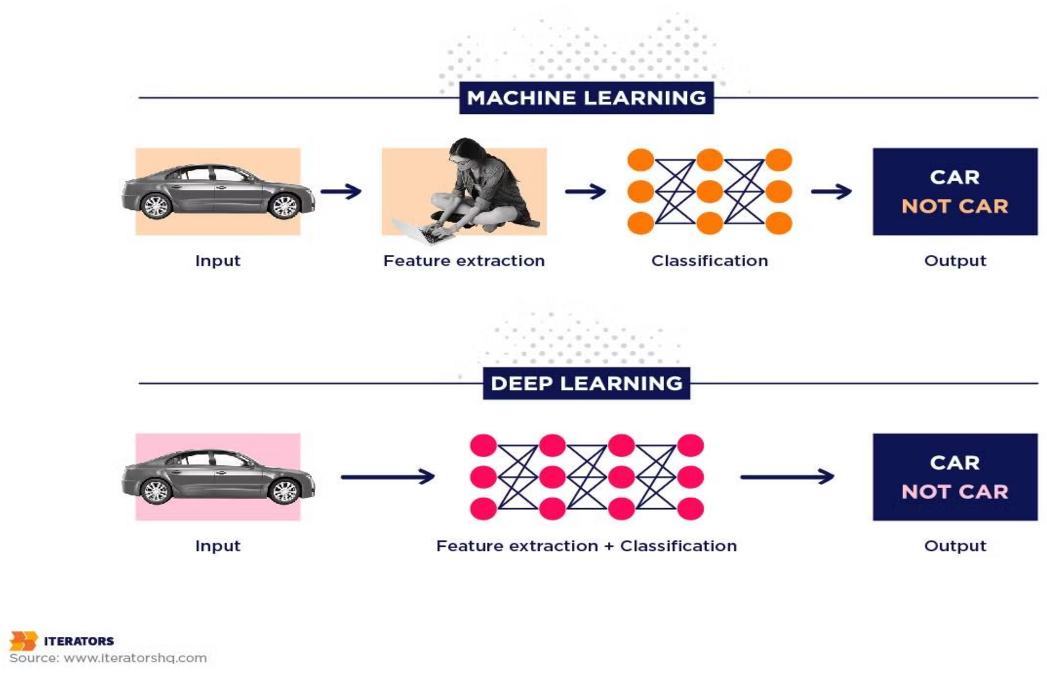


Figure 9

2.2.2 IMAGE CLASSIFICATION

Image classification is the process of categorizing and labeling an image based on its content. It involves using algorithms to analyze and assign a class or category label to an image. The classification typically occurs through the extraction of features from the image, such as color, texture, shape, and patterns, followed by the application of machine learning or deep learning models to categorize the image into predefined classes (e.g., "cat," "dog," or "plant disease").

Key Steps in Image Classification:

1. Preprocessing:
   a. Images are often resized, normalized, and sometimes augmented to enhance the model’s performance.
2. Feature Extraction:
   a. Features like edges, textures, and shapes are extracted from the image, which are then used to represent the image in a way the algorithm can understand.
3. Model Training:
   a. A model (such as a neural network) is trained on labeled images, learning to associate specific features with the correct class label.

4. Prediction:
    a. Once trained, the model can classify new, unseen images based on what it has learned.

Example:
Image classification is a task where the goal is to assign a label (class) to an input image. In **brain tumor classification**, the objective is to classify medical images (e.g., MRI scans) into categories, such as:

1. **No Tumor**
2. **Benign Tumor**
3. **Malignant Tumor**

This is a critical application in healthcare to assist radiologists and doctors in diagnosing brain tumors quickly and accurately.
Steps for Brain Tumor Classification
1. Define the Problem

- Objective: Classify MRI images into one of three categories: No Tumor, Benign Tumor, or Malignant Tumor.
- Input: MRI scans (images).
- Output: Predicted class label for each image.

2. Collect and Preprocess Data

- Data Collection:
    - Obtain MRI images of brain scans from medical datasets (e.g., Kaggle, TCIA, or hospital archives).
    - Ensure data includes diverse cases and imaging modalities.
- Data Preprocessing:
    - Resize all images to a uniform size (e.g., 224x224 pixels) for consistent input to the neural network.
    - Normalize pixel values to a range of [0, 1] by dividing by 255.
    - Augment the dataset to improve model robustness:
        - Rotate images.
        - Flip horizontally/vertically.
        - Add slight noise or adjust brightness.

- Label Encoding:
  - Assign numerical labels to classes: 0 (No Tumor), 1 (Benign), 2 (Malignant).

3. Split the Dataset

- Training Set: 70% of the images for training the model.
- Validation Set: 15% of the images for tuning hyperparameters.
- Test Set: 15% of the images for final evaluation.

4. Design the Neural Network
For image classification, a Convolutional Neural Network (CNN) is ideal because CNNs are excellent at capturing spatial patterns in images.
5. Train the Model

- Loss Function: Use categorical cross-entropy to measure the difference between predicted and true labels.
- Optimizer: Use Adam for faster convergence.
- Batch Size: Train in small batches (e.g., 32 images at a time).
- Epochs: Train the model over several iterations (e.g., 20–50 epochs).

6. Validate the Model

- Evaluate the model on the validation set after each epoch to monitor overfitting and adjust hyperparameters like:
  - Learning rate.
  - Number of layers or neurons.
  - Regularization parameters (e.g., dropout rate).

7. Test the Model

- Measure performance on the test set using metrics:
  - Accuracy: Overall correctness of predictions.
  - Precision: Focuses on correctly identifying true positives.

2.2.3 NEURAL NETWORK :

A neural network in deep learning is a computational model inspired by the way biological neural networks in the human brain process information.

It is a type of machine learning model that learns patterns from data and makes predictions or decisions based on those patterns.

Components of a neural network:

1. NEURONS:

   These are the basic units of a neural network. Each neuron receives inputs, processes them, and passes the result to the next layer.

2. LAYERS:

   Input Layer: Receives the raw data features.
   Hidden Layers: Intermediate layers where data transformations and computations occur.

   Output Layer: Produces the final result (e.g., classification, regression value).

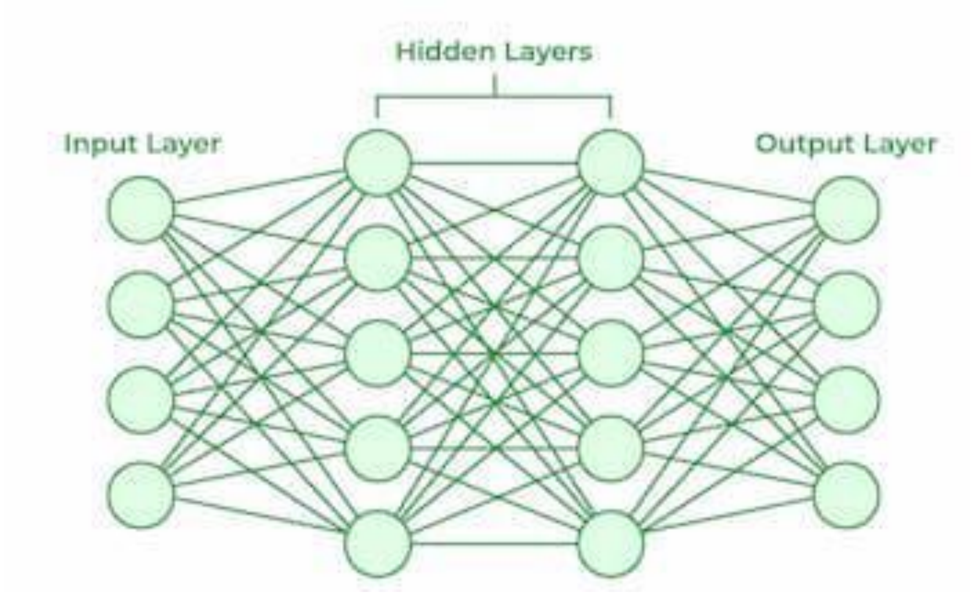


Figure 10

3. WEIGHTS AND BIASES:

Weights are parameters that scale the input values, determining their importance. Biases are additional parameters added to the weighted input to shift the output.

4. ACTIVATION FUNCTIONS:
   An activation function is a mathematical function applied to the output of a neuron. It introduces non-linearity into the model, allowing the network to learn and represent complex patterns in the data. Without this nonlinearity feature, a neural network would behave like a linear regression model, no matter how many layers it has.

   Examples: ReLU, Sigmoid, Tanh, Softmax.

Working of a neural network:

1. Forward Propagation

When data is input into the network, it passes through the network in the forward direction, from the input layer through the hidden layers to the output layer. This process is known as forward propagation. Here's what happens during this phase:

1. Linear Transformation: Each neuron in a layer receives inputs, which are multiplied by the weights associated with the connections. These products are summed together, and a bias is added to the sum. This can be represented mathematically as: z=w1x1+w2x2+…+wnxn+b
2. Activation: The result of the linear transformation (denoted as z is then passed through an activation function. The activation function is crucial because it introduces non-linearity into the system, enabling the network to learn more complex patterns. Popular activation functions include ReLU, sigmoid, and tanh.

2. Backpropagation

After forward propagation, the network evaluates its performance using a loss function, which measures the difference between the actual output and the predicted output. The goal of training is to minimize this loss. This is where backpropagation comes into play:

1. Loss Calculation: The network calculates the loss, which provides a measure of error in the predictions. The loss function could vary; common choices are mean squared error for regression tasks or cross-entropy loss for classification.

2. Gradient Calculation: The network computes the gradients of the loss function with respect to each weight and bias in the network. This involves applying the chain rule of calculus to find out how much each part of the output error can be attributed to each weight and bias.
3. Weight Update: Once the gradients are calculated, the weights and biases are updated using an optimization algorithm like stochastic gradient descent (SGD). The weights are adjusted in the opposite direction of the gradient to minimize the loss. The size of the step taken in each update is determined by the learning rate.

3. Iteration

This process of forward propagation, loss calculation, backpropagation, and weight update is repeated for many iterations over the dataset. Over time, this iterative process reduces the loss, and the network's predictions become more accurate.

Through these steps, neural networks can adapt their parameters to better approximate the relationships in the data, thereby improving their performance on tasks such as classification, regression, or any other predictive modeling.

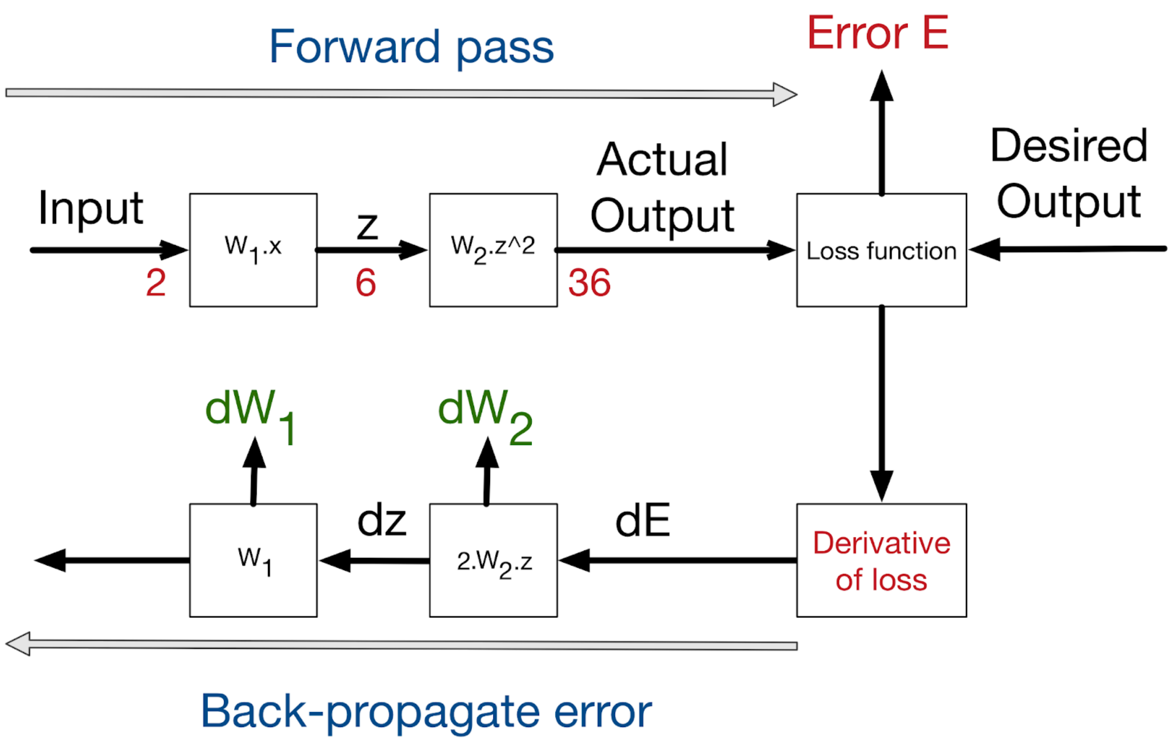


Figure 11

OVERFITTING AND UNDERFITTING:

1. Overfitting: refers to a scenario where a machine learning model can not generalize or fit well on unseen dataset. A clear sign of overfitting is that its error on testing and validation dataset is much greater than the error on testing dataset.
2. Underfitting: that can neither model the training dataset and nor generalize to new dataset

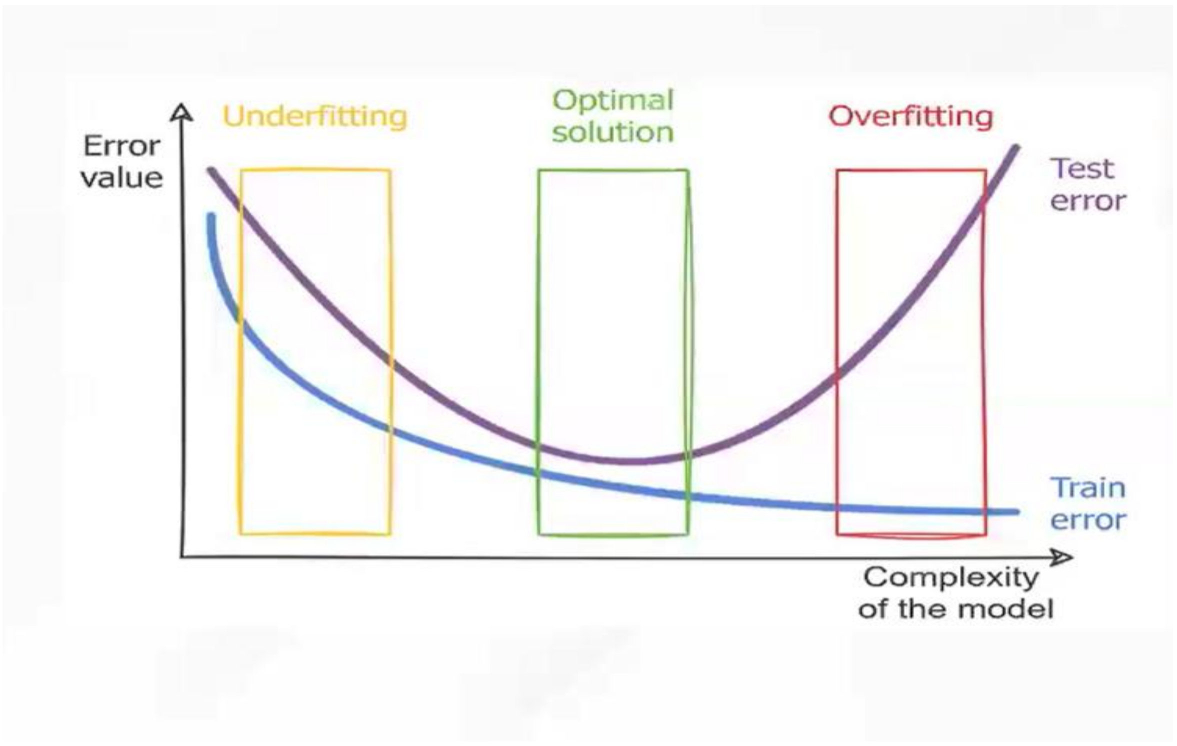


Figure 12

DROPOUT:

Dropout is a regularization technique used in neural networks to prevent overfitting. It works by randomly "dropping out" (deactivating) subsets of neurons during training, which forces the network to learn more robust and generalizable features.

2.2.4 CONVOLUTIONAL NEURAL NETWORK

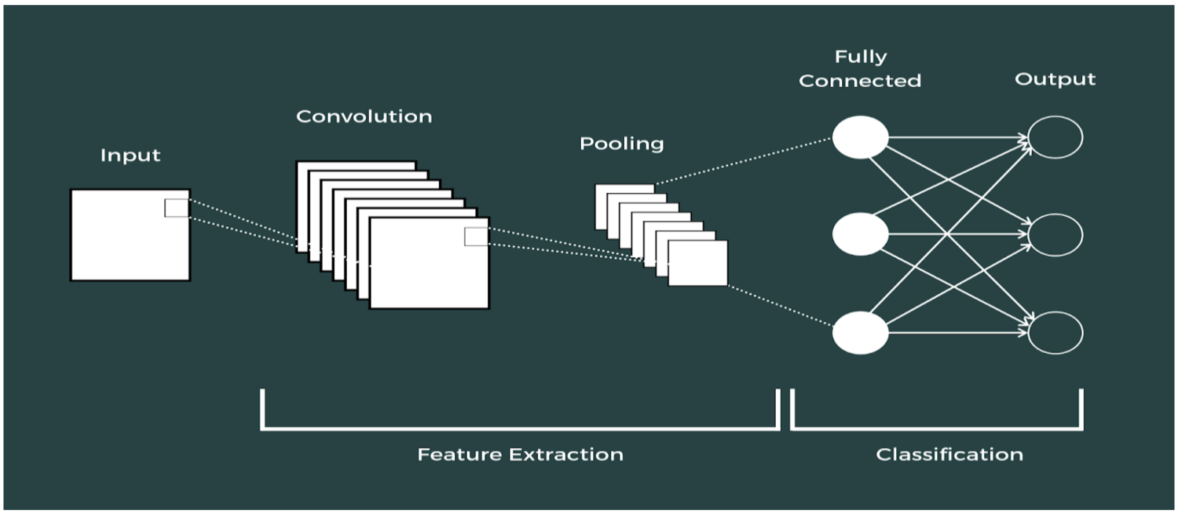


Figure 13

Introduction to CNN'S:

A Convolutional Neural Network (CNN) is a type of deep learning model primarily used for analyzing visual data such as images, video frames, or any data with a grid-like structure. CNNs are particularly well-suited for tasks like image classification, object detection, and segmentation due to their ability to automatically learn hierarchical features from raw image data.

The key advantage of CNNs over traditional neural networks is that they are designed to capture spatial hierarchies in images by exploiting the local relationships between pixels.

A CNN consists of several types of layers, each with its own function. Here's a breakdown of the main layers in a typical CNN:

1. Input Layer

- The input layer receives the raw image data. Images are usually represented as matrices of pixel values (e.g., 256x256 for a color image). Each pixel has values for the three-color channels: red, green, and blue (RGB).

2. Convolutional Layer

- The convolutional layer is the core building block of a CNN and performs the convolution operation, which is the process of applying filters (also called kernels) to the input image to detect patterns, edges, textures, and other local features.
- Filters: These are small matrices (e.g., 3x3 or 5x5) that slide over the input image, performing a dot product at each location. The result of the convolution is a feature map, which highlights the presence of specific features in the image.
- The idea is to learn different filters during training that can detect various features like edges, corners, and textures.

3. Activation Function (ReLU)

- After the convolution operation, the ReLU (Rectified Linear Unit) activation function is typically applied. ReLU introduces non-linearity by setting all negative values to zero and leaving positive values unchanged.
- This allows the network to learn more complex features and patterns beyond simple linear transformations.

4. Pooling (Subsampling) Layer

- The pooling layer is used to reduce the spatial dimensions (width and height) of the feature maps, helping decrease the number of parameters and computations, which reduces overfitting.
- Max Pooling is the most common technique, which selects the maximum value from a specific window (e.g., 2x2) of the feature map. This helps in retaining the most important features while reducing the dimensionality.

5. Fully Connected (FC) Layer

- After several convolutional and pooling layers, the feature maps are flattened (converted into a one-dimensional vector) and passed through one or more fully connected layers.
- These layers are traditional dense layers, where every neuron is connected to every neuron in the previous layer. The fully connected layers help in classifying the

extracted features into the final output classes (e.g., "cat" or "dog" in an image classification task).

6. Output Layer

- The final layer is typically a SoftMax layer for multi-class classification or a sigmoid layer for binary classification. It outputs a probability distribution over the classes, indicating which class the image belongs to.

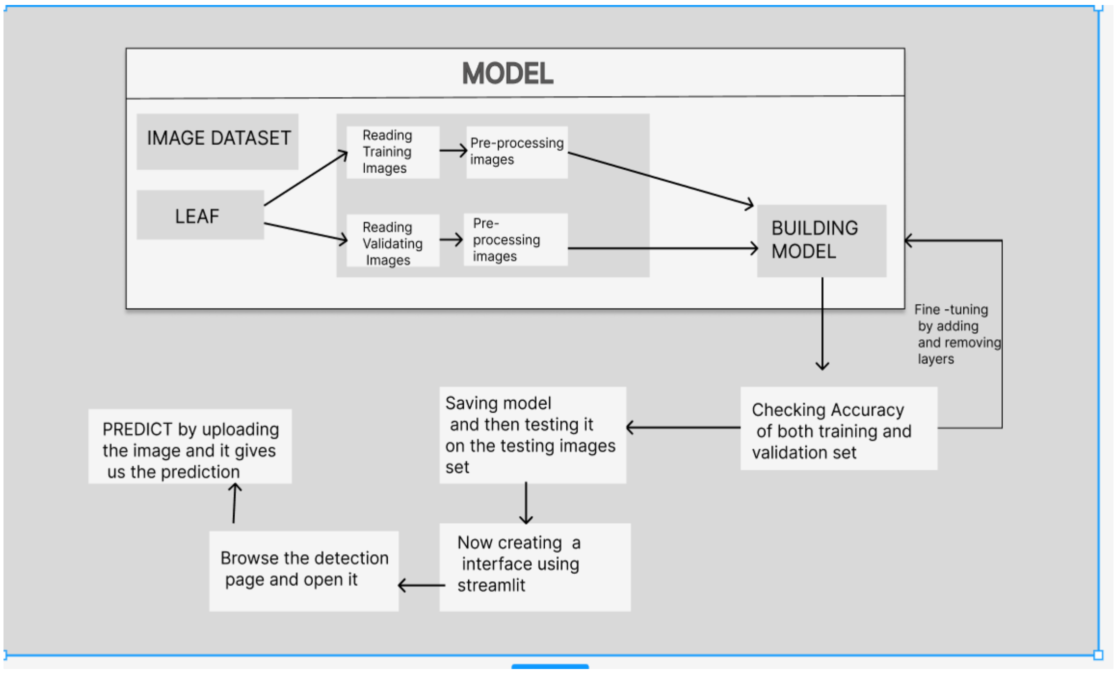
How CNNs Work:

1. Feature Learning:
   a. During training, CNNs learn filters that can automatically detect low-level features such as edges or textures in the initial layers and more complex features (like shapes or parts of objects) in deeper layers.
2. End-to-End Learning:
   a. CNNs can be trained end-to-end on labeled image datasets (e.g., ImageNet). The backpropagation algorithm adjusts the weights of filters based on the error between the predicted and true labels.
3. Hierarchical Feature Extraction:
   a. Each convolutional layer learns more abstract features by building on the features detected by previous layers. For example, while the first layer might learn simple edges, subsequent layers can detect patterns like corners, textures, or even higher-level structures like faces or animals.

CHAPTER 3
Proposed Methodology

3.1 System Design

Design is the first step into the development phase for any engineered product or system. Design is a creative process. A good design is the key to an effective system. The term “design” is defined as “the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization”. Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used.



MODEL
IMAGE DATASET
LEAF
Reading Training Images
Pre-processing images
Reading Validating Images
Pre-processing images
BUILDING MODEL
Fine -tuning by adding and removing layers
Checking Accuracy of both training and validation set
Saving model and then testing it on the testing images set
Now creating a interface using streamlit
Browse the detection page and open it
PREDICT by uploading the image and it gives us the prediction

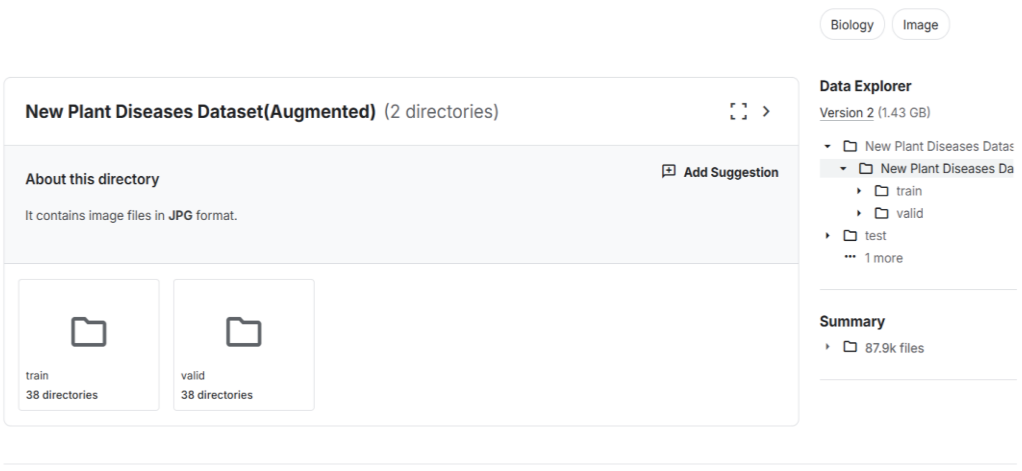
Figure 14

Model Section:

1. Image dataset:

This dataset consists of about 87K rgb images of healthy and diseased crop leaves which is categorized into 38 different classes. The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure.

A new directory containing 33 test images is created later for prediction purposes.


Biology Image

New Plant Diseases Dataset(Augmented) (2 directories)

About this directory

It contains image files in **JPG** format.

train
38 directories

valid
38 directories

Data Explorer
Version 2 (1.43 GB)

- New Plant Diseases Datas
  - New Plant Diseases Da
    - train
    - valid
- test
- 1 more

Summary
- 87.9k files

Figure 15

2. Leaf:

   a. Description: This refers to the images of plant leaves taken as input for disease detection.
   b. Purpose: These images are preprocessed and used for model training or validation.

3. Reading Training Images:

   c. Description: The system reads the training set of images from the dataset.
   d. Purpose: To load the training data into the pipeline for preprocessing and model training.

4. Preprocessing Images:
   e. Description: Images are resized, normalized, and potentially augmented to ensure consistent input for the model.
   f. Purpose: To prepare the images by removing noise, enhancing quality, and increasing dataset diversity.

5. Reading Validating Images:
   g. Description: The system loads the validation set of images.
   h. Purpose: To provide a separate dataset for evaluating the model's performance during training.

6. Building Model:

i. Description: The machine learning or deep learning model is built using techniques like convolutional neural networks (CNNs).
j. Purpose: To create a system capable of learning patterns in the training images for plant disease detection.

7. Fine-Tuning (Adding/Removing Layers):
   k. Description: The model is optimized by modifying its architecture, such as adding or removing layers, and adjusting hyperparameters.
   l. Purpose: To improve the model's accuracy and performance on both training and validation datasets.

Training and Evaluation Section

8. Checking Accuracy of Both Training and Validation Set:
   a. Description: The model is tested on both the training and validation datasets to evaluate its performance.
   b. Purpose: To ensure the model is learning effectively and not overfitting to the training data.

9. Saving Model and Then Testing It on the Testing Image Set:
   a. Description: After achieving satisfactory results, the trained model is saved and tested on a separate testing dataset.
   b. Purpose: To validate the model's real-world applicability and evaluate its accuracy on unseen data.

User Interface Section

10. Now Creating an Interface Using Stream lit:
    a. Description: A user-friendly interface is built using the Stream lit library.
    b. Purpose: To allow users to interact with the system by uploading images and receiving predictions.

11. Browse the Detection Page and Open It:
    a. Description: Users access the detection page via the Stream lit app.
    b. Purpose: To provide a gateway for users to upload plant images for disease detection.

12. Predict by Uploading the Image and It Gives Us the Prediction:
    a. Description: Users upload a plant image through the interface, and the system predicts whether the plant is healthy or diseased, along with the disease type (if applicable).

3.2 Requirement Specification

3.2.1 Hardware Configuration

● Processor: Intel i5

● Hard disk: 256 GB

● Memory : 8 GB

● Graphic Memory: N/A

3.2.2 Software Configuration

● Model: Convolutional Neural Network

● Frameworks and Libraries

● TensorFlow

● cv2

● NumPy

● IDE: Kaggle editor

● Platform: Windows

Explanation of various software requirements:

1. PYTHON:

Python is a dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming.

In Python, we don’t need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc.

a. Easy to code: Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, JavaScript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

b. Free and Open Source: Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

c. Object-Oriented Language: One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

d. GUI Programming Support: Graphical User interfaces can be made using a module such as PyQt5, PyQt4, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

e. High-Level Language: Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

f. Extensible feature: Python is an Extensible language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

g. Python is Portable language: Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

2. TENSORFLOW

TensorFlow is a Python-friendly open source library for numerical computation that makes machine learning and developing neural networks faster and easier. Machine learning is a complex discipline but implementing machine learning models is far less daunting than it used to be, thanks to machine learning frameworks—such as Google's TensorFlow—that ease the process of acquiring data, training models, serving predictions, and refining future results. Created by the Google Brain team and initially released to the public in 2015, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms (aka neural networks) and makes them useful by way of common programmatic metaphors. It uses Python or JavaScript to provide a convenient front-end API for building applications, while executing those applications in high-performance C++. 33 TensorFlow, which competes with frameworks such as PyTorch and Apache MXNet, can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation)-based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

3.NUMPY

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software.

It contains various features including these important ones:
a. A powerful N-dimensional array object
b. Sophisticated (broadcasting) functions
c. Tools for integrating C/C++ and Fortran code
d. Useful linear algebra, Fourier transform, and random number capabilities Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

4. OPENCV

OpenCV is an open source library which is very useful for computer vision applications such as video analysis, CCTV footage analysis and image analysis. OpenCV is written by C++ and has more than 2,500 optimized algorithms. When we create applications for computer vision that we don’t want to build from scratch we can use this library to start focusing on real world problems. There are many companies using this library today such as

Google, Amazon, Microsoft and Toyota. Many researchers and developers contribute. We can easily install it in any OS like Windows, Ubuntu and MacOS.

5. MATPLOTLIB

Matplotlib is a widely used Python library for creating static, interactive, and animated visualizations. It provides tools to produce high-quality graphs and plots for data analysis and presentation. As a foundational plotting library, Matplotlib supports various chart types, including line plots, scatter plots, bar charts, histograms, heatmaps, and more.

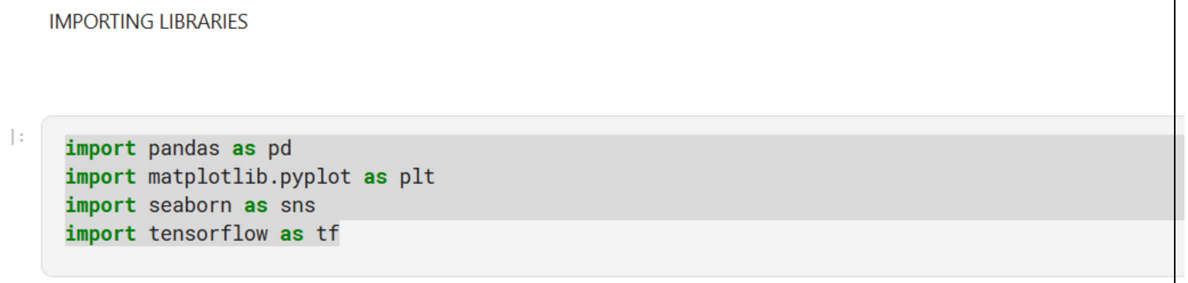
The library's most commonly used module is `pyplot`, which offers a MATLAB-like interface for simplicity.

CHAPTER 4

Implementation and Result

4.1 IMPLEMENTATION:

1. Importing libraries:



IMPORTING LIBRARIES

```
[ ]:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
```

Figure 16

Importing libraries in Python means bringing in external pre-written code (libraries or modules) into your program so you can use their functionalities without writing everything from scratch.

For example:

- pandas: For data handling.
- matplotlib: For visualizing data.
- TensorFlow: For building machine learning models.

By importing, you save time and leverage existing tools to make coding more efficient.

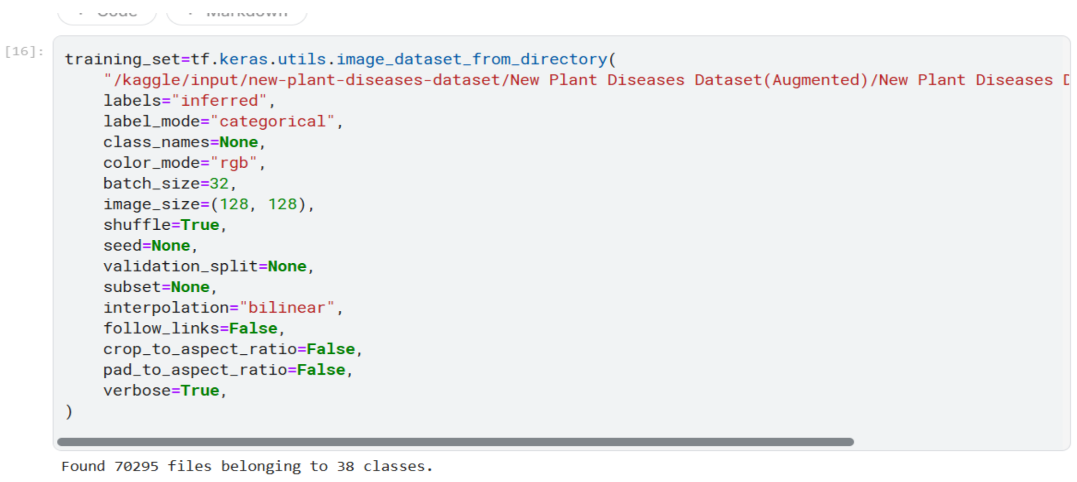
2. Data Preprocessing:

   Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves transforming raw data into a clean and usable format to ensure accurate and efficient model performance.

   It involves multiple steps to ensure the data is clean, consistent, and ready for modeling. The process begins with **data cleaning**, where missing values are handled by replacing them with the mean, median, or mode, using interpolation techniques, or dropping rows/columns with excessive missing data . Noise is removed by eliminating redundant or irrelevant data and filtering outliers. Next, **data integration** combines data from various sources into a unified dataset. In **data transformation**, numerical features are scaled using techniques like standardization or normalization (e.g., StandardScaler), and categorical variables are encoded into numerical formats using methods like one-hot or label encoding (OneHotEncoder).

Here we have used keras library for image preprocessing :

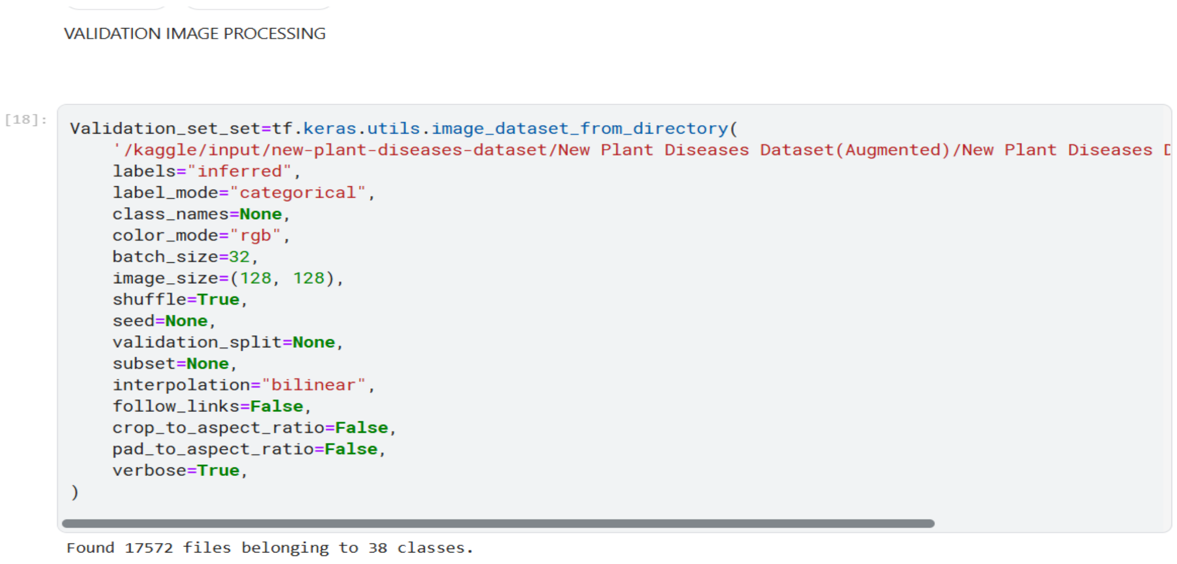
1. Firstly we have done it for training dataset and then for validation dataset



```
[16]: training_set=tf.keras.utils.image_dataset_from_directory(
          "/kaggle/input/new-plant-diseases-dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases D
          labels="inferred",
          label_mode="categorical",
          class_names=None,
          color_mode="rgb",
          batch_size=32,
          image_size=(128, 128),
          shuffle=True,
          seed=None,
          validation_split=None,
          subset=None,
          interpolation="bilinear",
          follow_links=False,
          crop_to_aspect_ratio=False,
          pad_to_aspect_ratio=False,
          verbose=True,
      )
```

```
Found 70295 files belonging to 38 classes.
```

Figure 17



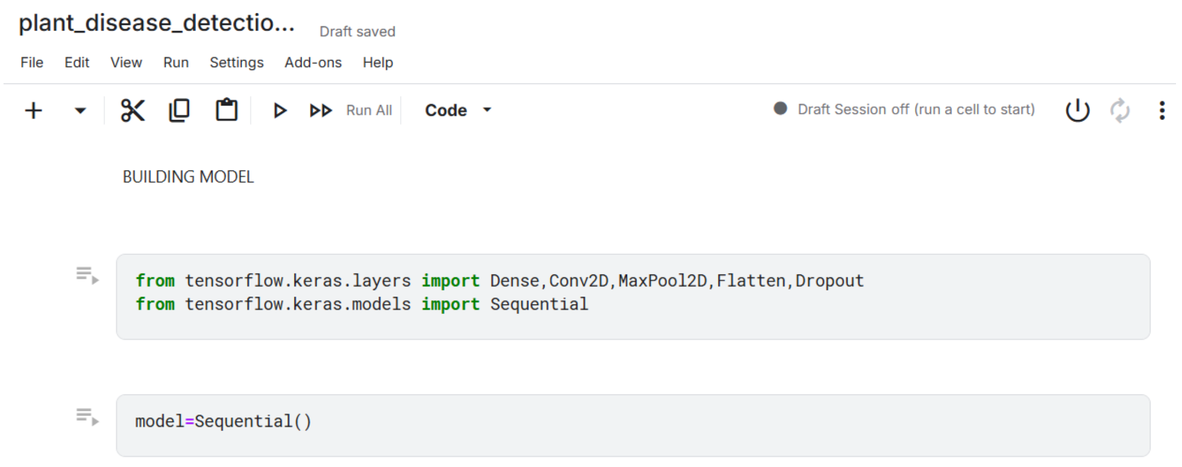
VALIDATION IMAGE PROCESSING

```
[18]: Validation_set_set=tf.keras.utils.image_dataset_from_directory(
          '/kaggle/input/new-plant-diseases-dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases D
          labels="inferred",
          label_mode="categorical",
          class_names=None,
          color_mode="rgb",
          batch_size=32,
          image_size=(128, 128),
          shuffle=True,
          seed=None,
          validation_split=None,
          subset=None,
          interpolation="bilinear",
          follow_links=False,
          crop_to_aspect_ratio=False,
          pad_to_aspect_ratio=False,
          verbose=True,
      )
```

```
Found 17572 files belonging to 38 classes.
```

Figure 18

3. Building model :


plant_disease_detectio...
Draft saved
File Edit View Run Settings Add-ons Help
Run All
Code
Draft Session off (run a cell to start)

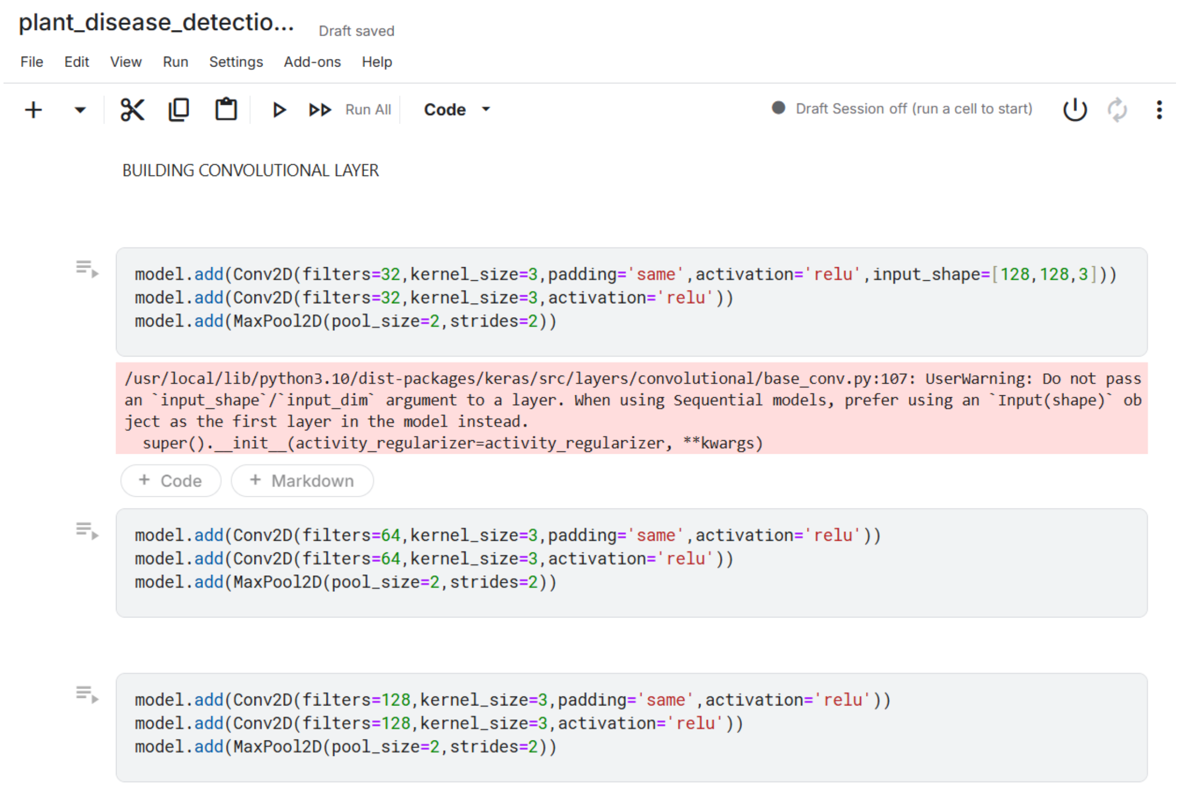
BUILDING MODEL

```
from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Flatten,Dropout
from tensorflow.keras.models import Sequential
```

```
model=Sequential()
```

Figure 19

Now we have built the default structure of the model and we will add layers to it.


plant_disease_detectio...
Draft saved
File Edit View Run Settings Add-ons Help
Run All
Code
Draft Session off (run a cell to start)
+ Code
+ Markdown

BUILDING CONVOLUTIONAL LAYER

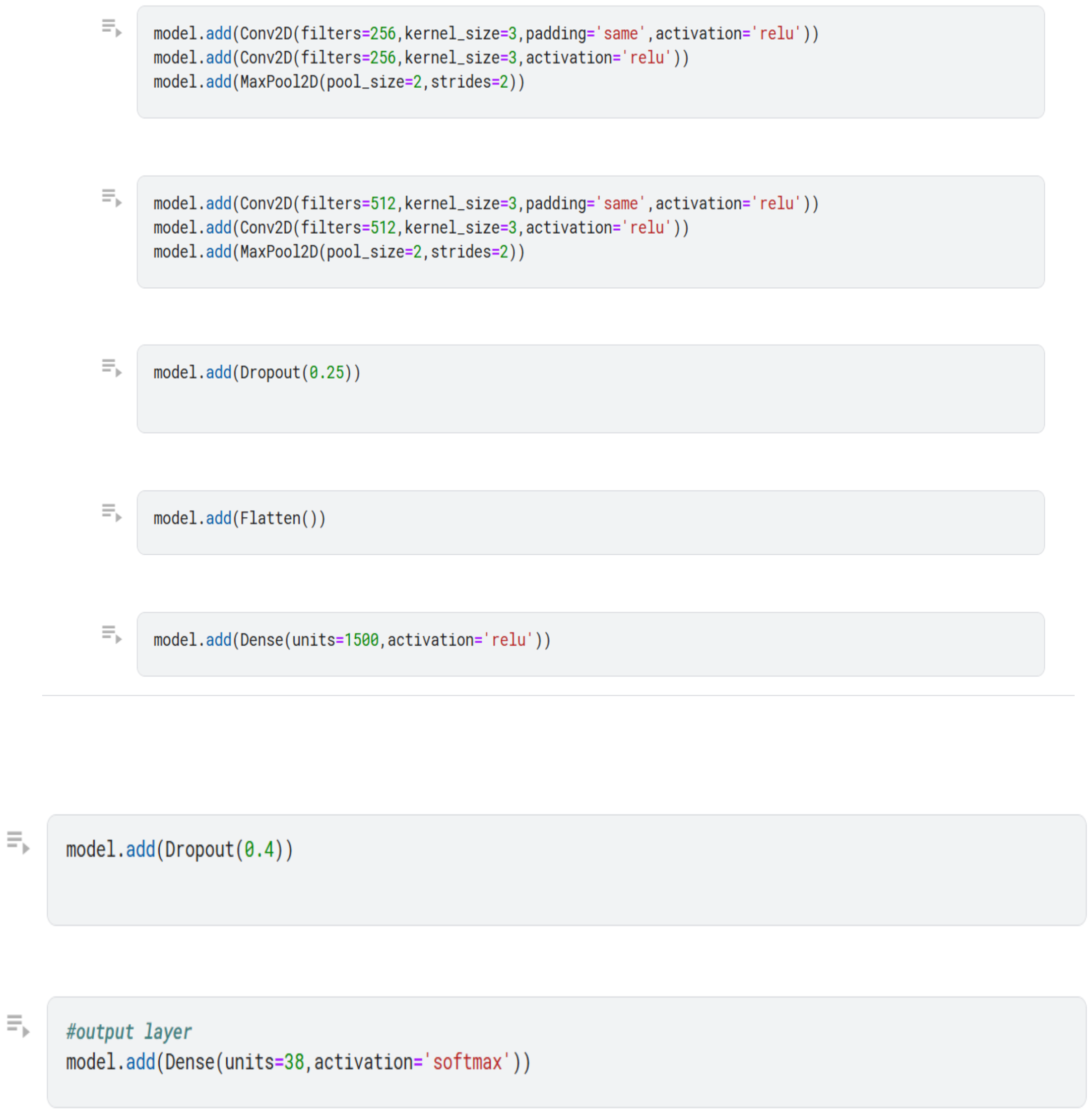
```
model.add(Conv2D(filters=32,kernel_size=3,padding='same',activation='relu',input_shape=[128,128,3]))
model.add(Conv2D(filters=32,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass
an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.add(Conv2D(filters=64,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=64,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
model.add(Conv2D(filters=128,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=128,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

Figure 20



```
model.add(Conv2D(filters=256,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=256,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
model.add(Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=512,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))
```

```
model.add(Dropout(0.25))
```

```
model.add(Flatten())
```

```
model.add(Dense(units=1500,activation='relu'))
```

```
model.add(Dropout(0.4))
```

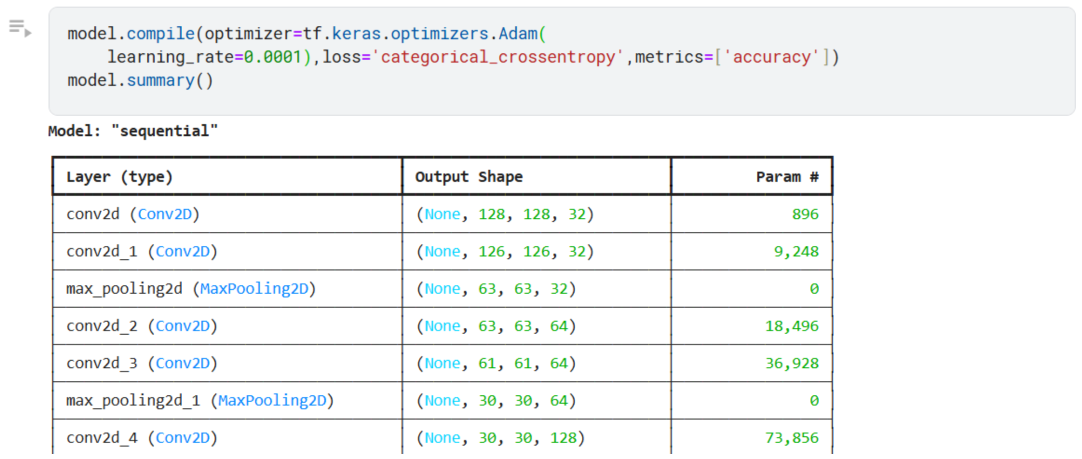
```
#output layer
model.add(Dense(units=38,activation='softmax'))
```

Figure 21

Here we have added various layers of Convolutional layers and also added dropout to prevent the complexity due to large number of layers and at the end we have added the output layer where we have applied the activation function ‘Soft max’.

4. Compiling model:

COMPILING MODEL



```
model.compile(optimizer=tf.keras.optimizers.Adam(
    learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

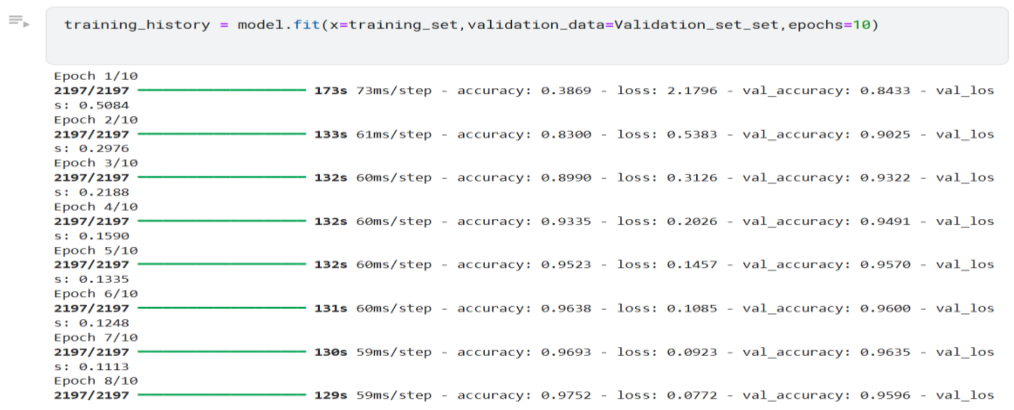
```
Model: "sequential"

┃ Layer (type)                    ┃ Output Shape            ┃      Param # ┃
│ conv2d (Conv2D)                 │ (None, 128, 128, 32)    │          896 │
│ conv2d_1 (Conv2D)               │ (None, 126, 126, 32)    │        9,248 │
│ max_pooling2d (MaxPooling2D)    │ (None, 63, 63, 32)      │            0 │
│ conv2d_2 (Conv2D)               │ (None, 63, 63, 64)      │       18,496 │
│ conv2d_3 (Conv2D)               │ (None, 61, 61, 64)      │       36,928 │
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 30, 30, 64)      │            0 │
│ conv2d_4 (Conv2D)               │ (None, 30, 30, 128)     │       73,856 │
```

Figure 22

Now we are compiling the model where we have added the optimizer in order to fine-tune the model so that the model properly learns every feature in the image. And we have used the ‘categorial_crossentropy’ to calculate the loss and we have chosen the metrics as accuracy.

5. Model Training:

MODEL TRAINING



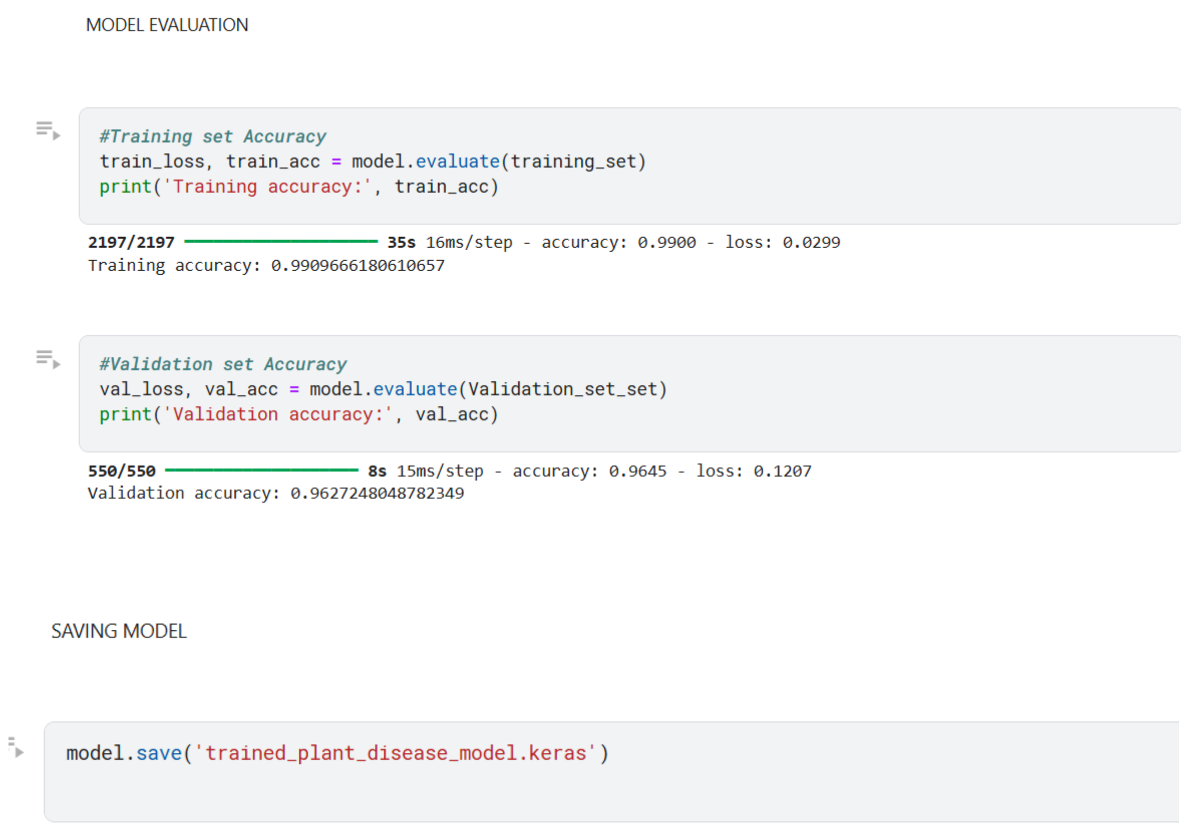
```
training_history = model.fit(x=training_set,validation_data=Validation_set_set,epochs=10)
```

```
Epoch 1/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 173s 73ms/step - accuracy: 0.3869 - loss: 2.1796 - val_accuracy: 0.8433 - val_los
s: 0.5084
Epoch 2/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 133s 61ms/step - accuracy: 0.8300 - loss: 0.5383 - val_accuracy: 0.9025 - val_los
s: 0.2976
Epoch 3/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 132s 60ms/step - accuracy: 0.8990 - loss: 0.3126 - val_accuracy: 0.9322 - val_los
s: 0.2188
Epoch 4/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 132s 60ms/step - accuracy: 0.9335 - loss: 0.2026 - val_accuracy: 0.9491 - val_los
s: 0.1590
Epoch 5/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 132s 60ms/step - accuracy: 0.9523 - loss: 0.1457 - val_accuracy: 0.9570 - val_los
s: 0.1335
Epoch 6/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 131s 60ms/step - accuracy: 0.9638 - loss: 0.1085 - val_accuracy: 0.9600 - val_los
s: 0.1248
Epoch 7/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 130s 59ms/step - accuracy: 0.9693 - loss: 0.0923 - val_accuracy: 0.9635 - val_los
s: 0.1113
Epoch 8/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 129s 59ms/step - accuracy: 0.9752 - loss: 0.0772 - val_accuracy: 0.9596 - val_los
```

Figure23

6. Model Evaluation:

Here we have evaluated the model on metric of accuracy and then saved the model.



MODEL EVALUATION

```
#Training set Accuracy
train_loss, train_acc = model.evaluate(training_set)
print('Training accuracy:', train_acc)
```

```
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 35s 16ms/step - accuracy: 0.9900 - loss: 0.0299
Training accuracy: 0.9909666180610657
```

```
#Validation set Accuracy
val_loss, val_acc = model.evaluate(Validation_set_set)
print('Validation accuracy:', val_acc)
```

```
550/550 ━━━━━━━━━━━━━━━━━━━━ 8s 15ms/step - accuracy: 0.9645 - loss: 0.1207
Validation accuracy: 0.9627248048782349
```

SAVING MODEL

```
model.save('trained_plant_disease_model.keras')
```

Figure24

7. Testing images:

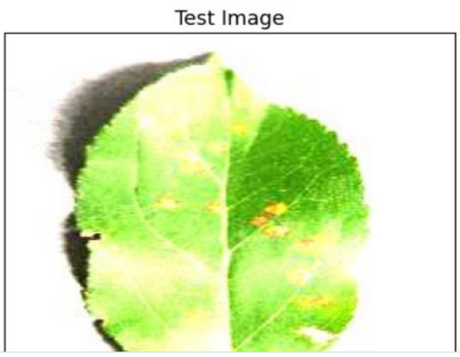
Now we will load the test images directory and apply the same pre-processing to the testing images. Then we will visualize the test image using open cv and matplotlib, further we perform the predictions on the unseen data.

TESTING IMAGES

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
validation_set = tf.keras.utils.image_dataset_from_directory(
    '/kaggle/input/new-plant-diseases-dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
class_name = validation_set.class_names
print(class_name)
```

```
#Test Image Visualization
import cv2
image_path = '/kaggle/input/new-plant-diseases-dataset/test/test/AppleCedarRust1.JPG'
# Reading an image in default mode
img = cv2.imread(image_path)
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB) #Converting BGR to RGB
# Displaying the image
plt.imshow(img)
plt.title('Test Image')
plt.xticks([])
plt.yticks([])
plt.show()
```



```python
import numpy as np
image = tf.keras.preprocessing.image.load_img(image_path,target_size=(128,128))
input_arr = tf.keras.preprocessing.image.img_to_array(image)
input_arr = np.array([input_arr])  # Convert single image to a batch.
predictions = model.predict(input_arr)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```

```python
print(predictions)
```

```
[[9.1804601e-08 9.8556967e-08 9.9992776e-01 1.7264391e-11 1.1525718e-09
  8.2156779e-09 2.6822924e-11 3.3138212e-10 4.2247042e-14 2.8496246e-13
  1.9390495e-14 2.4303294e-05 4.1274142e-10 1.9118287e-11 1.5048379e-07
  4.7864712e-09 1.3884731e-07 3.0488299e-11 2.4296710e-07 2.5462441e-09
  9.0464539e-13 5.5146217e-12 3.7650681e-11 1.0244301e-10 2.6888335e-11
  7.7279641e-12 2.2527633e-10 1.6339120e-10 8.4267940e-06 1.8608308e-08
  2.6645706e-09 1.4338265e-10 3.8535574e-05 1.2913228e-11 2.1194596e-07
  5.8414851e-10 4.0280673e-10 5.9756595e-09]]
```

```python
result_index = np.argmax(predictions) #Return index of max element
print(result_index)
```

```
2
```

```python
# Displaying the disease prediction
model_prediction = class_name[result_index]
plt.imshow(img)
plt.title(f"Disease Name: {model_prediction}")
plt.xticks([])
plt.yticks([])
plt.show()
```

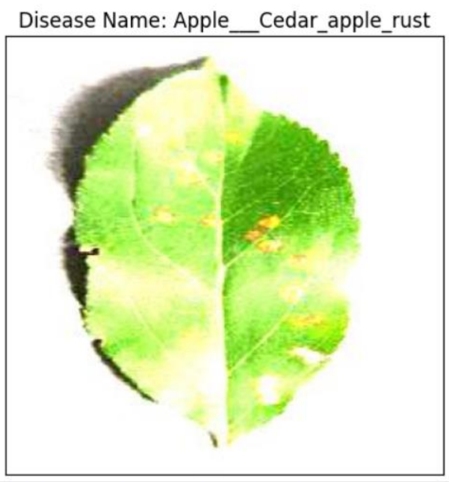
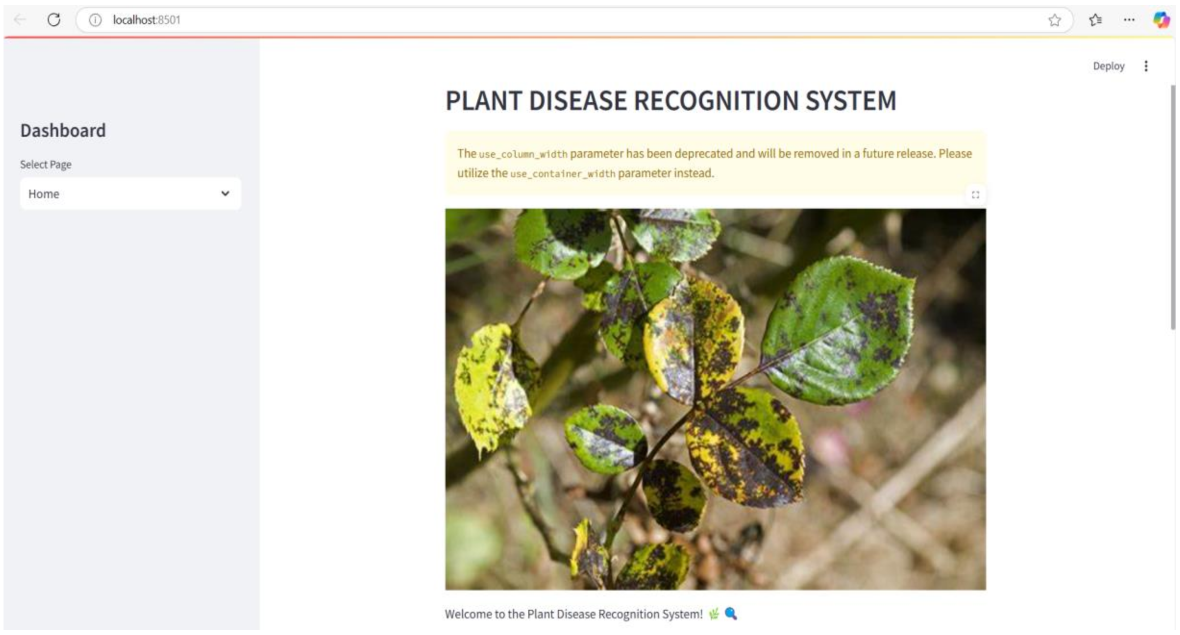


Figure 25

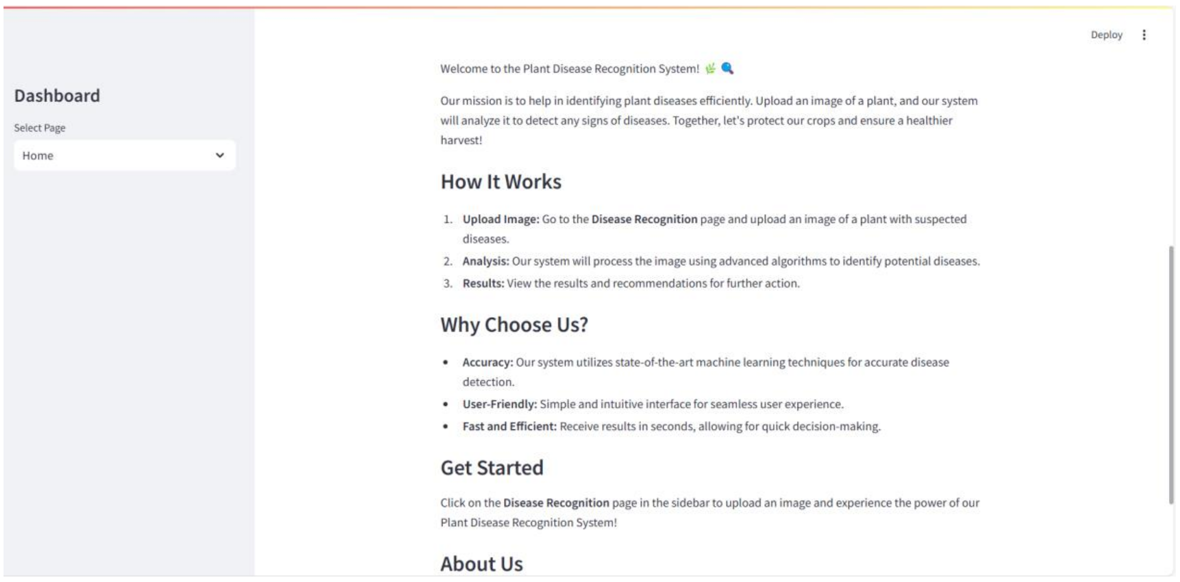
4.2 RESULTS-Snapshots of the interface created using stream lit



PLANT DISEASE RECOGNITION SYSTEM

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Welcome to the Plant Disease Recognition System! 🌿🔍



Welcome to the Plant Disease Recognition System! 🌿🔍

Our mission is to help in identifying plant diseases efficiently. Upload an image of a plant, and our system will analyze it to detect any signs of diseases. Together, let's protect our crops and ensure a healthier harvest!

How It Works

1. **Upload Image:** Go to the **Disease Recognition** page and upload an image of a plant with suspected diseases.
2. **Analysis:** Our system will process the image using advanced algorithms to identify potential diseases.
3. **Results:** View the results and recommendations for further action.

Why Choose Us?

- **Accuracy:** Our system utilizes state-of-the-art machine learning techniques for accurate disease detection.
- **User-Friendly:** Simple and intuitive interface for seamless user experience.
- **Fast and Efficient:** Receive results in seconds, allowing for quick decision-making.

Get Started

Click on the **Disease Recognition** page in the sidebar to upload an image and experience the power of our Plant Disease Recognition System!

About Us

Figure -The above Figure depicts the home page which is displayed once the code is run. On the left-hand side there is dashboard which further consists of three subpages that is: Home, About and Disease Recognition.

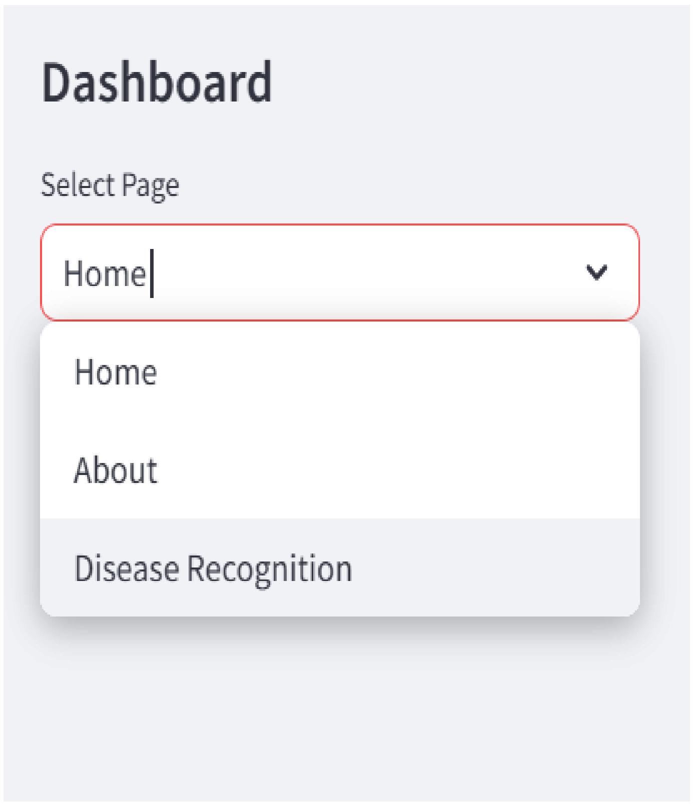
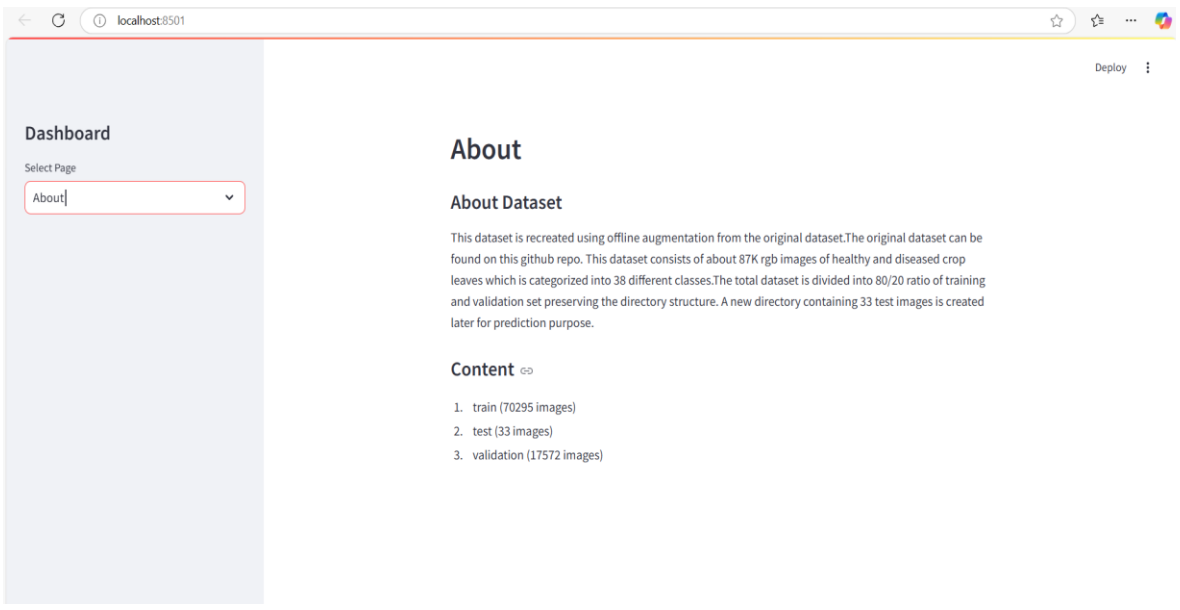


Figure Zoom-in view of dashboard



About

About Dataset

This dataset is recreated using offline augmentation from the original dataset.The original dataset can be found on this github repo. This dataset consists of about 87K rgb images of healthy and diseased crop leaves which is categorized into 38 different classes.The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purpose.

Content

1. train (70295 images)
2. test (33 images)
3. validation (17572 images)

Figure About Page

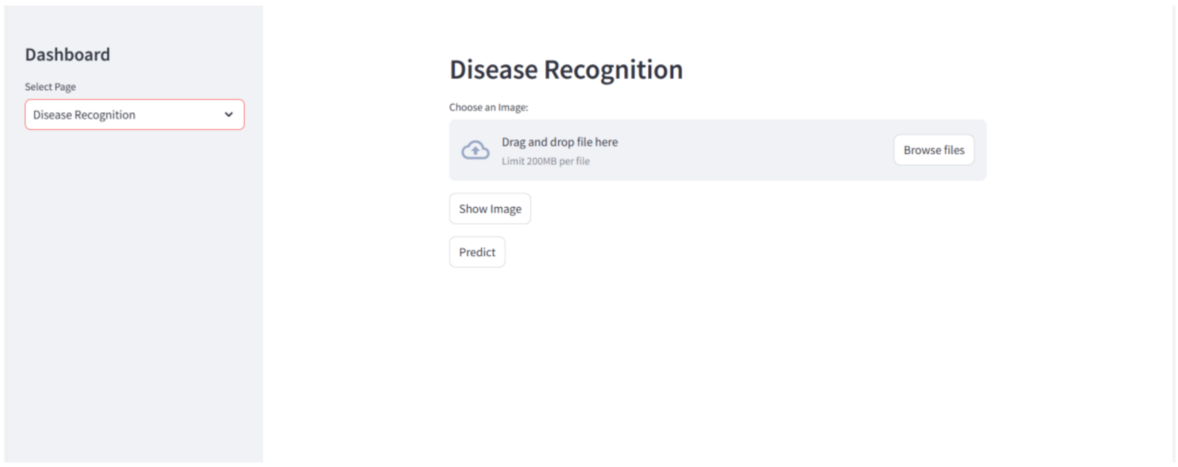


Figure Disease Recognition Page

Now we can predict the disease by first adding the image by browsing files through your computer and then there is also the functionality to view the image and then we click on predict, while the model is predicting the image I have added the snowflake feature it will throw snowflakes and then after predicting the disease, it provides us the result.

These features are showcased below figures

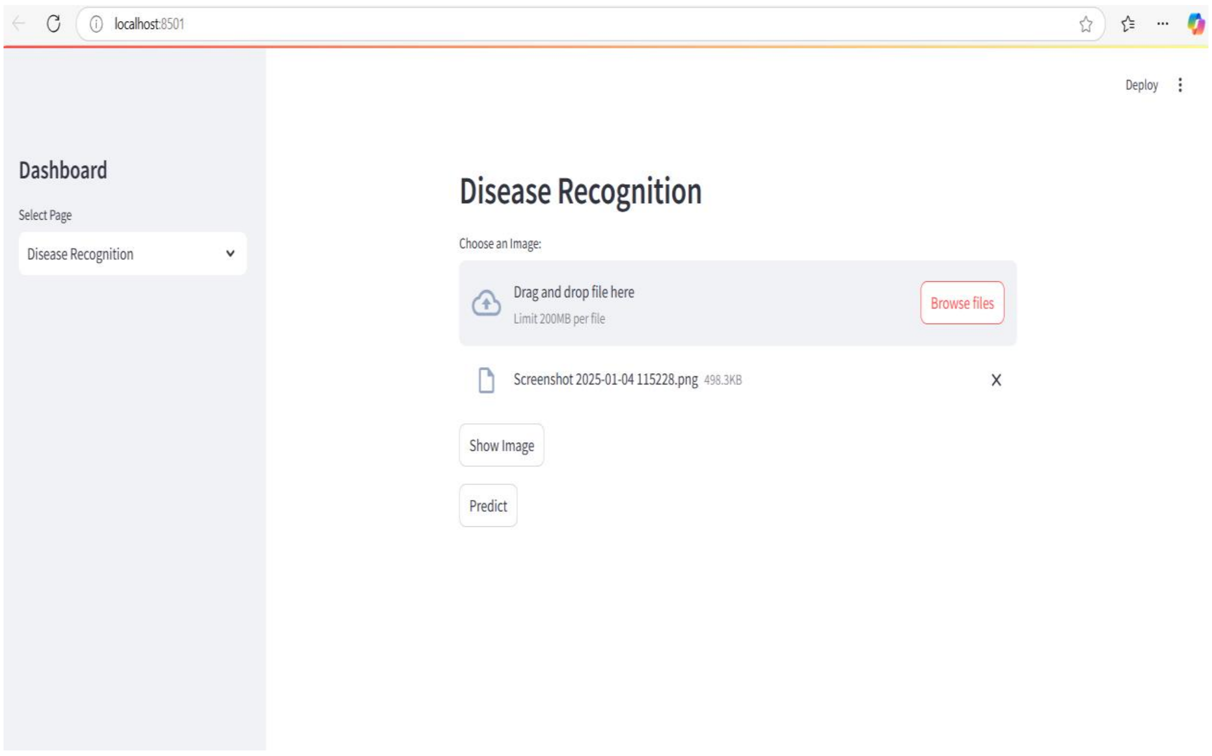


Figure After Selecting a image

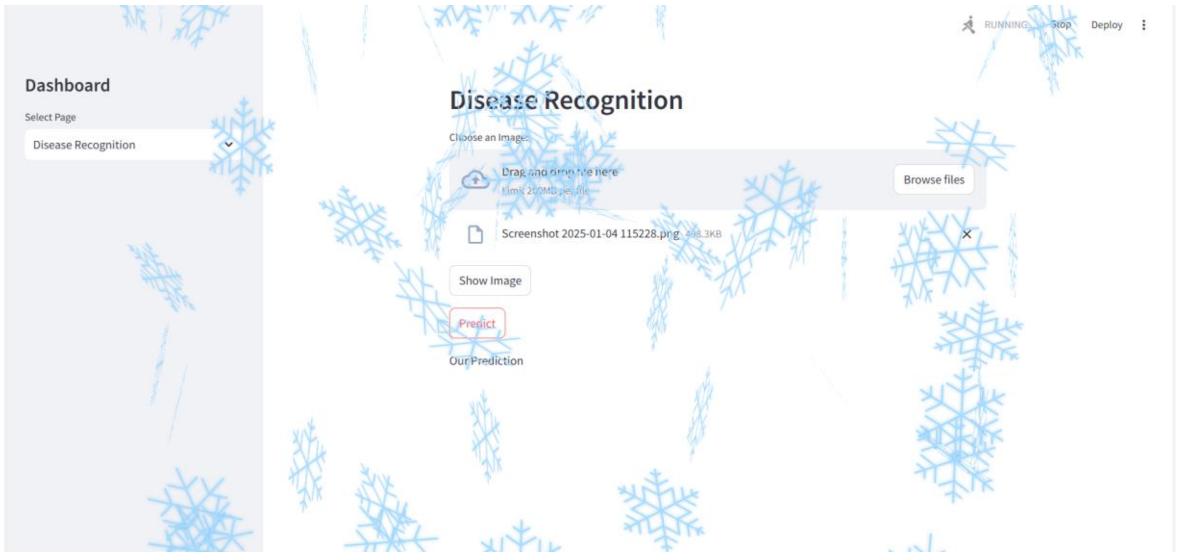
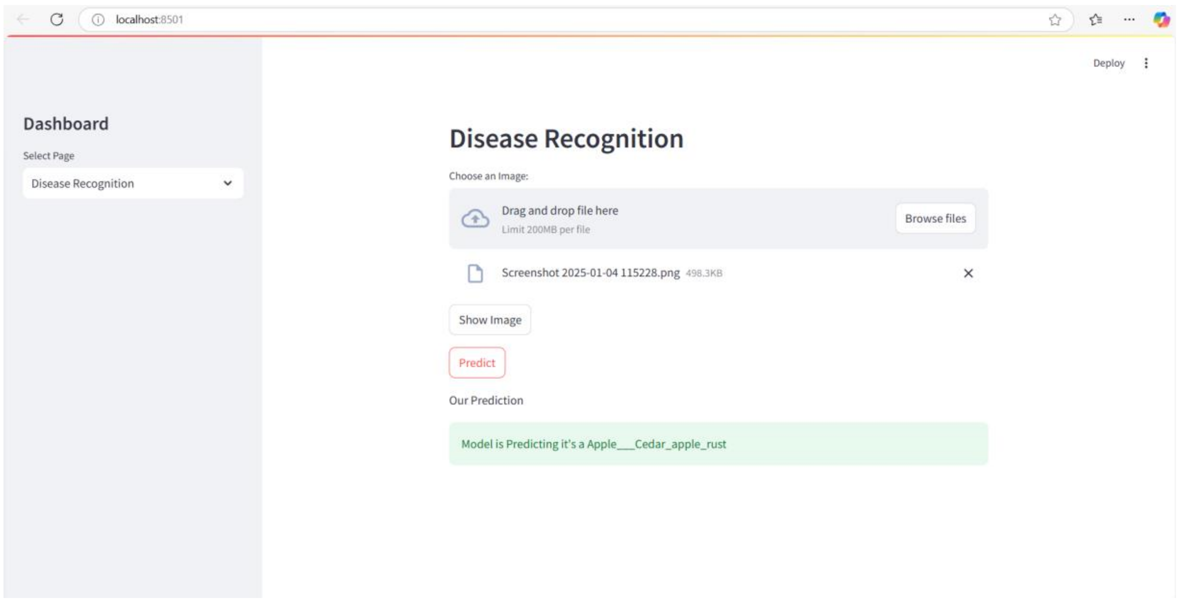


Figure Snowflakes while the model is predicting



Model is Predicting it's a Apple___Cedar_apple_rust

Figure After Prediction

4.2 GitHub Link for Code:

I have created a GitHub repository named ‘AICTE_internship’ where I have added the .ipynb file for the code and the code for the streamlit interface along with the dataset description which i have taken from kaggle.

sirat98/AICTE_internship

CHAPTER 5

Discussion and Conclusion

5.1 Future Work:

In the modern time with growing population across the world, it is important for us to maximize our resources. Agriculture is a field which is the basic requirement of all the living beings to survive. The early detection of disease in plants helps us to take steps in initial stages which increase our chances of a favorable outcome i.e., minimum damage done to the crop. This leads to increase in productivity and is also an important factor for the economy of the country.

Opportunities

1. To address the needs of disease detection in large areas: The application of current state of-the-art deep learning models for developing an automated visual inspection system for crops. This approach is cost effective and swifter than the manual approach.

2. To promote automation: The world is moving towards automation. This will save human resources and time required in the processing and prevention of crops from disease

Challenges

1. Unavailability of dataset: The datasets are often not available to train and hence will result in inaccurate results.

2. Accuracy Issues: The results often do not comply with the input dataset. This could be attributed to a lot of things. For instance, choosing an inefficient algorithm, setting less number of epochs, or due to other environmental constraints.

3. Improper selection of dataset.

4. Poor Quality of Images: If either the quality of images in the dataset is poor or the captured image is of the poor quality, the results shall also be inaccurate.

5.2 Conclusion:

A Plant Disease Detection Project using Convolutional Neural Networks (CNNs) aims to automate the identification of plant diseases through image analysis, which is crucial for early detection and effective crop management.

In this project, a large dataset of labeled images containing both healthy and diseased plant leaves is used to train the model. The CNN architecture, consisting of convolutional layers for feature extraction, activation layers for non-linearity, pooling layers for dimensionality reduction, and fully connected layers for classification, is employed to learn the patterns in plant images. The model is trained using a loss function like cross-entropy, optimized with algorithms such as Adam or SGD, and evaluated using metrics like accuracy, precision, recall, and F1-score.

Challenges such as dataset imbalance and overfitting are addressed using techniques like data augmentation and regularization methods. Once trained, the CNN model can accurately classify plant diseases, providing farmers with a tool for disease detection. Future improvements could include expanding the model to handle more plant species, optimizing the model for real-time deployment on mobile or IoT devices, and using advanced architectures like ResNet or Efficient Net to further improve accuracy.

Overall, this system has the potential to revolutionize agricultural practices by offering a scalable, automated solution for plant disease identification and management.

REFERENCES

[1]Introduction to Convolution Neural Network - GeeksforGeeks
[2] Deep Learning Tutorial - GeeksforGeeks
[1] New Plant Diseases Dataset