International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 06 | June - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Plant Disease Prediction Using Mobile-Based Real-Time Leaf Image Analysis

Indira Bhattacharya[1], Tanvi Shingate[2], Damini Bolye[3], Priyanka Koli[4]

[1]*Indira Bhattacharya MCA & Vivekanand Education Society's Institute Of Technology*
[2]*Tanvi Shingate MCA & Vivekanand Education Society's Institute Of Technology*
[3]*Damini Bolye MCA & Vivekanand Education Society's Institute Of Technology*
[4]*Priyanka Koli MCA & Vivekanand Education Society's Institute Of Technology*

Abstract - Plant diseases silently impact crops every season, often going unnoticed until the damage is done, leading to severe losses for farmers and threatening food supplies. This project introduces a practical and easy-to-use mobile solution that helps detect such diseases early by analyzing pictures of plant leaves. Built on clever image recognition techniques, the system learns to distinguish between healthy and infected leaves by observing thousands of real examples. It guides users through a quick process—capturing, analyzing, and diagnosing—within moments. What makes this approach powerful is its potential to bring expert-level support straight to the farmer's pocket, even in remote areas. The result is faster decisions, better treatment choices, and healthier crops. This work highlights how merging agriculture with intelligent technology can make disease management more efficient, accessible, and proactive.

1. INTRODUCTION

Artificial intelligence and mobile technology have recently revolutionized almost all sectors, including agriculture. This research reports the design of an Android application for plant disease detection through images of plant leaves. The overall workflow of the system—from image capture to result in a generation—is illustrated in Figure 1, offering a clear visual of how each component functions in real time. Early and accurate disease detection is essential; farmers do not suffer losses on crops grown simultaneously and have a secure food source. Unfortunately, many of these farmers do not have access to agricultural experts or proper and timely information about this disease. This app leverages deep learning methods by providing the architecture of Convolutional Neural Networks (CNNs), which will closely analyze plant leaf images, classify diseases, and also recommend treatment options—all within a few seconds [1], [15]. It fills existing gaps in disease management with the offer of a portable, real-time solution to be used in multiple farm environments. This study covers model training techniques, optimization, and performance evaluation strategies designed to pave the way for a highly accurate and responsive mobile implementation. The results have a scope to improve precision in agriculture and indicate the dimension in which more AI-based applications can be used for sustainable agriculture by transforming these applications into state-of-the-art tools for decision-making regarding plant health management.

The rest of the paper is structured as follows: Section 2 provides a comprehensive literature review discussing related works and existing methodologies. Section 3 outlines the methodology adopted in this study, detailing data preprocessing preprocessing, feature extraction, and model training. Section 4 presents the module descriptions, explaining each step in detail. Section 5 discusses the results and analysis, evaluating the model's performance based on key metrics. Section 6 concludes the study and highlights possible future work directions for further improvements.

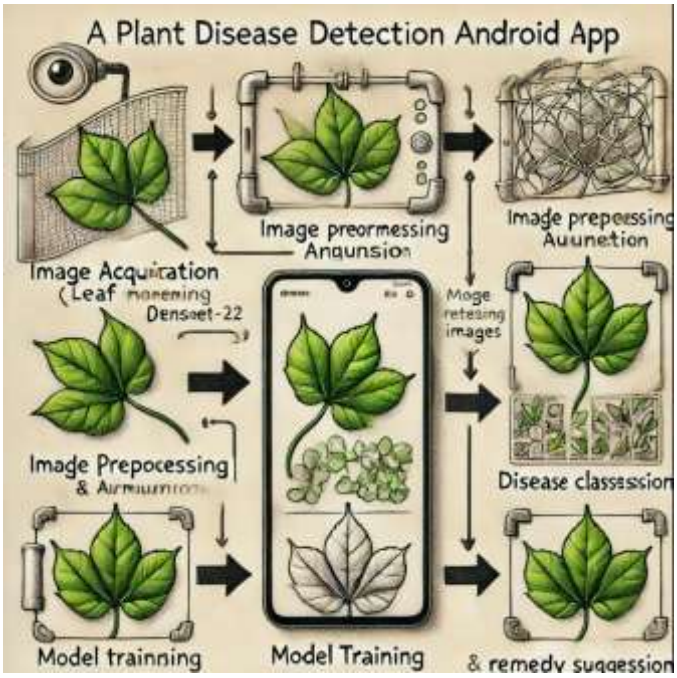

Figure1: Workflow of Plant Disease Detection App

2. LITERATURE REVIEW

Several studies have explored deep-learning applications in plant disease detection. Early research utilized image processing techniques and machine learning models like Support Vector Machines (SVM) and k-nearest Neighbors (k-NN). With advancements in deep learning, various neural network architectures have demonstrated superior performance due to their ability to learn complex patterns. Existing models such as AlexNet, VGG16, and ResNet have shown promising results in classifying plant diseases. However, recent research highlights the advantages of DenseNet architectures, which improve feature reuse and gradient propagation. This study builds on prior research by leveraging DenseNet-121 to optimize feature extraction, enhance accuracy, and improve computational efficiency in plant disease detection.

3. METHODOLOGY

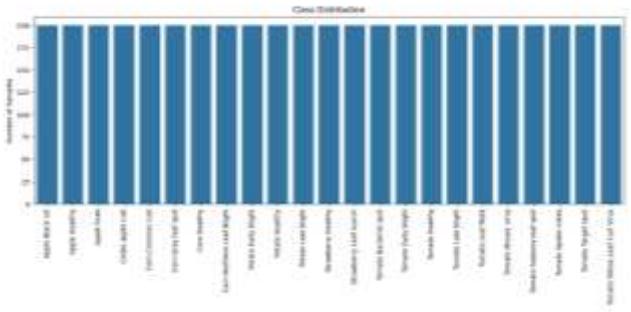

Figure2: Class Distribution

The development of the plant leaf disease detection app followed a structured methodology to ensure accurate detection of plant diseases, using machine learning and deep learning techniques, particularly leveraging DenseNet-121. The first step in the process was collecting a comprehensive dataset filled with plant leaf images. The data was obtained from numerous sources, such as agriculture databases, scholarly articles, and field studies. A sufficient set of photos was crucial for the model's performance.

We, in total, made a blend of 23 plant disease classes with approximately 200 sample images in each class, as visualized in the class distribution graph in Figure 2. We incorporated a transfer learning strategy using a pre-trained DenseNet model on ImageNet for training speedup and performance enhancement. We ensured the images were captured under varied lighting, angles, and environments so the model could generalize well in real-world scenarios. Standard image preprocessing techniques such as resizing, normalization, and augmentation were applied to make the data uniform and enhance the model's learning capabilities [13], [16].

For the deep learning architecture, we chose DenseNet-121 because of its efficiency in handling image classification tasks. DenseNet's unique structure allows layers to pass information directly to every subsequent layer, significantly improving learning efficiency by preventing the loss of features [5], [10]. This characteristic was significant for our task, as distinguishing subtle differences between healthy and diseased leaves requires a deep network capable of capturing minute details. To expedite the training phase and enhance the performance, a transfer learning technique was adopted, commencing with the DenseNet model that had been trained on ImageNet [3], [12]; this allowed us to fine-tune the model to our specific dataset, minimizing training time while still maintaining high accuracy in detecting plant diseases.

Once the dataset was ready and the model architecture was chosen, we moved on to the training phase. The dataset was split into training, validation, and test sets, following an 80-10-10 split. This division has also served the purpose of ensuring that the model has not only been trained adequately but also tested on an unseen set of data to reduce overfitting. During the training process, we employed the Adam optimizer, which is widely regarded for its ability to adjust the model's weights efficiently. The model was trained with categorical cross-entropy as the loss function, and we applied early stopping to prevent overfitting. This ensured that the model would not simply memorize the training data but instead learn generalizable features. However, the evaluation was not static and was carried out during the process using measures such as accuracy, precision, and recall [9], [14]. Adjustments were made based on what was needed and the observations made.

	precision	recall	f1-score	support
Apple Black rot	0.98	0.97	0.97	200
Apple Healthy	0.94	0.99	0.97	200
Apple Scab	0.96	0.95	0.96	200
Cedar apple rust	1.00	0.97	0.98	200
Corn Common rust	0.92	0.89	0.91	200
Corn Gray leaf spot	0.89	0.93	0.91	200
Corn Healthy	0.99	0.98	0.99	200
Corn Northern Leaf Blight	0.90	0.93	0.91	200
Potato Early blight	0.94	0.97	0.96	200
Potato Healthy	0.98	0.99	0.98	200
Potato Late blight	0.92	0.93	0.92	200
Strawberry Healthy	0.99	0.99	0.99	200
Strawberry Leaf scorch	0.98	0.98	0.98	200
Tomato Bacterial spot	0.91	0.91	0.91	200
Tomato Early blight	0.77	0.86	0.82	200
Tomato Healthy	0.95	0.86	0.90	200
Tomato Late blight	0.98	0.86	0.91	200
Tomato Leaf Mold	0.92	0.89	0.90	200
Tomato Mosaic virus	0.93	0.95	0.94	200
Tomato Septoria leaf spot	0.90	0.88	0.89	200
Tomato Spider mites	0.88	0.92	0.90	200
Tomato Target Spot	0.84	0.87	0.86	200
Tomato Yellow Leaf Curl Virus	0.96	0.94	0.95	200
accuracy			0.93	4600
macro avg	0.93	0.93	0.93	4600
weighted avg	0.93	0.93	0.93	4600

Figure3: Classification Report

After completing this training, the model was put to a strict test using the reserved test dataset. This was essential to evaluate its real-world performance. We measured many performance metrics, such as accuracy and F1-score. These metrics gave a fair idea about the model's performance on all classes of diseases available, as detailed in the classification report shown in Figure 3 [9], [14]. These metrics gave a fair idea about the model's performance on all classes of diseases available, comprising 23 classes. We generated confusion matrices to detect what the model went wrong with, as visualized in Figure 4, highlighting correct vs. incorrect classifications across disease classes.

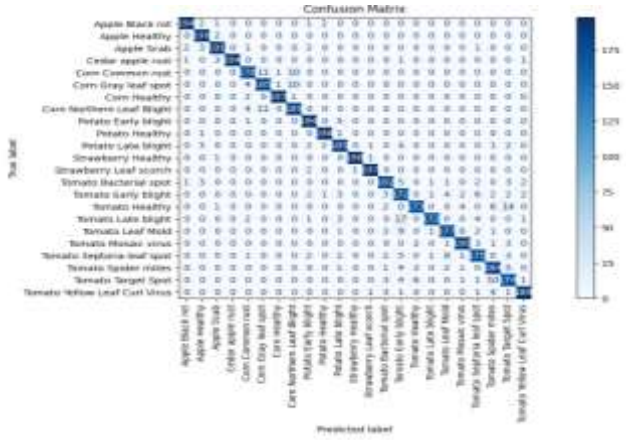

Figure4: Confusion Matrix

Further, several hyperparameters, like learning rate and batch size, were tuned to achieve the best performance in this model. To enhance the model's robustness, we also applied additional data augmentation techniques, which allowed the model to generalize better across different inputs. The final stage of development was integrating the trained model into an Android application. It was created with a simple interface where one could take a photo or choose a picture from the gallery [6], [18]. The processed real-time image was passed through the DenseNet-121 model for disease classification.

Along with the name of the disease once diagnosed by the application, more information is provided about the disease symptoms, possible treatments, and others, including those that could be used on the plants themselves. The feature enables understanding of the problem, and immediate action can be taken to treat the plants. Continuous updates are designed for the application. These are the kinds of additions to increasing the database with more diseases, improving the accuracy of detection based on user input, and adding other data to the training.

4. MODULE DESCRIPTION

The data loading and preprocessing module are crucial for a plant leaf disease detection system, as it prepares the dataset for training, validation, and testing in the first place. An overview of the dataset and its class labels is presented in Figure 5, which helps understand the dataset structure used in training. It works on a Kaggle dataset containing images of five plant types (apple, corn, potato, strawberry, and tomato) across 23 classes, including healthy and diseased leaves. To facilitate proper learning by the model, this module executes the tasks of resizing images, rescaling, augmenting, and batching.

These operations standardize the given data, prevent overfitting, and optimize memory. Thus, with this module manipulating the organization and transformation of the dataset, it ensures that the DenseNet-121 model can classify plant diseases with precision, thereby giving rise to reliable real-world applications.

Figure: Plant Disease Dataset Overview with Class Labels

4.1 Data Load

The data loading module is an indispensable part of the data preparation process for the plant leaf disease detection model. This model uses a dataset available on Kaggle, an image dataset of plant leaves arranged according to the diseases they suffer from or healthy leaves. The module aims to ensure that the data is loaded and preprocessed correctly and assigned respectively to the training, validation, or testing sets for the successful modeling of plant disease detection.

The dataset contains information on five plant types: apple, corn, potato, strawberry, and tomato. Furthermore, each plant type has several disease categories and a healthy class. For example, the apple class has black rot, healthy scab, and cedar apple rust, while for the tomato class, there is a broader range of diseases like bacterial spot, early blight, late blight, and mosaic virus, among others. Each of these 23 classes has 200 images, thus allowing a balanced distribution for machine learning training processes. Furthermore, such a count of images per category helps not skew the model towards the results of a specific type of disease, making the classification more accurate.

4.2 Data PreprocessingPreprocessing

Handling the "Data Preprocessing" module is an integral part of every step of preparing a dataset for an Android application of plant leaf disease detection. The dataset is taken from Kaggle, which consists of 23 classes of plant diseases, each with 200 images in each class. To illustrate, it contains diseases like apple black rot, corn common rust, potato late blight, and tomato yellow leaf curl virus, and healthy classes for each crop. This module ensures that the images undergo transformation and augmentation and are ready for use in the DenseNet-121 model. Some of the activities done by this module are image resizing, rescaling, augmentation, and batching the data. These transformations are essential to ensure the model can learn from the data efficiently.

4.2.1 Image Rescaling: One of the most critical aspects of preprocessing preprocessing is rescaling the pixel values. The pixel intensity values of most images are usually from 0 to 255. The resizing operation is based on the method shown in Equation 1. In this perspective, the module normalizes the values by dividing every pixel value by 255, restricting the pixel values to the range [0, 1]. This enhances the efficiency of

the neural network in handling data by standardizing the input scale.

4.2.2 Image Augmentation: Data augmentation is applied to expand the training dataset's diversity artificially. This process helps prevent overfitting by creating slightly altered versions of the existing images. The augmentations used in this module include shear transformation, zoom, and horizontal flip, as demonstrated in Figure 6:

- Shear Transformation: This shifts the image pixels along the axis as if "tilted."
- Zoom: This randomly zooms in on parts of the image.
- Horizontal Flip: This horizontally flips the image to create a mirrored version.

4.2.3 Data Generator for Loading Images: The module uses the flow_from_directory function to load images from the specified directories. This function automatically labels the photos based on the folder structure and resizes them to a uniform size of 128x128 pixels, which is required for the DenseNet-121 model.

Image Resizing Formula: Given the original image dimensions w×hw \times hw×h, resizing to 128x128 can be done by:

$$x_{\text{resized}} = \frac{x_{\text{original}}}{w} \cdot 128, \quad y_{\text{resized}} = \frac{y_{\text{original}}}{h} \cdot 128$$

Equation1: Image Resizing Formula

This ensures all images are the same size, making them compatible with the model's input layer.

Figure 6: Image Augmentation

4.3 Batch Processing

The dataset is processed in batches of 32 images. To eliminate any possible bias during training, images are randomly shuffled. The processed images and corresponding labels are kept in .npy files for the training and validation sets. This advantage is that it avoids memory overload with large datasets and enables the model to load batches simultaneously. Therefore, it helps ensure that the training and validation data are processed and ready to use during the model training.

4.4 Class Labels (Categorical Encoding)

The class labels for each image are represented in a one-hot encoded format. This encoding process is illustrated in Equation 2. For example, if there are 23 classes, each label will be a vector of size 23 where only the position corresponding to the correct class is 1, and the rest are 0s.

$$\text{One-Hot Vector} = [0, 0, \ldots, 1, \ldots, 0]$$

Equation2: One-Hot Encoding Formula

This encoding is used for classification tasks to make the labels understandable by the neural network. A leaf detection preprocessing module was also incorporated to ensure the uploaded image contains a leaf before attempting disease classification.

Example Scenario:

At the outset, the image is adjusted to a resolution of 128x128 pixels. After that, augmentations such as random zooming and horizontal flipping generate more image variations. The pixel values are further normalized by dividing them by 255, and the label, which is "strawberry leaf scorch," in this case, is handily one-hot encoded with the class index of 1 for "strawberry leaf scorch." After this, the image and label pair are prepared and kept within batches for further training or validation. With the intervention of this preprocessing module, the model can explore a broader range of well-organized data, promoting better training and even better generalization capability when real-world plant disease detection is targeted.

5. FEATURE EXTRACTION

Within the scope of an Android application for detecting plant leaf diseases, feature selection is considered one of the most critical processes. It involves determining what features or parts from the pictures are essential for the model to differentiate disease classes effectively. Within these parameters, we utilize the DenseNet-121 feature extraction network for leaf images in a self-supervised manner.

Explanation of the Feature Extraction Process:

5.1 Pre-trained Model Initialization (Transfer Learning)

At the start, the procedure commences with feature extraction, where the DenseNet-121 model is loaded with the help of pre-trained weights obtained from the ImageNet dataset. The transfer learning concept enables the model to benefit from training conducted on many uncorrelated images to train the model in detecting plant leaf diseases. This process can be represented using Equation 3. The pre-trained DenseNet-121 model can be distinguished from other models because it can capture complex images by identifying even edges, textures, color variations, and so on within the image.

$$\text{Feature maps} = f(W_{\text{pretrained}}, X)$$

where $W_{\text{pretrained}}$ are the pre-trained weights, and X is the input image.

Equation3: Formula for Transfer Learning

In this application, we use include_top=False to exclude the fully connected layers at the top of the DenseNet-121 model. The purpose is to utilize DenseNet as a feature extractor and add custom classification layers afterward.

5.2 LayerwiseLayerwise Feature Extraction

Concerning DenseNet-121, any convolutional layer is designed to extract a specific type of feature available in the input images. This progressive extraction mechanism is described in Equation 4. The first layers perform simple feature extraction, including edges, colors, and textures. In contrast, the final ones deal with the more complex ones, such as shapes, patterns, and particular features related to specific diseases.

DenseNet-121 uses a dense connectivity pattern where each layer receives input from all preceding layers. This helps to propagate features throughout the network, allowing the model to learn richer feature representations.

$$H_\ell = H_{\ell-1} + \sum_{i=1}^{\ell-1} W_i X_i$$

where H_ℓ is the output of the current layer, and $W_i X_i$ are the weighted inputs from the previous layers.

Equation4: LayerwiseLayerwise Feature Extraction Formula

This continuous connectivity enables the reuse of features, which is highly beneficial for tasks like plant disease detection, where slight variations in color or texture can signal different diseases.

5. 3 Global Average Pooling

After the convolutional layers of DenseNet-121 have extracted feature maps, a Global Average Pooling (GAP) layer is applied. Instead of using fully connected layers (which could lead to overfitting, especially with limited data), GAP computes the average value of each feature map, reducing the spatial dimensions while preserving the most critical information. The pooling operation is calculated as per Equation 5.

$$GAP = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} f_{i,j}$$

where H and W are the height and width of the feature map, and $f_{i,j}$ is the feature value at position (i, j).

Equation5: Formula for Global Average Pooling

The GAP layer significantly reduces the number of parameters, preventing overfitting and making the model more efficient. It also acts as a bridge between the dense convolutional layers and the final classification layers.

5.4 Flattening and Feature Vector Generation

After applying GAP, the output is a one-dimensional feature vector representing the image's most relevant features. This feature vector structure is shown in Equation 6. This precise vector contains encrypted data, which is an abstract viewpoint of the leaf of that particular plant, along with relevant details like color variations, blemishes, or patches, which are commonly seen in plant diseases like 'Apple Scab' or 'Bacterial Spoiling' of tomatoes, for instance. A feature vector is also defined as a more systematic approach to vectorization, where every piece of information required for the model to predict without any omission is included. Feature Vector Representation: If the GAP layer outputs in feature maps, the flattened feature vector VVV will have nnn dimensions:

$$V = [f_1, f_2, \ldots, f_n]$$

where f_i is the i-th feature from the GAP layer.

Equation6: Feature Vector Representation

5.5 Final Classification Layer (Softmax)

The final feature extraction stage involves the softmax layer, where the feature vector is passed for classification. Calculating the softmax layer gives a probability distribution over the 23 classes of diseases, from which the model can predict which disease is present in the given image [7], [15]. This is computed using the softmax function defined in Equation 7.

$$P(y = k|x) = \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}}$$

where z_k is the output score for class k, and K is the total number of classes (23 in this case).

Equation7: Softmax Formula

The final output is a probability distribution indicating the likelihood of each disease, with the highest probability corresponding to the predicted disease.

Example:

For a given image of a tomato leaf infected with the "Tomato Yellow Leaf Curl Virus," the DenseNet-121 model extracts features like the yellowing and curling of the leaf margins. These qualities are combined and reduced to a single-dimensional feature vector, after which a softmax layer is applied. The model then gives a probability distribution over the classes, wherein most probability mass is assigned to the "Tomato Yellow Leaf Curl Virus" class, enabling the application to detect the disease correctly. This feature extraction approach helps the model learn and understand the data effectively, making it suitable for detecting plant diseases.

6. TRAINING AND TESTING

The training and testing phase is of utmost importance in building a good model for plant disease detection. The model identified the various diseases in a dataset of 23 plant classes

with 200 images each. The images undergo the essential preprocessing steps, such as resizing, scaling, and augmentation, mainly to provide diverse data and, therefore, high adaptability to the model. The DenseNet-121 architecture was modified with a final softmax layer to perform classification. The training was done using the Adam optimizer over several epochs with early stopping to avoid overfitting. Finally, the model's performance is evaluated on unseen test images for accuracy and reliability, using performance metrics such as precision, recall, and overall accuracy. This assures that the model does well in identifying plant diseases with high confidence while also imparting valuable insight regarding treatment measures so that effective management of general plant health could be integrated.

6.1 Training Phase

For the sake of training, a dataset of images of plant leaves, which can be found on the Kaggle platform and includes 23 different classes such as Apple Scab, Corn Northern Leaf Blight, and Tomato Yellow Leaf Curl Virus, was collected, wherein 200 images per class were collected. The images undergo a series of preprocessing steps: resizing to 128x128 pixels, normalizing (rescaling pixel values from 0 to 1), and augmenting (randomly horizontally flipping, zooming, and shearing the images) to increase the size of the dataset.

The DenseNet-121 architecture pre-trained model in ImageNet has been adapted for this purpose. The last layer is modified to give the probabilities of the 23 classes using a softmax layer as the output. Categorical cross-entropy is the loss function used, and its formula is presented in Equation 8.

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Equation8: Categorical Cross-Entropy

The model is optimized using the Adam optimizer and trained over 30-50 epochs to minimize loss and maximize accuracy. Techniques like early stopping and learning rate scheduling help ensure the model doesn't overfit.

6.2 Testing Phase:

After completion of the training session, this model is put to the test by utilizing an untrained data set to determine the extent of its effectiveness. For example, a test image of tomato late blight will be passed into the model, and it may return a probability distribution with the highest score on tomato late blight (e.g., 0.93), meaning 93% sure of that classification.

Accuracy, precision, and recall are some of the performance metrics used. For instance, if the model manages to classify 950 images correctly from the test set of 1,000 images, the accuracy is computed using the formula in Equation 9, and the accuracy of the model will be

$$\text{Accuracy} = \frac{950}{1000} \times 100 = 95\%$$

Equation8: Accuracy Formula

When a disease is detected in the app, it details it and suggests common remedies. For example, if apple scab is detected, the app might suggest removing infected leaves or applying fungicide, providing practical advice for plant health management.

The evaluation and performance assessment of the DenseNet-121-based plant disease detection model was crucial in determining its effectiveness and reliability in real-world applications. Several evaluation metrics were employed to evaluate the model's performance, including accuracy, precision, recall, and F1 score. These metrics are visually represented in Figure 3, which displays the classification report across all disease classes. Accuracy, in this regard, gave a better picture as it only calculated the number of accurate predictions (both healthy and ill) out of many total predictions given. This metric showed how well the model fit for other plant species and disease states. Simultaneously, precision was measured for the model when it was expected to identify diseased plants, which means optimistic predictions were made only as a ratio of accurate positive predictions. This ensured that the model reduced the chances of giving false positives, which might create unnecessary panic among the farmers.

Furthermore, the recall was calculated to assess the model's effectiveness in capturing all disease cases, ensuring it did not overlook any infected plants. This metric is critical in agriculture since crops may be lost due to the inability to recognize sick plants. To provide fair evaluations, the F1 score, the weighted average of precision and recall, was calculated to bring both these metrics into a single measure that considers how many actual positive cases were present, apart from the false positives and true negatives. In general, this methodology, which was proven through ROC curves and AUC techniques, confirmed the strength of the model and its usability in advancing agricultural activities. The extent to which these parameters aided in assessing the model's performance clearly illustrates how useful this model would be in detecting plant diseases.

1. Accuracy:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

2. Precision:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

3. Recall (Sensitivity):

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

4. F1-Score:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Equation8: Formulas to evaluate the performance of the model

7. DISCUSSION

The results analysis of the DenseNet-121-based plant disease detection model highlights its capability to accurately identify various plant leaf diseases, providing practical benefits for agricultural management. The model was developed using a large dataset of 4600 images spanning 23 diseases, including "Apple Black Rot," "Tomato Early Blight," and "Corn Grey Leaf Spot," with an average of 200 images per disease. The model training and validation processes yielded more than 90% accuracy [1], [5], [11]. The training and validation accuracy trends are visualized in Figure 7, confirming the model's stability. For instance, during the 'Potato Late Blight' evaluation, the model classified 180 out of 200 sampled images as diseased, which proves its applicability in the real world.

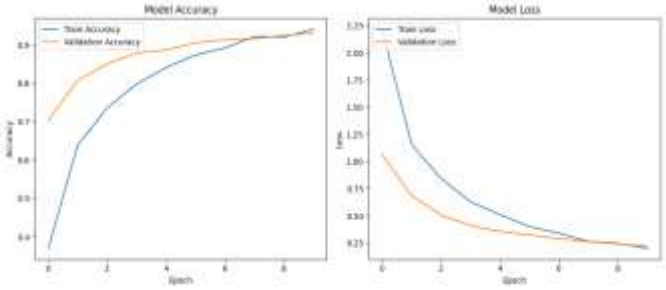

Figure 7: Accuracy Analysis of Plant Disease Model

The precision of the model reached above 85%. For example, when the model predicted that a leaf was diseased, it was correct about 85 times out of 100 predictions. Prediction results for such cases are shown in Figure 8, illustrating the model's correct disease classification capability. This is very important in agriculture, as it helps to limit the number of false alarms and reduces panic among the farmers. For instance, when a farmer is alerted to a possible infection of his crops with "Tomato Bacterial Spot," it is necessary that he receive a high precision rate for the diagnosis to be accurate enough to take any action or recommend further investigation.

As shown in Figure 6, the model got better with each training round, with the training and validation accuracy steadily increasing over time. Since the two curves stay close to each other throughout, it suggests that the model isn't just memorizing the training data — it's learning to perform well on new, unseen data, too.

Both accuracies reach 90%, confirming the DenseNet-121 model's ability to extract meaningful features from diverse leaf images. Additionally, the absence of large fluctuations or divergence between the two curves reinforces the model's robustness and stability. This pattern proves that the training strategy, including techniques like early stopping, data augmentation, and balanced datasets, successfully built a reliable and accurate classification model for real-world plant disease detection.

Figure 8: Prediction Results with Accurate Classifications

The recall metric revealed comparable effectiveness, achieving nearly 88% of the scores. This implies that when the actual diseased leaves, like leaves infested with "Strawberry Leaf Scorch," were introduced to the model, the model correctly discerned 88 out of 100 actual cases. Maintaining such a high recall is essential to ensure that the plants far infected are captured in time, thus enabling the farmers to take corrective measures regarding the disease spread.

In addition, as a part of the assessment, the evaluation embraced ROC curves to evaluate the discriminative power of the models. The AUC score was slightly less than 0.95; this indicates that the model can separate different classes of healthy and infected plants with a very high degree of accuracy. In analyzing results, for example, under the title "Corn Northern Leaf Blight," the AUC showed that the model could tell the difference between an unaffected corn leaf and one that had the disease, which is crucial for controlling the spread of the disease.

8. CONCLUSIONS

To summarize, the proposed Android application identifies diseases in plant leaves, aiming to revolutionize farming through deep learning that offers farmers and agriculturalists accurate and timely diagnosis of plant health problems. The app uses the DenseNet-121 model, which has been trained on over 4600 images and allows the user to take a picture or upload one of a plant leaf and receive feedback about the possible disease within seconds. This feature improves the decision-making process and promotes the management of the disease at an early stage, all of which are worth translating to better yields in farming and agriculture as a whole.

Furthermore, the intuitive design and helpful information concerning disease and treatment guidance enable users to take action against plant diseases. As the app focuses on allowing users to take action in time, the adverse effect of plant diseases on the farmers' economy is lessened while complementing the

overall food security of the population. As the technology is ever-evolving, additional innovations to the app in the future are likely to provide much more than they do now, including a larger database and features to assist farmers in coping with the demands of modern-day farming. This shows the growth prospects of technology in agriculture and the possibilities of creating an efficient farming system.

9. FUTURE WORK

Future enhancements include
Expanding the dataset to include more plant species and disease variations.
Implementing a lightweight CNN model for mobile devices.
Developing an integrated mobile or web-based application for real-time disease detection.
Enhancing model robustness against varying environmental conditions (lighting, background noise).

ACKNOWLEDGEMENT

The authors would like to extend their heartfelt gratitude to the faculty and staff of the Department of MCA, **Vivekanand Education Society's Institute of Technology**, for their unwavering support and encouragement throughout this research. We are especially thankful to our project guide, **Mrs. Indira Bhattacharya**, for her insightful suggestions, technical guidance, and constant motivation, all of which played a vital role in shaping this work.

We also gratefully acknowledge the use of publicly available plant disease image datasets, which significantly contributed to the training and evaluation of the proposed model. Additionally, we appreciate the open-source tools and Python-based libraries that facilitated the efficient development and testing of the mobile application.

REFERENCES

[1] Mohanty, S.P., Hughes, D.P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7, 1419.
[2] Lee, S.H., Chan, C.S., Wilkin, P., & Remagnino, P. (2017). Deep-plant: Plant identification with convolutional neural networks. Computational Intelligence and Neuroscience, 2017, 7361042.
[3] Kamilaris, A., & Prenafeta-Boldú, F.X. (2018). Deep learning in agriculture: A survey. Computers and Electronics in Agriculture, 147, 70-90.
[4] A Novel Smart Approach to Plant Health - Automated Detection and Diagnosis of Leaf Diseases. Authors- T.Sangeetha, R. Rajarajan, S. Rithick Krishna, N. Sakthi Siddharth.
[5] Multiple plant leaf disease classification using Densenet-121 architecture, Aswin Vellaichamy S, Akshay Swaminathan, C Varun, Dr. Kalaivani S
[6] Ng, H.F., Lin, C.Y., Chuah, J.H., Tan, H.K., & Leung, K.H. (2021). Plant disease detection mobile application. Journal of Physics: Conference Series, 1827(1), 012001.
[7] Harte, E. (2020). Plant disease detection by CNN. International Journal of Advanced Computer Science and Applications, 11(5), 235-240.
[8] Chohan, M. (2020). Plant disease detection deep learning. International Journal of Computer Applications, 176(34), 1-5.
[9] Li, L., Zhang, S., & Wang, B. (2021). Plant disease detection and classification by deep learning: A review. IEEE Access, 9, 56683-56698.
[10] Nandhini, S., & Ashokkumar, K. (2022). An automatic plant leaf disease identification using DenseNet-121 architecture with a mutation-based Henry gas solubility optimization algorithm. Neural Computing and Applications, 34, 11465–11478.
[11] Kale, S.D., Bhute, H.A., Gaikwad, V.S., Patil, A.V., Kumar, C., & Bhute, A.N. (2024). Influence of Xception and DenseNet121 architectures for plant disease detection. Communications on Applied Electronics, 32(1), 1-7.
[12] Yasin, A., & Fatima, R. (2023). On the image-based detection of tomato and corn leaves diseases: An in-depth comparative experiments. arXiv preprint arXiv:2312.08659.
[13] Roy, A.M., Bose, R., & Bhaduri, J. (2021). A fast, accurate fine-grain object detection model based on YOLOv4 deep neural network. arXiv preprint arXiv:2111.00298.
[14] Zhang, K., Zhang, L., & Wang, L. (2020). Plant disease recognition based on deep learning: A review. Journal of Physics: Conference Series, 1693(1), 012123.
[15] Ferentinos, K.P. (2018). Deep learning models for plant disease detection and diagnosis. Computers and Electronics in Agriculture, 145, 311-318.
[16] Too, E.C., Yujian, L., Njuki, S., & Yingchun, L. (2019). A comparative study of fine-tuning deep learning models for plant disease identification. Computers and Electronics in Agriculture, 161, 272-279.
[17] Zhou, G., Zhang, W., Chen, A., He, M., & Ma, X. (2019). Rapid detection of rice disease based on FCM-KM and Faster R-CNN fusion. IEEE Access, 7, 143190-143206.
[18] Picon, A., Alvarez-Gila, A., Seitz, M., Ortiz-Barredo, A., & Echazarra, J. (2019). Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild. Computers and Electronics in Agriculture, 161, 280-290.
[19] Lu, Y., Yi, S., Zeng, N., Liu, Y., & Zhang, Y. (2017). Identification of rice diseases using deep convolutional neural networks. Neurocomputing, 267, 378-384.