International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Predictive Models in Agriculture: Combating Crop Diseases with Advanced Data Analytics

S Narayanan[1], S Suresh Kumar[2], D Nisha[3], S Sekar[4]

Submitted:14/03/2024 **Revised**: 29/04/2024 **Accepted**: 06/05/2024

Abstract: This research evaluates the performance of three advanced convolutional neural network (CNN) architectures—Xception, InceptionV3, and VGG19—on an image classification task using transfer learning with pre-trained ImageNet weights. Each model was fine-tuned for a specific dataset and assessed using key performance metrics such as accuracy, precision, recall, and F1-score. To enhance generalization, data augmentation techniques like rescaling, shifting, zooming, and flipping were applied during training, while validation and test data were rescaled for unbiased evaluation. The Xception model achieved an impressive overall accuracy of 95%, performing well across most classes, though it exhibited lower recall for class 7. The InceptionV3 model surpassed Xception, attaining an accuracy of 97%, but encountered difficulties in classifying instances from classes 7 and 8, where precision-recall trade-offs were evident. The VGG19 model, with a total accuracy of 94%, excelled in most classes but showed reduced precision and recall for classes 1 and 7.

Keywords: *Convolutional neural network, Multiple Twin SVM, Support Vector Machines*

1. Introduction

Plants are vital to sustaining life, providing essential nutrition for both humans and animals. Over 80% of the human diet relies on plants, and they form the backbone of livestock nutrition. However, the threat posed by plant diseases and pests significantly undermines food security and agricultural productivity. These challenges not only reduce crop availability but also compromise the quality and safety of the food supply. Annually, global losses in staple crop yields due to diseases and pests can reach up to 30%, translating into economic losses worth hundreds of billions of dollars, further exacerbating food scarcity issues worldwide. [1]

The identification of plant diseases is crucial to humanity, as plants provide essential resources such as food, furniture, clothing, environmental benefits, and even housing materials. For over 8 billion people worldwide who rely on plant-based products for survival, the impact of plant diseases can be profound. These diseases can mean the difference between a life of comfort and stability or one marked by hunger and, in extreme cases, death due to starvation. [2]

However, the subsistence of diseases on plants and crops especially on leaves has reduced the quality of agricultural food products. These severe leaf diseases can affect any nation's economy, specifically in those where 70% of the inhabitants rely on the harvest from agriculture for their livelihood and subsistence. These problems can be solved by the identification and prevention of crop diseases. This research analyses the diseases based on images of leaves.[3]

Agricultural yields will be drastically reduced because of leaf diseases and it will affect the quality and quantity of agricultural products. Thus, artificial intelligence and deep learning techniques recognition of diseases on leaves play a crucial role in the agriculture sector. This plant and crop disease detection research is based on the same idea which can be used to detect disease in specific crops on which it is trained.[4]

This study aims to identify crop diseases by analyzing leaf images through deep learning techniques. By mimicking the problem-solving approach of the human brain, deep learning utilizes neural networks to address complex challenges. The research leverages Convolutional Neural Networks (CNN), a powerful tool specifically designed for processing image data. CNNs excel in detecting intricate features quickly and efficiently, making them well-suited for classifying diseased leaves, even when the images contain a vast number of detailed attributes. [5]

The input data used in this research comprises a disease image representing different plants and its associated diseases. The dataset contains the four main agricultural food products: corn, potato, rice, and wheat. Each food product plant leaf consists of specific disease classes and a

[1] .SRM Valliammai Engineering College , Tamilnadu –603203,INDIA
ORCID ID : 0000-0001-8210-322X
[2] SRM Valliammai Engineering College , Tamilnadu –603203,INDIA
ORCID ID : 0009-0004-6303-3704
[3] SRM Valliammai Engineering College , Tamilnadu –603203,INDIA
ORCID ID : 0000-3343-7165-777X
[4] SRM Valliammai Engineering College , Tamilnadu –603203,INDIA
ORCID ID : 0000-0002-6630-2074
* Corresponding Author Email: narayanans.it@srmvalliammai.ac.in

healthy class for comparison. The total number of disease classes in the dataset is 14, with a combined total of 13,024 images.

Corn Plant: The corn plant category contains images depicting diseases such as Common Rust, Gray Leaf Spots, and Northern Leaf Blight, along with a class representing healthy corn plants. The dataset consists of 3,852 total images, with Common Rust has 1,192 images, Gray Leaf Spot having 513 images, Northern Leaf Blight has 985 images, and the Healthy class has 1,162 images.

Potato Plant: The potato plant category contains images related to Early Blight, Late Blight, and healthy potato plants. A data set consisting of 2,152 total images are included in this category, with 1,000 images each for Early Blight and Late Blight, and 152 images representing the Healthy class.

Rice Plants: The rice category plants include leaves images representing diseases like Brown Spot, Leaf Blast, and Neck Blast, along with a Healthy class. The dataset consists of 4,078 images, with 613 images for Brown Spot, 977 images for Leaf Blast, 1,000 for Neck Blast, and 1,488 for the Healthy class.

Wheat Plant: The wheat plant category consists of images related to diseases such as Brown Rust, Yellow Rust, and healthy wheat plants. It contains 2,942 images, with 902 images for Brown Rust, 924 images for Yellow Rust, and 1,116 images representing the Healthy class.

Deep learning network in crop disease prediction:

In artificial intelligence machine learning's most remarkable advancement is deep learning. Deep learning is processed by observing and making decisions based on learned knowledge from previously executed data. Deep learning algorithms are used in computer vision, face detection, audio recognition and processing, and various other applications [6].

Deep learning uses open-source algorithms, image segmentation, and clustering to detect tomato plant leaf disease and create a trustworthy, secure, and accurate system for identifying leaf disease, emphasizing tomato plants. Durmuş in their research identified [7] the diseases that have been recognized to harm greenhouse or field-grown tomatoes. Deep learning identified different diseases in tomato plant leaves. The analysis aimed to use the deep learning algorithms run in real-time on the robot. As a result, the robot can identify plant diseases either manually or automatically. Two different deep-learning network topologies, AlexNet and SqueezeNet, were used. These deep-learning networks were trained and validated on the Nvidia Jetson TX1. Tomato leaves photos from the PlantVillage database were used for the training. Ten separate classes were used, all of which have healthy imagery [8]. This research created a unique approach to disease detection in tomato plants. tomato plant were photographed using a motor-controlled picture-taking box in a four sides to detect and diagnose leaf diseases. The tomato variety Diamante Max was tested. Phoma Blight, Leaf Miner, and Targeted Spot diseases were detected by this approach. The convolutional neural network was used to assess whether tomato infections were present in the plants.

2. Related work

Mohit Agarwal et al. proposed a deep learning-based approach for detecting and classifying diseases in tomato leaves using a Convolutional Neural Network (CNN). Highlighting the significance of tomatoes as a globally cultivated crop vital to cuisine, their research emphasizes the critical need for effective disease management. The CNN model developed in the study comprises three convolutional layers, three max-pooling layers, and three fully connected layers. Experimental results demonstrated that this model outperformed other deep learning models, including VGG16, InceptionV3, and MobileNet, achieving classification accuracy between 76% and 100%, depending on the disease class.[9]

The paper "Leaf disease detection using machine learning and deep learning: Review and challenges" by Chittabarni Sarkar et al provides a detailed insight into advances in detecting leaf diseases using machine learning (ML) and deep learning (DL) techniques. The research shows the importance of early detection of plant diseases. The data taken for research from 2010 to 2022, highlights the use of multispectral and hyperspectral imaging technologies for disease detection. It provides information about ML models and deep learning models. The Support Vector Machines (SVM), Random Forest, and Multiple Twin SVM (MTSVM) ML models and Convolutional Neural Networks (CNN), Visual Geometry Group (VGG), and ResNet (RNet) models are used in this research for identifying leaf diseases. The paper evaluates F1 score, precision, and accuracy, and introduces a workflow mechanism.[10]

The article titled "Classification of Sugarcane Leaf Disease using Deep Learning Algorithms" by Hernandez et al., investigates diseases in sugarcane leaves through image analysis using deep learning models. The analyzed dataset consists of 16,800 training images, 4,800 validation images, and 2,400 testing images. The experimental results on various deep learning models tested—InceptionV4, VGG16, ResNetV2-152, and AlexNet—InceptionV4 successfully achieved the highest accuracy at 99.61%. ResNetV2-152 and AlexNet also provide 99.23% and 99.24% accuracy and VGG16 achieved a slightly lower accuracy of 98.88%. [11]

The article titled "Sugarcane Disease Detection Using CNN-Deep Learning Method: An Indian Perspective" by Sammed Abhinandan Upadhye et.al addresses the effective detection of diseases in sugarcane crops. These diseases can reduce both the quality and quantity of sugarcane products, early and accurate detection is crucial to eradicate financial losses and prevent excessive crop damage. This research highlights the advantages of Artificial Intelligence (AI), specifically, Convolutional Neural Net works (CNN), which achieved an impressive detection accuracy of 98.69% to improve plant disease detection. By employing a CNN model to classify sugarcane images into healthy or diseased categories, with a further breakdown of disease types, the researchers. Additionally, the study introduces a web-based application that allows farmers to access real-time disease detection and monitoring, enhancing their ability to manage crop health effectively.[12]

The research titled "DeepCrop: Deep learning-based crop disease prediction with web application" was implemented to improve plant disease detection using advanced deep learning techniques. By using various models—like CNN, VGG-16, VGG-19, and ResNet-50—the researchers analyzed a dataset of 10,000 plant

images. In the ResNet-50 model got the highest accuracy rate of 98.98%, making it the most important for disease identification. Additionally, the ResNet-50 model was integrated into a web application designed to assist farmers in plant diseases detection from leaf photos. This research application aims to facilitate early disease detection, allowing farmers to implement timely interventions, conserve resources, and reduce economic losses through improved management practices.[13]

3. Proposed methodology

The different phases for detecting diseases in plants are presented in Fig. 1. The below section describes each of the steps of the proposal. The next section provides a breakdown of each step within this architectural framework.

3.1. Data Collection

The dataset analyzed consists of images representing various plant species and their associated diseases, categorized into four primary groups: corn, potato, rice, and wheat. Each group includes specific disease classes as well as a healthy class for comparison. The dataset comprises a total of 14 classes with 13,024 images in total. The distribution of images across plant types is as follows: corn with 3,852 images, potato with 2,152 images, rice with 4,078 images, and wheat with 2,942 images.

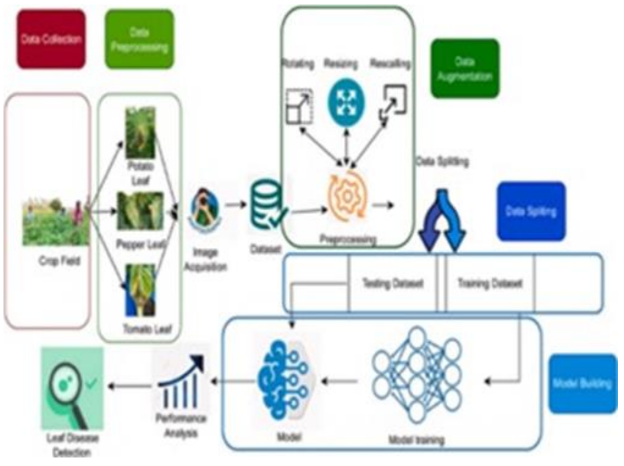

Fig-1 System Architecture

A detailed analysis of the dataset reveals variations in the frequency of different plant conditions. Among the categories, Rice Healthy has the highest representation with 1,488 instances, followed by Corn Common Rust with 1,192 occurrences. Other significant categories include Wheat Healthy with 1,116 records and Potato Early Blight with 1,000 instances. Both Corn Northern Leaf Blight and Rice Neck Blast also feature prominently with 1,000 images each. However, some categories are less represented, such as Potato Healthy with only 152 images and Corn Gray Leaf Spot with 513 images.

Table-1 Image Category

S. No	Category	Total Image
1	Corn Common Rust	1192
2	Corn Gray Leaf	513
3	Corn Healthy	1162
4	Corn Northern Leaf Blight	985
5	Potato Early Blight	1000
6	Potato Healthy	152
7	Potato Late Blight	1000
8	Rice Brown Spot	613
9	Rice Healthy	1488
10	Rice Leaf Blast	977
11	Rice Neck Blast	1000
12	Wheat Brown Rust	902
13	Wheat Healthy	1116
14	Wheat Yellow Rust	924

The variety in counts highlights the differing prevalence of these plant conditions, providing valuable insight into their relative impact. In a standard machine learning workflow, the dataset is typically divided into three sets: training, validation, and test. The training set, comprising 70% of the data, is used to train the model by optimizing its parameters based on the input features and their corresponding labels. The validation set, which makes up 20% of the data, is used to fine-tune hyperparameters and select the best model configuration while avoiding overfitting to the training data. The remaining 10% is allocated to the test set, which evaluates the model's performance on unseen data to measure its generalization ability.During training, data is processed in batches; for example, a batch size of 32 indicates that the model updates its weights after processing 32 images at a time.

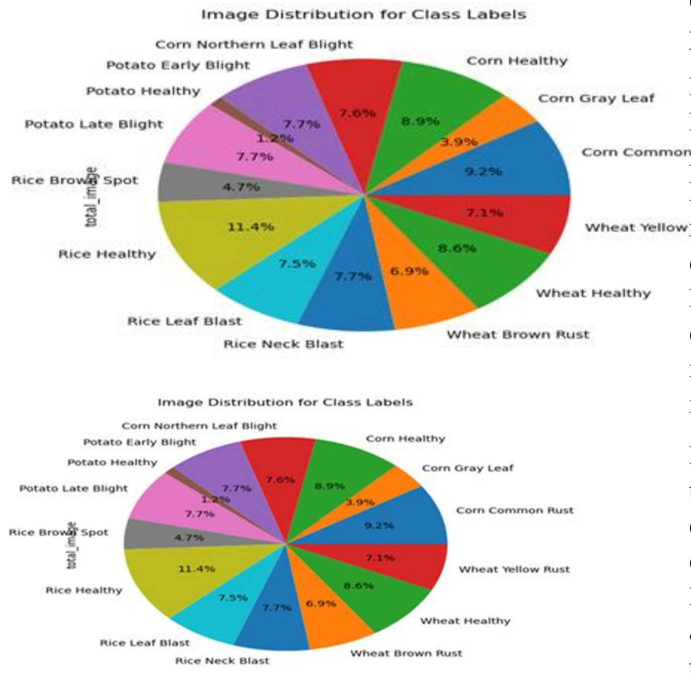

Fig 2. Differing Prevalence of Plant Conditions

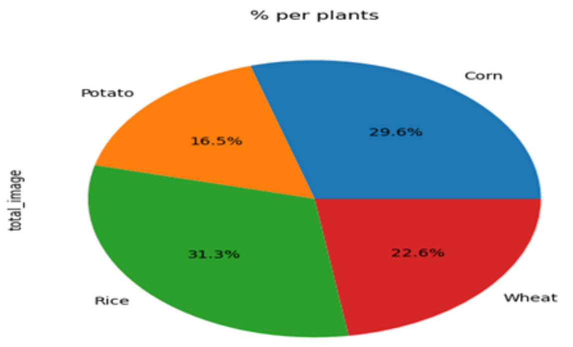

Fig 3. Percentage per Plant

This batch-wise processing ensures efficient memory usage and impacts the stability and speed of the learning process.

3.2. Image Preprocessing

Image preprocessing and enhancement techniques are essential for preparing datasets for deep learning tasks. Consistently resizing images to uniform dimensions, such as 299x299 pixels, ensures a standardized input shape across the dataset, simplifying batch processing and model architecture design. Normalizing pixel values to a range between 0 and 1 or -1 and 1 standardizes the data, enabling faster convergence by maintaining manageable input ranges and reducing computational demands.

Data augmentation introduces variability to the dataset without requiring additional data. Techniques such as color jitter, random flips (horizontal/vertical), random scaling and cropping, rotations, and zooms enhance the model's ability to generalize. By simulating diverse real-world conditions, these techniques improve adaptability and performance on unseen data.

Pre-trained CNN models like InceptionV3, VGG19, and EfficientNet are powerful tools for capturing intricate image patterns without training from scratch. By leveraging their deep feature extraction capabilities, researchers can achieve improved efficiency and reduced computational costs, making them ideal for large datasets. Each model has unique strengths, making it valuable to explore whether one consistently outperforms the others or if certain models excel with specific image characteristics in research applications.

Model training begins with designing a neural network tailored to the dataset and task requirements. This involves defining the architecture, including layers and configurations, to address the problem effectively. Hyperparameters like learning rate, batch size, and epochs are fine-tuned to optimize the training process. Once trained, the model is ready for image classification tasks, categorizing images into predefined classes by comparing extracted features against known patterns.

Convolutional Neural Networks (CNNs) are integral to this process, excelling at handling image data by extracting features such as texture and density through convolutional and pooling layers. In crop disease detection, CNNs analyze images of plants to identify and classify diseases, enabling early diagnosis and effective management strategies.

3.3. Data description

Adopting a structured naming convention, such as "CropType_Disease," is an effective way to organize and streamline data for analysis. This approach allows researchers and analysts to quickly identify folder contents and distinguish between various crop types and disease states. Including "Healthy" folders further supports the setup of control groups, which are essential for comparative studies and understanding the impact of diseases.

Such a standardized organization also simplifies automated data processing, enabling software to categorize and process data efficiently based on folder names. Moreover, it enhances reproducibility and facilitates collaboration, as others working with the dataset can easily understand its structure and purpose. This method demonstrates how thoughtful data organization can significantly improve the efficiency and accuracy of data analysis workflows.

3.4. Renaming

The code iterates through a list of folder names using enumerate, which provides both the index and folder name for each iteration. For each folder, it builds the full path to the current folder and the new target path, where the new folder name is derived from the index. Then, it uses os rename to rename each folder according to this new name. After each renaming, it prints a message displaying the old and new folder names, providing immediate feedback on the changes made.

3.5. Error Handling

To enhance the script's robustness, error handling has been implemented to manage potential issues such as FileNotFoundError, PermissionError, and other unexpected exceptions. This approach allows the script to handle various scenarios gracefully, providing clear error messages when issues occur. The os.makedirs function is used to create directories for organizing data into training (train_dir), testing (test_dir), and validation (val_dir) sets. By setting the exist_ok=True argument, the script avoids errors if the directories already exist, making it more robust and idempotent. This setup helps ensure a smooth directory creation process, which is crucial for efficient data management. Once all folders are processed, a completion message confirms that the renaming operations are finished. This approach is widely used in data processing and machine learning workflows to systematically organize and prepare data for model training and evaluation.

3.6. Data Splitting

Preparing a dataset for machine learning involves dividing it into training, testing, and validation sets, typically in the ratios of 70%, 20%, and 10%, respectively. This process begins with random shuffling to ensure each subset accurately represents the entire dataset. The training set, comprising 70% of the data, is used to train the model, enabling it to learn patterns from the data. The remaining 30% is split into a testing set (20%) and a validation set (10%). The testing set evaluates the model's performance on unseen data, while the validation set is used to fine-tune hyperparameters and optimize the model before final evaluation. This structured methodology ensures the model generalizes well, delivering reliable performance across diverse data subsets.

4. Exploratory data analysis (EDA)

Building and training a deep learning model involves several key stages. The process begins with training the model on a dataset of labeled images. During this phase, the data undergoes preprocessing, which typically includes resizing images to a uniform size and normalizing pixel values to ensure consistency. Convolutional layers are employed to extract features from the images, capturing details ranging from simple edges to intricate patterns. These features are then flattened into a one-dimensional array and fed into fully connected layers, which combine them to generate predictions. The model's weights are adjusted using a loss function, and this optimization process is repeated over multiple epochs.

Once training is complete, the model is validated using a separate test set of images that were not part of the training data. This step evaluates the model's ability to generalize to new data, using metrics such as accuracy, precision, and recall. Based on these evaluations, hyperparameters may be fine-tuned to further improve performance.

Finally, the trained and validated model is saved for deployment, enabling it to make predictions on unseen data. This systematic approach ensures that the model not only learns effectively from the training data but also performs reliably in real-world applications

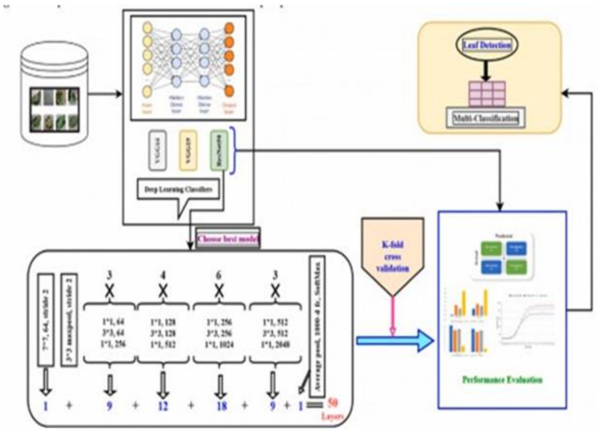

Fig. 4. Proposed model: three pre-trained deep learning models are used from then ResNet50 is finalized for selection; the layer-wise architecture details of ResNet50 are included.

5. Evaluate The Model

5.1 Xception model

This setup outlines a detailed process for building, training, and evaluating a neural network using TensorFlow and Keras, with the Xception model serving as the foundation for transfer learningtion model is initialized with `include_top=False`, excluding the final classification layers to enable feature extraction or customization for a specific task. The model is designed to accept 299x299 RGB images (`input_shape=(299, 299, 3)`) and leverages pre-trained ImageNet weights to provide a robust base of learned features. A batch size of 32 is used, updating weights after every batch of 32 samples.

During training, images are augmented with transformations such as rescaling, shifts, zooms, and flips to improve generalization, while validation and test datasets are only rescaled to ensure unbiased performance evaluation. The dataset is divided into 9,112 images for training, 2,600 for validation, and 1,312 for testing, ensuring a balanced allocation for training, tuning, and evaluation.

Using the Functional API, a custom model is built by passing inputs through the base Xception model (`model1`), flattening the outputs, adding dropout layers, and including dense layers with ReLU and softmax activations for final classification. Training involves 30,000 iterations, calculated based on the total samples, batch size, and steps per epoch. Monitoring mechanisms include `steps_per_epoch` and `validation_steps` to track performance. Key callbacks like `ModelCheckpoint` save the model when validation accuracy improves, while `EarlyStopping` halts training if no further improvements are observed, reverting to the best-performing model.

After training, the saved model is loaded, predictions are generated on the test data, and class labels are derived from the highest probabilities. This systematic approach, from initializing the model to final evaluation, establishes an efficient training pipeline that maximizes both performance and generalization.

5.2. Model performance

To evaluate the model's performance, the `classification_report` function from scikit-learn is used, providing detailed metrics such as precision, recall, F1-score, and support for each class. These metrics allow a comprehensive assessment by comparing the predicted labels with the true labels from the test dataset. It is crucial to ensure that class labels in the classification report align with those in the test dataset and that predictions and true labels are correctly matched for accurate evaluation.

The classification report includes overall performance metrics like accuracy, precision (P), recall (R), and the weighted F1-score, which are used to compare performance across different classes. The misclassification rate is also analyzed to highlight the model's ability to differentiate between classes effectively. Among the metrics, the weighted F1-score is particularly emphasized to evaluate the model's balance between precision and recall.

The results show that the model performs exceptionally well for most classes, with notable variations across labels. For example, classes 0, 10, 12, and 6 achieve near-perfect or perfect precision, recall, and F1-scores, reflecting the model's high accuracy in predicting these classes. Class 1 demonstrates strong performance with a balanced F1-score of 0.89, despite slightly lower precision (0.84) compared to recall (0.96), indicating effective identification of class 1 with minimal false positives.

However, class 7 exhibits a lower F1-score of 0.79, mainly due to a significant drop in recall (0.70). This suggests the model struggles to capture all true instances of class 7, despite being relatively accurate when it does make predictions. Overall, the model achieves an accuracy of 95%, with macro and weighted averages for precision, recall, and F1-score consistently around 0.95, highlighting its reliability and robustness in handling diverse classification tasks.It sets up a figure with a size of 12 by 6 inches and divides it into two subplots. The first subplot displays the training and validation loss over epochs, while the second subplot shows the training and validation accuracy. Legends are added to each plot to differentiate between the metrics, and titles are given to indicate the focus of each subplot.

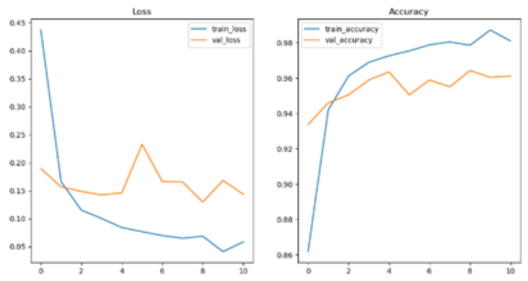

Fig 5. Training and Validation Accuracy

Finally, plt.show() is called to render and display the plots, providing a clear overview of how the model's performance evolved during training.

First, it calculates the confusion matrix using confusion_matrix from the sklearn.metrics module, comparing the true labels (generator_test.classes) with the predicted labels (y_pred). It then creates a heatmap of this matrix using seaborn, with annotations indicating the count of predictions for each class. The heatmap is displayed with matplotlib, where xticklabels and yticklabels are set to class_labels, representing the names of the classes in your dataset. The figure is sized at 10 by 8 inches, and the heatmap is color-mapped using a "Blues" gradient. Labels and a title are added to the plot to indicate the axes and the purpose of the visualization. This graphical representation helps in understanding how well the model performs across different classes by showing the distribution of correct and incorrect predictions.

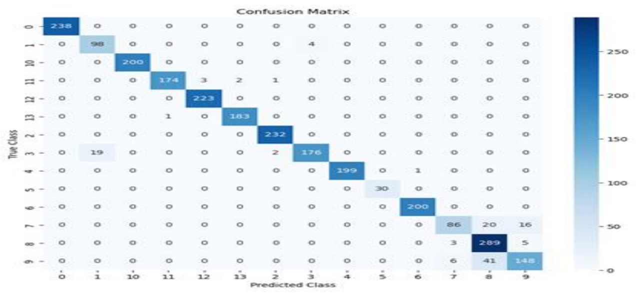

Fig 6. Distribution of Correct and Incorrect Predictions

This research is designed to identify and display misclassified images from a classification task. It starts by iterating through the predicted labels (y_pred) and compares them with the true labels obtained from generator_test.classes. If a discrepancy is found, indicating a misclassification, the image path, true label, and predicted label are collected and stored in a list called misclassified_images_X.

To visualize these misclassified images, select a subset, defined by num_display, of these misclassified entries. For each of these images, it loads the image from the file path and resizes it to the dimensions (299, 299) to match the expected input size. It then uses Matplotlib to display the image, setting the title to show the true label and the predicted label. This provides a clear visual representation of how the model's predictions diverge from the actual classes. Additionally, the paths and image objects of these displayed images are saved into image_path_show_X and image_show_X, respectively, for potential further use.

To ensure robustness, initialize image_path_show_X and image_show_X before the loop. Handling errors during image loading is also advisable to avoid interruptions if some images fail to load. Additionally, confirm that class_labels is correctly defined, mapping indices to class names, and ensure compatibility between y_pred and generator_test.classes in terms of format.

This approach helps in evaluating the performance of the classification model by providing insights into where and why the predictions are incorrect.

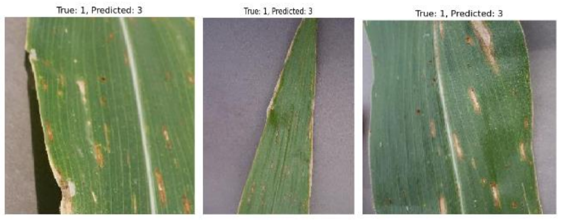

Fig 7. Misclassified Images

This research is designed to identify and display misclassified images from a classification task. It starts by iterating through the predicted labels (y_pred) and compares them with the true labels obtained from generator_test.classes. If a discrepancy is found, indicating a misclassification, the image path, true label, and predicted label are collected and stored in a list called misclassified_images_X.

To visualize these misclassified images, select a subset, defined by num_display, of these misclassified entries. For each of these images, it loads the image from the file path and resizes it to the dimensions (299, 299) to match the expected input size. It then uses Matplotlib to display the image, setting the title to show the true label and the predicted label. This provides a clear visual representation of how the model's predictions diverge from the actual classes. Additionally, the paths and image objects of these displayed images are saved into image_path_show_X and image_show_X, respectively, for potential further use.

To ensure robustness, initialize image_path_show_X and image_show_X before the loop. Handling errors during image loading is also advisable to avoid interruptions if some images fail to load. Additionally, confirm that class_labels is correctly defined, mapping indices to class names, and ensure compatibility between y_pred and generator_test.classes in terms of format.

This approach helps in evaluating the performance of the classification model by providing insights into where and why the predictions are incorrect.

5.3. Correctly Classify Images

To visualize and save correctly classified images, first ensure you have the necessary imports and that all required variables are properly initialized. Then, define image dimensions and initialize lists to store the paths and images of correctly classified examples. It then iterates through predictions, comparing each predicted label to the true label. If the prediction is correct, the image path, true label, and predicted label are appended to the classified_images_X list.

Subsequently, specifies the number of images to display and iterates through the list of correctly classified images. For each image, it loads and resizes it to the specified dimensions, then displays the image using Matplotlib with a title showing both the true and predicted labels. Finally, it appends the image path and the image itself to the lists class_image_path_show_X and class_image_show_X, respectively, for potential later use.

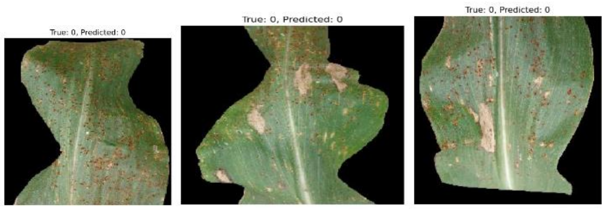

Fig 8. Correctly Classified images

5.4. InceptionV3

The InceptionV3 model from TensorFlow/Keras has been configured with a custom input size of 228x228 pixels instead of its default input size of 299x299 pixels. While InceptionV3 supports various input sizes, altering the default dimensions may impact model performance and accuracy. The pre-trained weights are optimized for the default size, so fine-tuning or retraining may be necessary to achieve optimal results with a custom size. If the default configuration is preferred, the model should be initialized with `input_shape=(299, 299, 3)`. For custom sizes, it is essential to ensure data compatibility and be prepared to make adjustments to the training process if needed.

The model architecture includes a sequential model (`new_model1`) built using the pre-trained InceptionV3 model (`model2`) as the base. Additional layers, such as dropout and dense layers, are added to enhance performance. The model is compiled with a specific optimizer, loss function, and evaluation metrics. Training involves calculating the number of epochs based on the desired number of iterations and batch size.

Custom callbacks are utilized during training for effective monitoring and optimization: DesiredAccuracyCallback stops training when the model reaches a specified accuracy. ModelCheckpoint saves the model with the highest validation accuracy.EarlyStopping halts training when validation loss stops improving. The training process uses the `fit` method instead of the deprecated `fit_generator`, managing training and validation data with generators to ensure efficient learning.

The classification report indicates an exceptional overall accuracy of 97%. Most classes are classified with high precision and recall, achieving nearly perfect metrics for several, such as classes 0, 2, 4, 6, 10, and 12. These results highlight the model's strong capability to accurately identify and classify samples from these classes. However, some classes exhibit slightly lower performance. For example: Class 7 has the lowest recall, indicating the model is missing some true instances of this class.

Class 8 shows high recall but lower precision, suggesting more false positives. Classes 1 and 9 also show relatively lower precision and recall, which could benefit from targeted improvements.

To address these performance gaps, analyzing misclassified samples could help uncover why certain classes are more challenging. Data augmentation and fine-tuning can further improve results. Additionally, ensuring that training data is balanced and representative of all classes can enhance the model's performance across the board. Overall, the model is well-designed and shows strong performance, with room for refinement to achieve even better results.

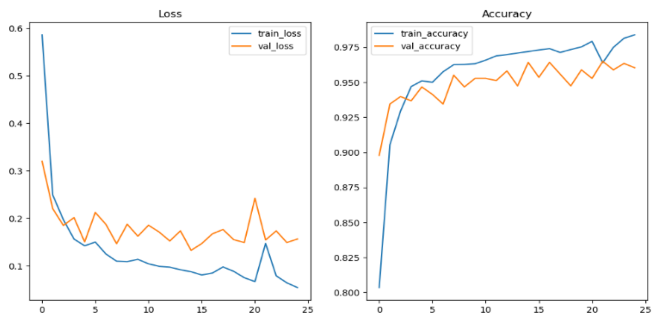

Fig 9. Performance Analysis

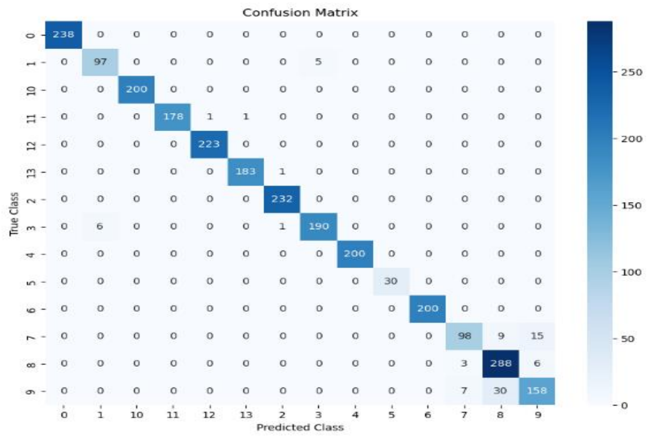

Fig 10. Confusion Matrix

5.5. VGG19 model

It used a data pipeline for training a model using the VGG19 architecture with TensorFlow/Keras. It correctly loaded the VGG19 model without its top layer and specified the input shape as (228, 228, 3), which is appropriate for the use case. However, when configuring the image data generators,it should use the target_size parameter as (228, 228) (height and width), instead of input_shape ImageDataGenerator instances:

Define the ImageDataGenerator for training with data augmentation such as width and height shifts, zoom, and horizontal flips. For testing and validation, only rescaling is applied to normalize the pixel values.

Create flow_from_directory instances for training, testing, and validation datasets, ensuring that target_size is specified as (228, 228), which matches the dimensions used by your VGG19 model. The batch_size and shuffle parameters should also be correctly set according to your needs.

The setting up of a new neural network model by utilizing a pre-trained model (model 3). It first creates a new Model instance (conv_model2) that has the same input and output layers as model3. This setup allows to use the pre-trained features from model 3. Then, define a new Sequential model (new_model2) and add conv_model2 as its first layer. After this, it appends a Flatten layer to convert the multi-dimensional output from conv_model2 into a one-dimensional array suitable for dense layers. To prevent overfitting, it add a dropout layer with a dropout rate of 0.5. Following that, you include a Dense layer with 512 units and ReLU activation to learn complex patterns, and another Dense layer with 14 units and a softmax activation function for classification into 14 classes. It then compiles the model with the specified optimizer, loss function, and evaluation metrics.

For training, it set num_iters to 30,000, which represents the total number of iterations it intends to train the model. It calculates the number of batches per epoch from your training generator, ensuring that the batch size is appropriate to avoid division by zero. Based on this, it determines the number of epochs required. Additionally, it prints out the number of steps per epoch and validation steps to monitor the training process. Ensure it's batch size and data generators are correctly defined and that the parameters align with it's dataset and computational resources.

It has implemented a custom callback called DesiredAccuracyCallback which is designed to stop training when a specified training accuracy is reached. This callback overrides the on_epoch_end method to check if the current training accuracy meets or exceeds the desired accuracy. If it does, the training is halted by setting self.model.stop_training to True. This approach is effective for early stopping based on a specific performance threshold, but it's important to handle the logs dictionary carefully to avoid issues if the accuracy key is missing. In your implementation, it's a good practice to use logs.get('accuracy', 0) to ensure the code doesn't break if accuracy is not present in the logs.

Additionally, the fit_generator method used for training has been deprecated in favor of the fit method, which supports generators directly. This change aligns with the latest updates in TensorFlow/Keras and ensures that the code remains compatible with current versions. It should replace fit_generator with fit for a more up-to-date and streamlined approach.

Make sure the metric names used in the callbacks, such as accuracy and val_accuracy, match those used in your model's compilation and evaluation. This ensures that the callbacks monitor the correct metrics. Finally, confirm that the desired accuracy value (desired_train_accuracy) is appropriately set to your target performance level. Adjust your callbacks and training method as necessary to fit your specific use case and TensorFlow/Keras version.

The performance of your VGG19 model on the test dataset is impressive, with an overall accuracy of 94%. The classification report reveals that the model achieves high precision, recall, and F1-scores for most classes. For instance, it excels in classes like 0, 2, 4, 5, 6, and 12, demonstrating near-perfect performance. However, there are a few areas where the model could improve. Class 1, for example, has a lower precision, indicating that while it successfully identifies most relevant instances, it also includes some false positives. Similarly, Class 7 shows a lower recall, suggesting that the model sometimes misses instances belonging to this class. To address these issues, you might consider investigating potential class imbalances or augmenting the dataset for underperforming classes. The model is robust, but focusing on these areas could further enhance its performance.

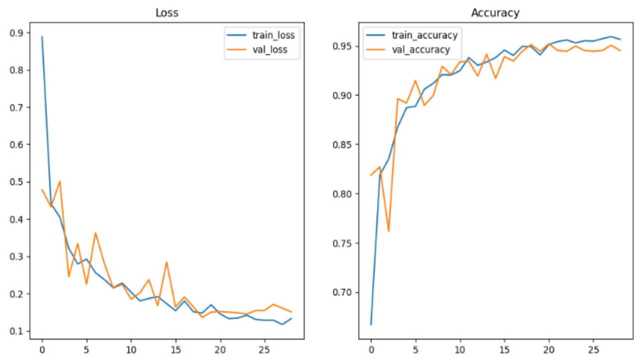

Fig 11. Performance Evaluation

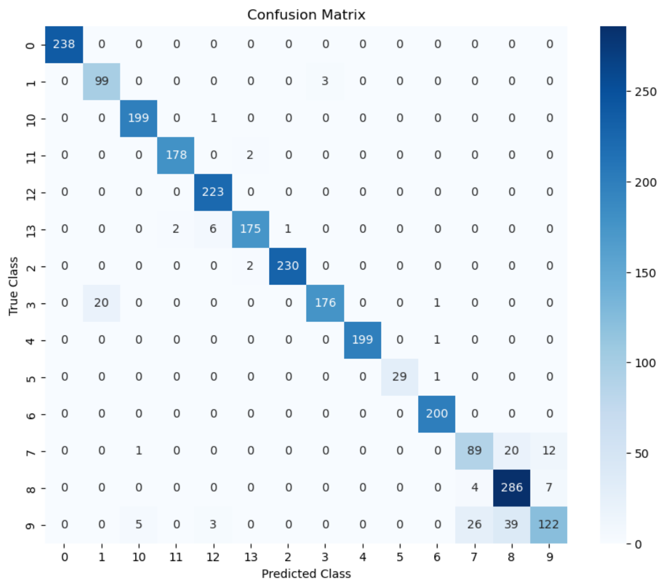

Fig 12. Confusion Matrix

6. Conclusion and Future Improvements

The Xception model achieved strong overall accuracy but struggled with class 7. This could be improved by enhancing recall for this class. The InceptionV3 model has high accuracy overall, but performance for class 7 could be enhanced. Addressing precision-recall trade-offs for class 8 and others might improve the model further. The VGG19 model produced a solid performance with an overall accuracy of 94%, but precision and recall issues for classes 1 and 7 suggest additional tuning or data augmentation for these classes. To improve the performance of these models, various strategies were used to fine-tune the last few layers to tailor the models to the specific dataset. Data augmentation to generate more diverse training samples. Class balancing techniques like class weights or oversampling underrepresented classes. Error analysis (e.g., investigating misclassified images) to identify patterns and further optimize the models.

References

[1] David M. Rizzo, Maureen Lichtveld, Jonna A. K. Mazet, Eri Togami & Sally A. Miller "Plant health and its effects on food safety and security in a One Health framework: four case studies" *One Health Outlook* volume 3, Article number: 6 (2021)

[2] GEORGE N. AGRIOS, " Plant Pathology " (Fifth Edition), 2005.

[3] P. A. Nazarov, D. N. Baleev, M. I. Ivanova, L. M. Sokolova, and M. V. Karakozova" Infectious Plant Diseases: Etiology, Current Status, Problems and Prospects in Plant Protection" Acta Naturae. 2020 Jul-Sep; 12(3): 46–59.

[4] Aakanksha Rastogi, Ritika Arora, Shanu Sharma" Leaf disease detection and grading using computer vision technology & fuzzy logic" Conference: 2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)-IEEE.

[5] Andrew J.,Jennifer Eunice,Daniela Elena Popescu,M. Kalpana Chowdary and Jude Hemanth" Deep Learning-Based Leaf Disease Detection in Crops Using Images for Agricultural Applications" *Agronomy* 2022, *12*(10), 2395.

[6] Ashok, G. Kishore, V. Rajesh, S. Suchitra, S.G. Soph ia, B. Pavithra"Tomato leaf disease detection using deep learning techniques"2020 5th International Conference on Communication and Electronics Systems (ICCES), IEEE (2020), pp. 979-983.

[7] H. Durmuş, E.O. Güneş, M. Kırcı" Disease detection on the leaves of the tomato plants by using deep learning"2017 6th International Conference on Agro-Geoinformatics, IEEE (2017), pp. 1-5.

[8] R.G. De Luna, E.P. Dadios, A.A. Bandala" Automated image capturing system for deep learning based tomato plant leaf disease detection and recognition" TENCON 2018-2018 IEEE Region 10 Conference, IEEE (2018), pp. 1414-1419.

[9] Mohit Agarwal , Abhishek Singh , Siddhartha Arjaria , Amit Sinha , Suneet Gupta " ToLeD: Tomato Leaf Disease Detection using Convolution Neural Network" Procedia Computer Science ,Volume 167, 2020, Pages 293-301.

[10] Chittabarni Sarkar , Deepak Gupta , Umesh Gupta , B arenya Bikash Hazarika "Leaf disease detection using machine learning and deep learning: Review and challenges" Applied Soft Computing,Volume 145, September 2023, 110534.

[11] Alexander A. Hernandez; Joferson L. Bombasi; Ace C. Lagman" Classification of Sugarcane Leaf Disease using Deep Learning Algorithms" 2022 IEEE 13th Control and System Graduate Research Colloquium (ICSGRC) : IEEE

[12] Sammed Abhinandan Upadhye, Maneetkumar Rangnath Dhanvijay , Sudhir Madhav Patil " Sugarcane Disease Detection Using CNN-Deep Learning Method: An Indian Perspective" Universal Journal of Agricultural Research 11(1): 80-97, 2023.

[13] Md. Manowarul Islam , Md Abdul Ahad Adil , Md. Alamin Talukder , Md. Khabir Uddin Ahamed , Mdshraf Uddin , Md.Kamran Hasan , Selina Sharmin , Md.Mahbubur Rahman , Sumon Kumar Debnath" DeepCrop: Deep learning-based crop disease prediction with web application" Journal of Agriculture and Food Research Volume 14, December 2023, 10076

[14] Shallu Sharma, Sumit Kumar "The Xception model: A potential feature extractor in breast cancer histology images classification" ICT Express,Volume 8, Issue 1, March 2022, Pages 101-108.

Acknowledgments

This research was supported/partially supported by SRM Valliammai Engineering college. We thank our colleagues from SRM Valliammai Engineering College who provided insight and expertise that greatly assisted the research, although they may not agree with all of the interpretations/conclusions of this paper.

Author contributions

Dr. S. Narayanan: Conceptualization, Methodology, Software, Field study.

S. Suresh Kumar: Data curation, Writing-Original draft preparation, Software, Validation., Field study
D. Nisha: Visualization, Investigation,

Dr. S Sekar: Writing-Reviewing and Editing.

Conflicts of interest

The authors declare no conflicts of interest.