Smart Farming: Predicting Crop Diseases with AI and Machine Learning

Himanshu[a]

[a]*Student, Department of mathematics, Indian Institute of Technology, Hauz Khas, Delhi, 110016, Delhi, India*

Abstract

This study explores the application of Machine Learning techniques for predicting crop diseases, leveraging Convolutional Neural Networks (CNNs) and OpenCV for image-based analysis. The developed model achieved an impressive accuracy of 96% on untrained data, demonstrating its effectiveness in identifying crop diseases. In addition, a user-friendly application was built using Streamlit to facilitate practical implementation. This paper delves into the architecture of CNNs, the preprocessing steps used and the methodology that led to the achieved accuracy, along with details on the application development.

Keywords: Crop Disease Prediction, Machine Learning, Convolutional Neural Networks, OpenCV, Image Analysis, Streamlit Application

1. Introduction

Crop diseases significantly impact agricultural productivity and global food security. Traditional diagnostic methods are time-consuming and require expert knowledge, making them inaccessible to many farmers. Machine Learning and Computer Vision offer promising solutions to automate disease detection with high accuracy.

This study uses Convolutional Neural Networks (CNNs) and OpenCV to predict crop diseases, achieving accuracy 96% on untrained data. In addition, a user-friendly application was developed using Streamlit to enable practical implementation. This paper outlines the methods, results, and potential applications of this approach in modern agriculture.

2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specific type of neural network commonly used for image analysis tasks. They generally consist of layers such as convolutional layers, pooling layers, and fully connected layers.

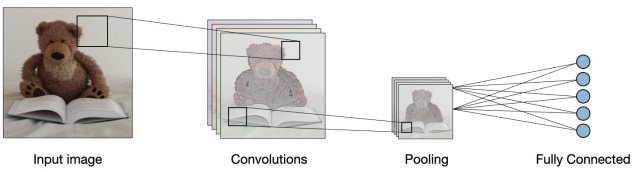

Figure 1: Different types of layers in CNNs

- **Convolution Layer(CONV)** - The convolution layer (CONV) applies filters to input I, performing convolution operations as it traverses the input dimensions. Key hyperparameters of this layer include the filter size F and the stride S. The output O generated from these operations is referred to as a feature map or activation map.

- **Pooling(POOL)** - The pooling layer (POOL) performs a downsampling operation, usually applied after the convolutional layer, to introduce spatial invariance. Specifically, max-pooling selects the maximum value from a given region, while average-pooling computes the average value of the region.

- **Fully Connected(FC)** - The fully connected layer (FC) processes a flattened input, where every input is linked to each neuron. FC layers are typically located near the end of CNN architectures and are often used to optimize objectives, such as generating class scores.

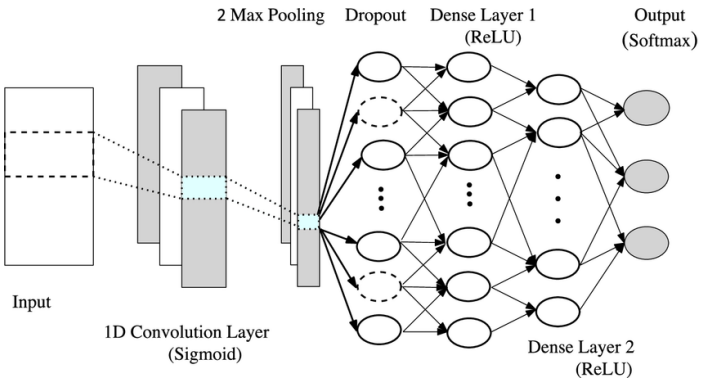

Figure 2: Neural Network

2.1. Filter Hyperparameters

Now we shall discuss filter hyperparameters. There are mainly three types of hyperparameters.

- **Dimensions of a filter** - A filter of size $F \times F$ is applied to an input containing C channels, resulting in output feature map of size $O \times O \times 1$.

- **Stride** - In a convolutional or pooling operation, the stride S refers to the number of pixels the window shifts after each computation.
- **Zero-padding** - Zero-padding denotes the process of adding P zeroes to each side of the boundaries of the input.

2.2. Activation Functions

Activation functions are used in neural networks to introduce non-linearity, enabling the model to learn complex patterns. Some commonly used activation functions are as follows.

- **Rectified Linear Unit (ReLU)** - The Rectified Linear Unit (ReLU) layer is an activation function g applied to each element of the input volume. It introduces non-linearity to the network, allowing it to learn more complex patterns. We can write
$$g(z) = \max(0, z).$$
- **Softmax** - The Softmax function can be viewed as an extension of the logistic function, which takes a vector of scores $\mathbf{x} \in \mathbb{R}^n$ as input and transforms it into a probability distribution vector $\mathbf{p} \in \mathbb{R}^n$. This transformation is applied at the final step of the network architecture. The Softmax function is defined as
$$\mathbf{p}_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \text{for} \quad i = 1, 2, \ldots, n.$$
- **Sigmoid** - Maps input to a range between 0 and 1. It's often used for binary classification but can suffer from vanishing gradients. We can write
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

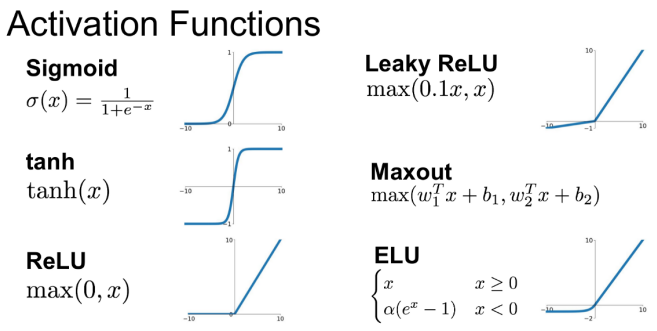

Figure 3: Activation Functions

3. Methodology

3.1. Data Collection

For this study, the dataset used consists of plant images sourced from the New Plant Diseases Dataset (Augmented) available on Kaggle. This dataset includes images of healthy and diseased plants, specifically targeting different crop diseases. The data was loaded using *tf.keras.utils.image_dataset_from_directory*, which reads the images from a directory structure where each subdirectory corresponds to a different disease class. The dataset was split into training and validation sets, with each image resized to a consistent shape of 128×128 pixels for uniformity. Here are the key parameters used for loading the dataset:

- **Labels**- Labels are inferred from the directory structure where each subdirectory name represents the class label. These labels are encoded categorically.
- **Batch Size**- A batch size of 32 was used to process the data in chunks during training.
- **Image Size**- The images were resized to 128×128 pixels to standardize the input size for the neural network.
- **Color Mode**- The images were loaded in RGB color mode, retaining color information for better disease recognition.
- **Shuffle**- The data was shuffled to ensure that the model sees a diverse range of images during training, promoting generalization.
- **Interpolation**- Bilinear interpolation was used to resize the images, maintaining image quality during transformation.

3.2. Model Architecture

The model follows a typical Convolutional Neural Network (CNN) architecture, consisting of multiple convolutional layers for feature extraction, max-pooling layers for downsampling, and fully connected layers for classification. The architecture is as follows:

- **Input layer**- The input image size is $128 \times 128 \times 3$, corresponding to the height, width, and number of color channels (RGB).
- **Convolutional layers**-
  1. The first convolutional block consists of two 3×3 filters with 32 filters each, followed by a max-pooling layer with a pool size of 2×2. The activation function used is ReLU.
  2. The second block consists of two 3×3 convolutional layers with 64 filters, followed by a 2×2 max-pooling layer.
  3. The third block follows the same pattern but uses 128 filters.
  4. The fourth block again follows the same pattern, but the number of filters increases to 256.
  5. The fifth and final block of convolutions contains 512 filters for feature extraction, again with max-pooling layers.
- **Dropout layers**- Dropout layers with rates of 0.25 and 0.4 are added after the convolutional layers and the fully connected layers, respectively, to prevent overfitting.
- **Fully Connected layers**-

1. The flattened output of the convolutional layers is passed to a dense layer with 1500 units and ReLU activation. This layer combines the features learned from the convolutional layers.
2. The final fully connected layer has 38 units (corresponding to the number of classes) and uses a softmax activation function to output a probability distribution over the 38 classes.

Summary of model:

- **Convolutional layers**- 5 blocks, with increasing numbers of filters (32, 64, 128, 256, 512), using 3×3 filters and ReLU activations.
- **Pooling layers**- Max-pooling layers with a 2×2 pool size and stride 2.
- **Dropout layers**- Two dropout layers with rates of 0.25 and 0.4 for regularization.
- **Fully Connected layers**- One dense layer with 1500 units and ReLU activation, followed by another dense layer with 38 units and a softmax activation for classification.

```
# Initialize the Sequential model
cnn = Sequential()

# Define the CNN architecture
cnn.add(Conv2D(filters=32, kernel_size=3, padding='same', activation='relu', input_shape=(128, 128, 3)))
cnn.add(Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(MaxPool2D(pool_size=2, strides=2))

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

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

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

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

# Add Dropout and Fully Connected Layers
cnn.add(Dropout(0.25))
cnn.add(Flatten())
cnn.add(Dense(units=1500, activation='relu'))
cnn.add(Dropout(0.4))
cnn.add(Dense(units=38, activation='softmax'))

# Display the model summary
cnn.summary()
```

Figure 4: Model summary

3.3. Training our model

In the training process, the model is trained using the *fit()* method, which takes the training dataset and validation dataset as inputs, along with the number of epochs to run. Here's a breakdown of the steps involved in training the CNN model:

- **Data Feeding**- The model is provided with two datasets: the *training_set* and *validation_set*. The *training_set* contains the labeled images that the model uses to learn and adjust its internal parameters, while the *validation_set* is used to evaluate the model's performance at the end of each epoch to monitor overfitting.
- **Epochs**- The model will undergo 10 epochs. An epoch refers to one full pass through the entire training dataset. After each epoch, the model is evaluated on the validation data to check for performance metrics like accuracy. If the model performs well on the validation data, it's an indication that it is generalizing well and not overfitting to the training data.
- **Optimizer**- The Adam optimizer is used with a learning rate of 0.0001. Adam is an adaptive learning rate optimization algorithm that computes individual learning rates for different parameters, adjusting the learning rate based on the historical gradients. The learning rate of 0.0001 ensures that the model learns at a slower pace, which can be beneficial for preventing the model from overshooting the optimal point.
- **Loss function**- The loss function chosen is *categorical_crossentropy*, which is commonly used in multi-class classification problems. This loss function calculates the difference between the true labels and the predicted output, guiding the model on how to adjust its weights to minimize this error.
- **Metrics**- The model tracks accuracy as its evaluation metric during training. Accuracy measures the percentage of correct predictions made by the model, providing an indication of how well the model is performing in terms of classification.
- **Training**- During training, the model adjusts its internal weights using backpropagation and gradient descent. For each training batch, the optimizer updates the weights to minimize the loss, and this process repeats for each epoch. At the end of each epoch, the model's performance on the validation set is calculated, helping in assessing if the model is overfitting or underfitting.

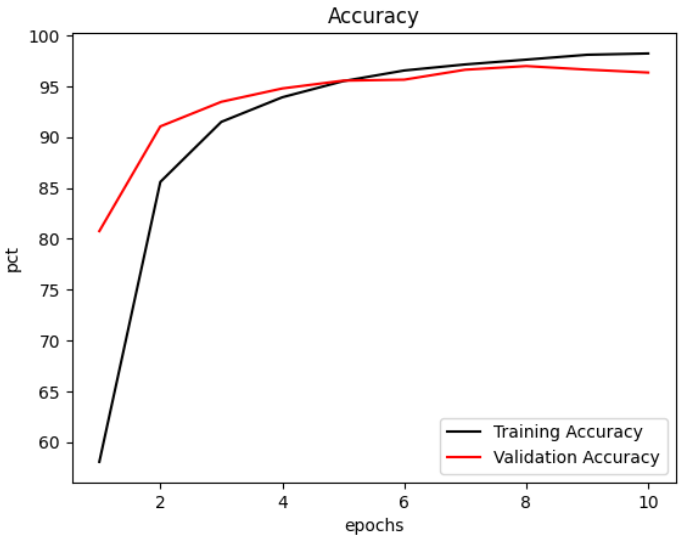

Figure 5: Model accuracy v/s epoch

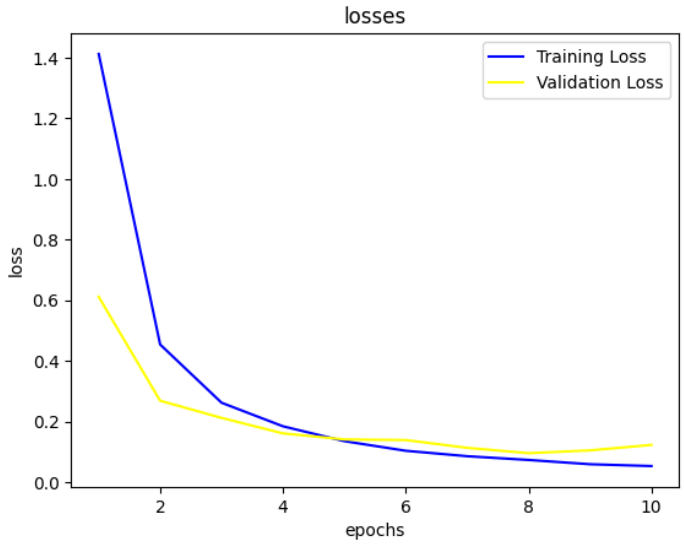

Figure 6: loss v/s epoch

3.4. *Evaluation of model*

Key metrics used in evaluation are:

- **Precision**- Measures the proportion of true positives out of all positive predictions (how many selected items are relevant).
- **Recall**- Measures the proportion of true positives identified out of all actual positives (how many relevant items are selected).
- **F1-Score**- Harmonic mean of precision and recall, providing a balance between the two metrics.
- **Support**- The number of true instances for each class in the dataset.

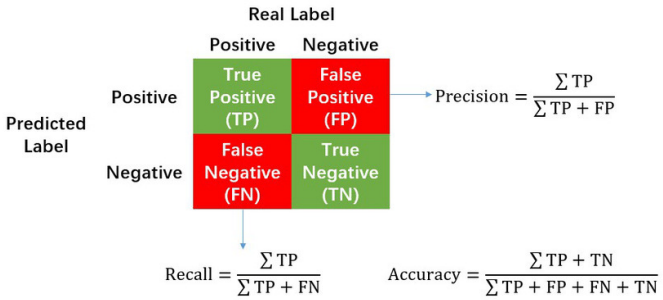

Figure 7: Visualization of Precision and Recall scores

Highlights of report are:

- Several classes, like *Apple___healthy*, *Cherry___healthy*, *Corn___healthy*, and *Grape___healthy*, show very high precision, recall, and F1-scores (above 0.95). This indicates that the model performs exceptionally well in identifying healthy crops.
- The *Apple___Black_rot* class has the lowest F1-score (0.87), primarily due to lower precision (0.78). This suggests the model struggles with accurately identifying this disease, possibly confusing it with other classes.
- Most classes maintain a good balance between precision and recall, reflected in high F1-scores across the board.
- **Accuracy**- The model achieves an overall accuracy of 96%, indicating its robustness in classifying the dataset correctly.
- **Macro Average**- Precision, recall, and F1-score are all 0.97, indicating that the model is well-calibrated across all classes without being skewed by the support of individual classes.
- **Weighted Average**- Similar to macro averages but takes into account the support of each class. The values here also hover around 0.96, showing consistency in model performance.

	precision	recall	f1-score	support
Apple___Apple_scab	0.95	0.87	0.91	504
Apple___Black_rot	0.78	1.00	0.87	497
Apple___Cedar_apple_rust	0.98	0.96	0.97	440
Apple___healthy	0.98	0.92	0.95	502
Blueberry___healthy	0.99	0.91	0.94	454
Cherry_(including_sour)___Powdery_mildew	0.99	0.97	0.98	421
Cherry_(including_sour)___healthy	1.00	0.98	0.99	456
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	0.94	0.93	0.93	410
Corn_(maize)___Common_rust_	0.99	0.99	0.99	477
Corn_(maize)___Northern_Leaf_Blight	0.95	0.97	0.96	477
Corn_(maize)___healthy	1.00	0.99	0.99	465
Grape___Black_rot	0.96	0.99	0.97	472
Grape___Esca_(Black_Measles)	0.98	0.99	0.99	480
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	1.00	0.97	0.99	430
Grape___healthy	0.99	0.99	0.99	423
Orange___Haunglongbing_(Citrus_greening)	0.99	0.97	0.98	503
Peach___Bacterial_spot	0.94	0.97	0.96	459
Peach___healthy	1.00	0.97	0.98	432
Pepper,_bell___Bacterial_spot	0.91	0.94	0.93	478
Pepper,_bell___healthy	0.94	0.97	0.95	497
Potato___Early_blight	0.99	0.97	0.98	485
Potato___Late_blight	0.97	0.96	0.96	485
Potato___healthy	0.98	0.95	0.97	456
...				
accuracy			0.96	17572
macro avg	0.97	0.96	0.96	17572
weighted avg	0.97	0.96	0.96	17572

Figure 8: Precision, Recall, F1 scores of model

3.5. *Saving our model*

Keras provides the *save()* function to store the trained model in various formats, including the .h5 format. This method saves the entire model, including the architecture, weights, and optimizer state, which allows the model to be loaded later for inference or further training.

3.6. *Deployment*

The model was deployed to a user-friendly interface to make it accessible for practical use by farmers and agricultural experts.

- **Streamlit Application**- The deployment was carried out using Streamlit, a Python-based framework that allows for the rapid creation of web applications. The application was designed to take user-uploaded images of plants and predict whether the plants are diseased or healthy, along with the specific disease, if applicable.

- **Hosting the model**- The application was hosted on a web server, allowing farmers and agricultural experts to access the model remotely through their browsers. Streamlit simplifies deployment by providing an easy-to-use interface for running the model in a production environment.

Figure 9: Application interface

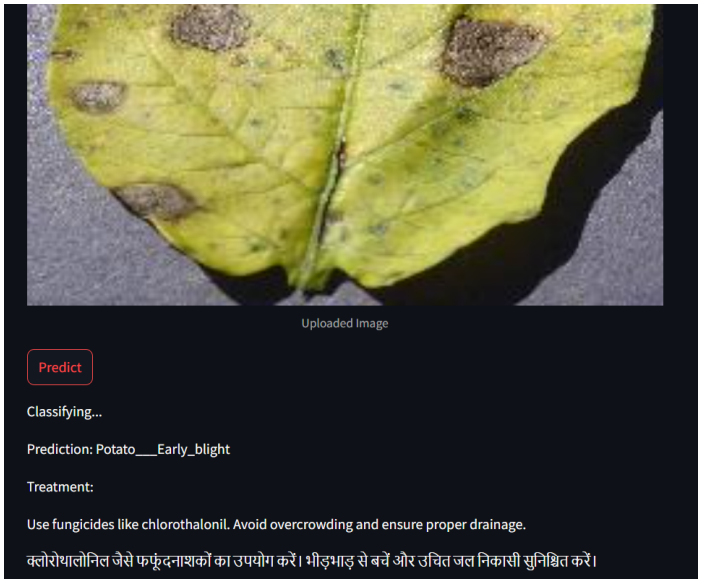

Figure 10: Application predicting the disease and then telling the treatment

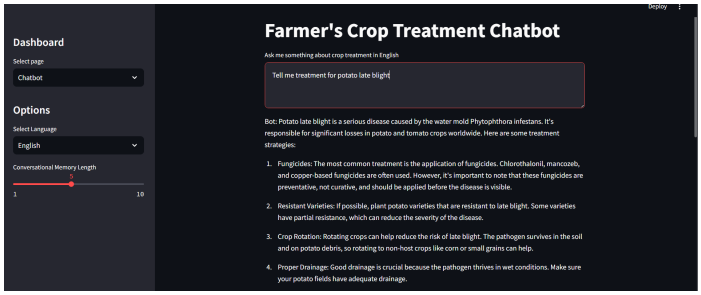

Figure 11: Chatbot for aftermath

4. Discussion

1. Strengths

- The model achieved an impressive accuracy of 96% on the test dataset, which is indicative of its robustness in identifying crop diseases. This performance is attributed to the effective use of Convolutional Neural Networks (CNNs) for image classification, which allows the model to learn complex patterns in the images of plant leaves.
- Traditional methods of crop disease detection are time-consuming and require expert knowledge. On the other hand, the use of AI and machine learning, especially CNNs, enables faster, more accurate, and scalable disease detection, which is a significant advantage in precision agriculture.
- The ability of the model to handle large datasets and make real-time predictions opens up opportunities for large-scale deployment in precision agriculture. This could empower farmers to diagnose diseases on the spot, improving response times and potentially reducing crop losses.
- The Streamlit-based application developed as part of this study provides a user-friendly interface that can be easily used by farmers, even those with limited technological expertise. By uploading images of their plants, farmers can receive real-time disease diagnoses, enabling quicker intervention and potentially saving crops from further damage.
- The use of mobile applications for disease detection reduces the cost of diagnostics by eliminating the need for expensive equipment and expert consultation, thus providing an affordable solution for smallholder farmers.

2. Weaknesses

- One of the major challenges faced by the model is the similarity in leaf appearance across different plant species or varieties. Some crops, despite being different species, may have similar-shaped leaves, making it difficult for the model to distinguish between them, particularly when they are affected by diseases that result in similar symptoms.
- Another limitation arises from the fact that many diseases exhibit overlapping symptoms. For instance, leaf spot diseases can appear similar in many crops, even though they are caused by different pathogens. This can result in misclassification, especially in cases where the symptoms are subtle or not fully developed.
- The model's performance could be affected by environmental factors such as lighting conditions, background noise, or variations in image quality, especially when images are captured under uncontrolled field conditions. These factors may result in misclassifications or lower accuracy when deployed in real-world scenarios.

3. Future Improvements

- To improve the model's robustness in real-world environments, future work could focus on incorporating noise into the training process. By adding noise, such as random pixel distortions, changes in lighting

conditions, or background clutter, the model could become more adaptable to diverse environmental factors, making it more reliable when deployed in less-controlled conditions.

- Future work could explore more sophisticated neural network architectures to focus on more subtle symptoms and fine-grained details in the images. These models could detect early-stage symptoms or minor disease indicators that are often overlooked by standard CNNs, leading to better accuracy in real-world applications.
- Expanding the dataset to include a broader range of plant species, diseases, and environmental factors would improve the model's ability to generalize to new regions and conditions. Including images from different geographic locations and under varying environmental conditions could help the model better adapt to real-world applications.

5. Summary and conclusions

In this research, we presented an approach for detecting and classifying plant diseases using Convolutional Neural Networks (CNNs) and deep learning techniques. The model was trained on the "New Plant Diseases Dataset (Augmented)" from Kaggle, which contains images of plants suffering from various diseases. The CNN model was designed with multiple convolutional and pooling layers, followed by fully connected layers to perform disease classification. The model was trained on a dataset of resized images (128×128 pixels) and evaluated on its ability to predict plant disease categories.

The model achieved satisfactory accuracy, demonstrating the potential of using deep learning models for plant disease detection. Key steps in the methodology included data collection from a publicly available plant disease dataset, pre-processing the data to standardize input images, model architecture design, training using an Adam optimizer, and saving the trained model for deployment. The evaluation metrics used included accuracy, which indicated the model's performance in distinguishing between healthy and diseased plant images across multiple classes. The model showed promising results in identifying diseases, although further improvements are needed for handling challenging cases like early-stage diseases and distinguishing similar-looking diseases.

This study demonstrated the feasibility of using deep learning for plant disease detection, particularly through the use of CNNs. The model successfully classified plant diseases with high accuracy, providing an automated solution that could assist farmers and agricultural experts in identifying crop diseases. The approach showcased the power of CNNs in image-based tasks and confirmed that machine learning could play a pivotal role in precision agriculture.

However, several challenges remain, including handling noisy real-world data, detecting early-stage diseases with subtle symptoms, and dealing with diseases that share similar symptoms. Further research is required to enhance the robustness of the model, especially in terms of handling data variability such as changes in environmental conditions, lighting, and scale. In addition, integrating temporal data or using transfer learning could improve generalization across different plant species and regions.

In the future, improvements in data diversity, model complexity, and user interface design could lead to more reliable and accessible plant disease detection systems. By incorporating these enhancements, the system could become a valuable tool for farmers, enabling them to take timely action to mitigate crop losses and improve agricultural productivity.

Acknowledgements

I would like to express my gratitude to Professor S. Dharmaraja and Postdoctoral Fellow Raina Raj for their invaluable assistance and guidance throughout the course of this project.

References

- **CNNs**
  - Click Here
- **Activation Functions**
  - Click Here
- **Neural Networks**
  - Click Here
- **Key metrics of evaluation**
  - Click Here