Krishi Rakshak

Plant Disease Detection

Group Members

ASMITA MITRA	2105705
AVANI SINGH	21052825
RADHIKA MANISH	2105730
ANUJ PRADHAN	21051716
BODHISATTA BHATTACHARJEE	22057025
NAALI MOKSHA	2105725

Our Vision

- Plant disease detection system designed to assist farmers in identifying and managing crop diseases early using advanced image recognition techniques.

- Technology: The system uses **CNN** to detect plant diseases from leaf images, utilising a large dataset.

Back to Agenda

Challenges

- Reduced Crop Yield
- Increased Costs of Treatment
- Loss of Income and Market Value
- Environmental Impact
- Spread of Disease to Neighboring Farms

Solution

- Digitization of agricultural problems
- Early Detection
- Data-Driven Decision Making

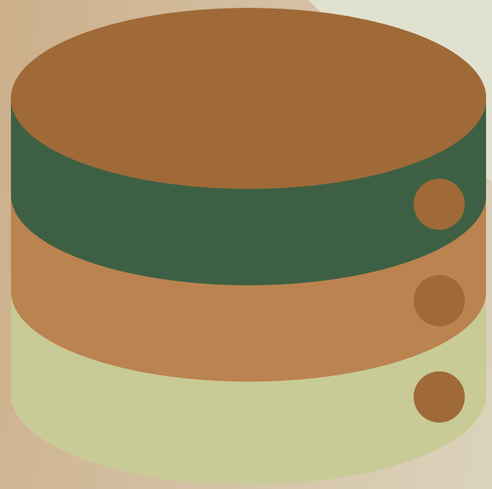
CNN

- classify plant diseases from leaf images using pattern recognition
- identification of diseases across multiple plant species.
- detect diseases early

Future Enhancements

- Adding more data
- Advanced Data Augmentation
- Transfer Learning

Datasets



01 The dataset, sourced from Kaggle, is 3GB in size and includes 38 disease classes across 14 plant species, specifically designed for plant disease classification.

02 The dataset covers common crops such as wheat, maize, and tomatoes, with disease types including fungal, bacterial, viral, and more.

03 The dataset contains over 1,500 labeled training images for model learning and more than 400 validation images for performance evaluation.

04 The dataset helps machine learning models recognize disease patterns in plants by focusing on labeled data for specific plant diseases.

05 The dataset is used to test model generalization and fine-tune performance, helping prevent overfitting by testing on unseen data.

- Utilized the **ImageDataGenerator** method from the **TensorFlow library**, which facilitates real-time data augmentation and preprocessing during training, helping to enhance the diversity of the training dataset.

- Implemented various image augmentation techniques to avoid overfitting, including adjusting the shear range to create variability in angles, applying random zooming effects to introduce different scales, and flipping images horizontally to enhance dataset diversity.

- These augmentation strategies play a crucial role in improving model generalization, helping to ensure that the model performs well on unseen data by exposing it to a broader range of image variations during training.

Key Operations

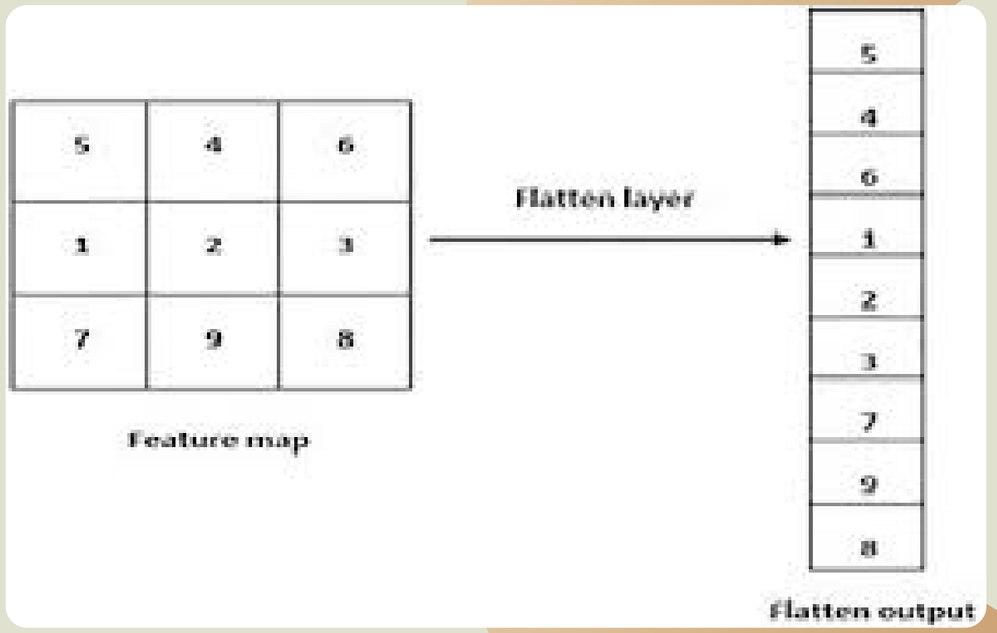
- **Convolution Operation**
  - Uses filters to detect features in images (edges, textures, etc.).
- **Stride and Padding**
  - Stride: Step size of filter movement.
  - Padding: Adds borders to retain image size.
- **Sequential Model (TensorFlow)**
  - Layers are added sequentially; each layer's output is input for the next.

Key Operations (continued)

- **Conv2D Layer Parameters**
  - **Filters: Number of feature detectors.**
  - **Kernel Size: Size of the filter (e.g., 3x3).**
  - **Activation: ReLU introduces non-linearity.**
  - **Input Shape: [64, 64, 3] (image size and RGB channels).**
- **Max Pooling**
  - **Reduces image size, highlights key features, and prevents overfitting.**

Flattening Layer

- Converts **3D output** into a **1D vector**.
- **Input:** 3D array (height × width × channels).
- **Process:** Flattens multi-dimensional data into a single vector.
- **No Learnable Parameters:** Only reshapes data, doesn't learn weights.



Fully Connected Layer

1

Purpose

Connects every **neuron** between **layers**, forming a dense network.

2

Input

Takes the flattened **1D vector** from the previous layer

3

Process

Computes **weighted sums** followed by an **activation function**

4

Reason

The fully connected layer combines **high-level** features from **convolutional** and **pooling layers** for final decisions.

5

Learnable Parameters

It has weights (one for each connection between neurons) and biases, which are updated during training through backpropagation.

6

Output

- If the fully connected layer has NNN neurons, it will output a vector of size NNN.
- The last fully connected layer is often the final classification layer, where the number of neurons corresponds to the number of output classes.

CNN Architecture

Input Layer

- Takes in **raw data**, usually images (e.g., 32x32x3 for color images).
- Each **pixel** in the image becomes an **input to the network**.

Convolutional Layer

- Applies a set of filters (**kernels**) to the input image.
- Captures **spatial patterns** like edges, textures, and more complex features.
- Produces feature maps (**activation maps**).
- **Learnable parameters**: Filter weights and biases.

Activation Function

- Introduces **non-linearity** after convolution.
- **ReLU** (Rectified Linear Unit) is commonly used
- Helps the **network capture complex patterns**.

Pooling Layer

- Reduces the **spatial dimensions** of feature maps.
- Keeps important features while **reducing computational load**.
- **Common methods**: Max Pooling, Average Pooling.

CNN Architecture

Repeat Convolution + Activation + Pooling

- **Stacks of Conv, Activation**, and **Pooling layers** are used to progressively extract more complex features from the input.
- As we go deeper, the spatial dimension decreases, but the depth (number of feature maps) increases.

Flattening Layer

- Converts the **multi-dimensional output** of the last pooling layer into a **1D vector**.
- Prepares the data for fully **connected layers.**

Fully Connected (FC) Layer

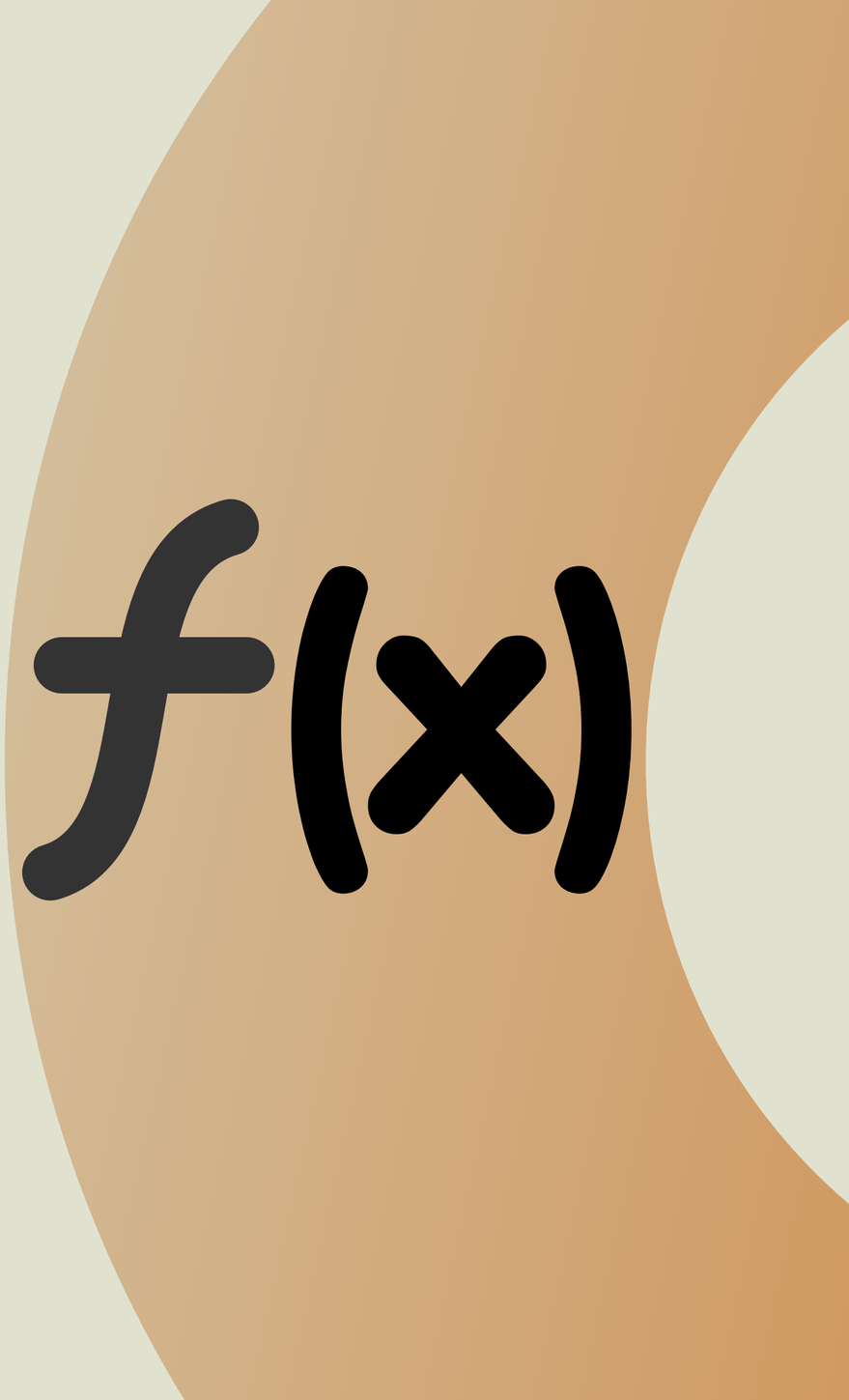
- Connects each **neuron** from the **previous layer** to all neurons in the **current layer**.
- Usually at the end of the network for final decision-making (**classification or regression**)

Output Layer

- Provides the **final output**, often using a softmax activation function for classification.
- The **number of neurons equals the number of classes** (for classification tasks)

Cost Function in CNN

- **Purpose**:
  - Measures the difference between the predicted output and the actual label.
  - The goal of training is to minimize the cost function.
- **Common Cost Functions**:
  - **Cross-Entropy Loss (for classification)**:
    - Measures how well the predicted probabilities match the actual class labels.
  - **Mean Squared Error (MSE) (for regression)**:
    - Measures the average squared difference between predicted values and actual values



Backpropagation in CNN

- **Purpose:**
  - Backpropagation is used to adjust the weights and biases in the CNN to minimize the cost function.
- **Steps:**
  - **Forward Pass:** Compute predictions by passing the input through the network.
  - **Compute Error:** Calculate the error using the cost function (e.g., Cross-Entropy Loss).
  - **Backward Pass:**
    - Gradients are calculated for each weight in the network using the **chain rule of calculus.**
    - **Gradients** represent how much **each weight contributes** to the error.
  - **Weight Update:**
    - Weights are updated using **Gradient Descent** (or variants like Adam) based on the computed gradients
- **Iterative Process:**
  - **Backpropagation** occurs after each batch of training data.
  - The process is repeated over many iterations (epochs) until the cost function converges

1

Confusion Matrix Overview

- **True Positives** (TP): Correctly predicted positives.
- **True Negatives** (TN): Correctly predicted negatives.
- **False Positives** (FP): Incorrectly predicted positives.
- **False Negatives** (FN): Incorrectly predicted negatives.

2

Metrics Derived from Confusion Matrix

- **Accuracy**: Overall correctness.
- **Precision**: Correctness of positive predictions.
- **Recall** (Sensitivity): Ability to identify actual positives.
- **F1 Score**: Balance between precision and recall.

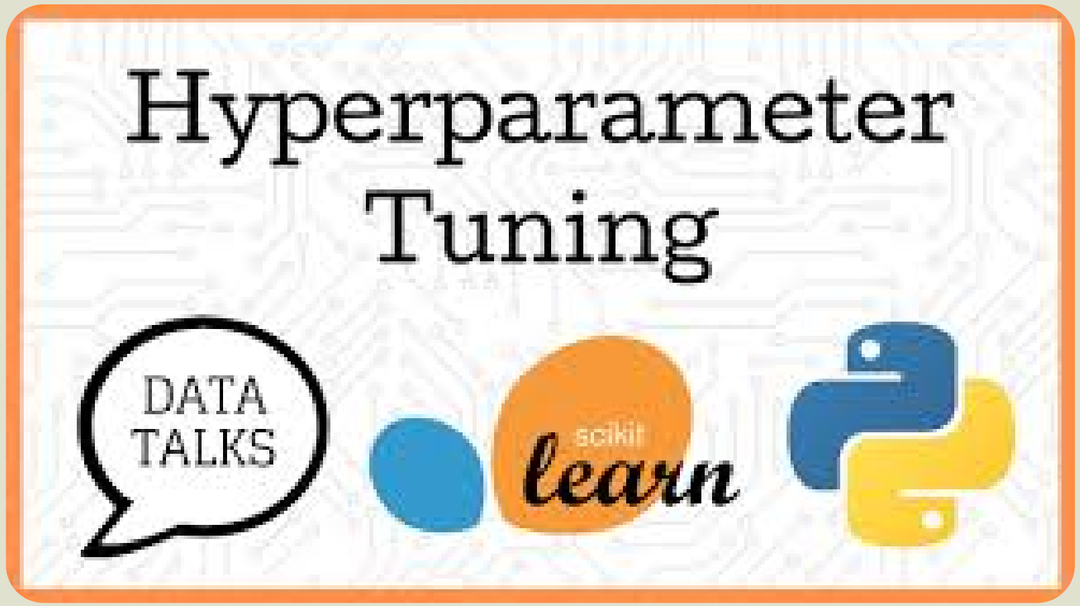
3

Hyperparameter Tuning for Model Improvement

- **Learning Rate**: Controls model adaptation speed.
- **Batch Size**: Samples per iteration.
- **Number of Epochs**: Number of dataset passes.
- **Optimizer**: Algorithm to adjust weights (e.g., Adam, SGD).
- **Dropout Rate**: Prevents overfitting.

Hyperparameter Tuning Techniques

- **Grid Search**: Systematic but time-consuming.
- **Random Search**: Faster, less comprehensive.
- **Automated Tools**: Use Keras Tuner, HyperOpt for tuning.



Thank you!