Implementation of Plant Disease Classification Model Using Transfer Learning and DCGAN Data Augmentation in Android Based Application

(Case Study on Tomato Leaves)

Christopher Justine William
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
Email: 13519006@std.stei.itb.ac.id

***Abstract*—The early detection of plant diseases plays a crucial role in preventing their spread and minimizing damage. With advancements in computer vision technology, deep learning approaches have emerged as effective methods for disease detection in plants. However, to make these models accessible to a wider audience, it is important to develop models that can run smoothly on simpler devices like mobile gadgets. The objective of this paper is to implement a plant disease classification model using transfer learning and the DCGAN data augmentation technique in an Android based application. The study focuses on tomato leaf diseases using the PlantVillage dataset. The transfer learning approach is chosen to leverage pre-trained models and accelerate the training process by leveraging existing knowledge. Additionally, the DCGAN data augmentation technique is employed to address data limitations by generating additional data that enhances the balance and diversity of the dataset. Through testing and evaluation, the model achieved the highest accuracy of 97.83% when trained and tested on the PlantVillage dataset. This model, with the highest accuracy, is then integrated into an Android application for disease classification on tomato plant leaves.**

Keywords—plant disease classification, transfer learning, DCGAN augmentation, android application.

I. INTRODUCTION

Indonesia is a country with enormous natural resource potential in agriculture. This is supported by the agricultural industry which is the second largest supporting sector for the Indonesian economy which contributes 13.28% to national GDP in 2021 [1]. However, this great potential is also followed by threats that can harm the growth of cultivated plants which can result in losses, one of which is the threat of diseases in plants which are estimated to collectively cause around 16% loss of crop yields globally [2]. Early detection of disease in a plant needs to be done before the disease spreads and attacks other plants which can cause greater losses.

Utilizing deep learning in agriculture for identifying diseases in cultivated plants necessitates a model that can run on basic devices, enabling a wide range of users to employ it. However, models designed for simple devices, which have limited memory and computing capabilities, often have to compromise with accuracy. Developing a model that can be implemented on such devices poses a challenge. In a study by Ahmed et al. [3], the detection of diseases on tomato plant leaves was examined through the utilization of the transfer learning method, employing a pre-trained MobileNetV2 model. By leveraging the knowledge derived from pre-trained models, transfer learning improves the training process, leading to faster and more efficient training. Similarly, Ahmad et al. [4] conducted a study on disease detection in plants, employing the stepwise transfer learning method with pre-trained models such MobileNetV3. The results of their research indicate enhanced accuracy, precision, recall, and f1-score for models trained with the stepwise transfer learning approach. It is important to note that this research did not focus on a single plant type but encompassed multiple plants within one model training. Additionally, Wu et al. [5] conducted a study where they examined disease detection on tomato plant leaves by employing the DCGAN data augmentation technique.

This paper focuses on the implementation of deep learning models for identifying plant leaf diseases on mobile devices. It specifically presents a case study on tomato leaf disease using the PlantVillage dataset. Acquiring large plant disease datasets directly is challenging in reality. Hence, suitable data augmentation techniques are necessary to increase the initial dataset size for deep learning model training. The choice of transfer learning with a pre-trained model is favored over building a model from scratch due to its ability to expedite the training process by leveraging knowledge from the pre-trained model. Transfer learning enables reducing the dataset size while benefiting from expertise encoded in pre-trained models from relevant scientific domains. Additionally, the DCGAN data augmentation technique is employed to address the issue of limited and imbalanced datasets by generating additional data. Multiple basic model architectures are utilized during training, and the results are compared in terms of accuracy and model size.

II. Related Works

The current deep learning model faces several challenges, including the necessity for a large number of parameters, lengthy training time, and difficulties in implementing the model on resource-constrained devices. Ahmed et al. [3] addressed these challenges by developing a classification model for tomato plant leaf diseases using transfer learning method with pre-trained models such MobileNetV2, followed by a classifier network. By employing transfer learning, the model training process becomes faster and more efficient as it leverages knowledge derived from pre-trained models. Furthermore, the MobileNetV2 architecture is specifically designed to generate models that can operate effectively on devices with limited computing resources, making them suitable for deployment on simple devices such as mobile device. The model implemented using MobileNetV2 architecture achieved an accuracy of 97.27% with a total of 2.28 million parameters when evaluated on the tomato plant leaf dataset from PlantVillage. In this study, the transfer learning method involved activating all layers in the model. However, it is important to note that the model may lose its ability to recognize fundamental features of unseen data if it becomes overly focused on studying the training data, resulting in reduced accuracy for previously unseen data.

In the study conducted by Ahmad et al. [4], the model used for plant disease classification employed the stepwise transfer learning method with pre-trained models. Unlike conventional approach of training all layers simultaneously, the stepwise transfer learning model initially freezes all layers except the classification layer. This approach gradually introduces updates to the layers after they have reached a certain level of saturation over a specific number of epochs. In contrast, conventional transfer learning begins training by updating the weights of all layers, effectively overwriting the weights inherited from the previously trained model. However, in stepwise transfer learning, the previously trained weights are retained by initially freezing the layers, and they are only activated when the model's capacity to learn new features becomes insufficient. The implemented model, utilizing the MobileNetV3 architecture, achieved an impressive accuracy of 99% on the plant leaf dataset obtained from PlantVillage. It is important to note that this research did not exclusively focus on tomato plant leaves but encompassed various types of plants and other plant parts as well.

In the study conducted by Wu et al. [5], a novel approach for data augmentation was proposed in the model used to classify diseases on tomato plant leaves. Conventional data augmentation techniques such as rotation, flip, and translation often do not achieve satisfactory accuracy. To address this issue, the researchers utilized the Deep Convolutional Generative Adversarial Network (DCGAN) model. This model combines the original image with the augmented image generated by DCGAN as input for the identification model. The images generated by DCGAN exhibit improved consistency and diversity compared to those produced by conventional GANs. The accuracy results of the implemented model, utilizing GoogLeNet, achieved 94.33% when evaluated on the tomato plant dataset from PlantVillage.

III. Material and Method

The proposed solution in this paper is an application that runs on a device, allowing users to input images of tomato plant leaves and receive predictive results regarding the detection of diseases affecting those leaves. In order to generate these predictions, a lightweight model is developed, prioritizing computational and memory efficiency. The model will then be integrated into a mobile application to facilitate the identification of diseases on tomato plant leaves. There are three main components in this solution: the DCGAN model, the classification model, and the device application. The research foundation for this paper is based on the work of Wu et al. [5] research on DCGAN data augmentation and explores the potential of different pre-trained models and transfer learning methods. Additionally, it incorporates the transfer learning approach from Ahmed et al. [3] and Ahmad et al. [4].

A. Dataset

The initial dataset is processed through the data preparation phase, following a similar approach as in Wu et al. [5] study, to partition it into training, validation, and test data. The dataset employed in this paper consists of tomato plant leaf data obtained from PlantVillage. It is divided into ten classes representing different types of leaf diseases in tomato plants, resulting in a total of 300 instances. Each class contains 240 training samples and 60 test samples, randomly distributed. The DCGAN model is trained using the 240 training samples from each class. Subsequently, the DCGAN model expands the training dataset to include 1000 instances for each class, resulting in 800 training samples and 200 validation samples. These augmented samples will be utilized in the training process of the classification model. The distribution of data quantities per class after the data preparation process is presented in Table I.

TABLE I. Distribution of Classes In the Dataset After Processing

Class Label	Initial Data	DCGAN Data	Train Data	Test Data
Bacterial Spot	300	760	1000	60
Early Blight	300	760	1000	60
Healthy	300	760	1000	60
Late Blight	300	760	1000	60
Leaf Mold	300	760	1000	60
Septoria Leaf Spot	300	760	1000	60
Target Spot	300	760	1000	60
Tomato Mosaic Virus	300	760	1000	60
Two Spotted Spider Mites	300	760	1000	60
Yellow Leaf Curl Virus	300	760	1000	60
Total	3000	7600	1000	600

B. DCGAN Model

The DCGAN model is developed to expand the training dataset by generating artificial images, thereby increasing its size. It consists of two distinct models: the Generator and the Discriminator. The Generator is responsible for creating artificial images, which are then used to augment the training dataset. On the other hand, the Discriminator plays a crucial role in the overall training process of the DCGAN model by distinguishing between original and generated images. During the training process, the DCGAN model undergoes a set number of epochs. In each epoch, both the Generator and the Discriminator are trained iteratively on batches of the dataset. The Generator receives a noise input at each iteration and generates artificial images, while the Discriminator takes an input image and determines whether it is real or generated. The Discriminator's output value is used to calculate the losses for both the Generator and the Discriminator. The gradient of each model is then multiplied by the model parameters, allowing them to be trained and updated.

The architecture of the DCGAN model used in this paper is presented in Fig 1. Within the generator component, a dense layer employed to transform the input noise into a more complex representation with dimensions of 32x32x256. This is followed by multiple deconvolution layers that upscale the image to achieve the desired resolution. The inclusion of the dense layer with these dimensions enables it to capture complex patterns from the initial noise and generate improved output. To ensure compatibility during deconvolution operations, the input size of each deconvolution layer matches the output size of the preceding layer. On the other hand, the discriminator model incorporates a convolution layer responsible for extracting features from the image being evaluated.

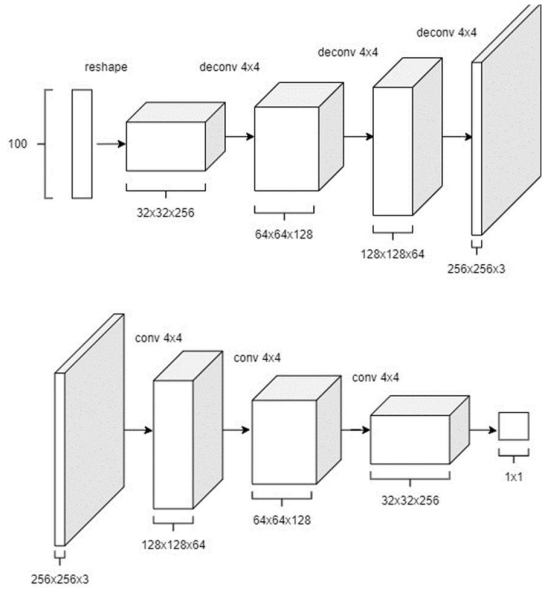

Fig. 1. Architecture of the Generator model (top) and Discriminator model (bottom).

C. Classification Model

The design solution in this paper incorporates architectural modification derived from the Ahmed et al. [3]. In this design, a pre-trained model is employed as a feature extractor, serving as input for additional classification layers. The additional classification layer comprises a combination of dense layers, batch normalization, and dropouts, as depicted in Fig 2. The dense layer, utilizing the ReLu activation function, calculates the weights between the nodes in the preceding layer and the nodes in the current layer. The batch normalization layer normalizes input values, aiming to expedite the training process and prevent overfitting of the training data. Meanwhile, the dropout layer aids in mitigating overfitting by randomly deactivating a small portion of neurons during each training iteration. Through experiments conducted using Bayesian Optimization on Keras Tuner to identify the model with the most optimal parameters, it was determined that the ideal number of nodes is 128 and 64, with a dropout rate of 0.2. The number of nodes indicates the quantity of units utilized in the dense layer, while the dropout rate represents the proportion of units randomly deactivated during the training process.

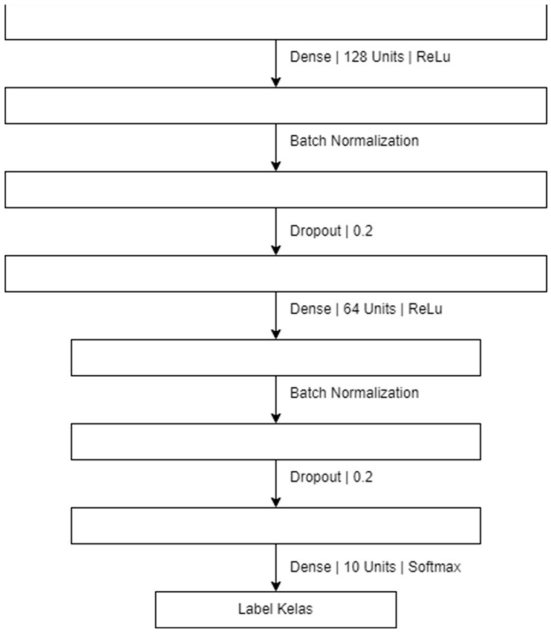

Fig. 2. Classification Layer Architecture.

Experiments were conducted on pre-trained models utilizing diverse base architectures, such as DenseNet121, MobileNetV2, ResNet50, and VGG19. Each architecture underwent training using the transfer learning technique on an initial dataset containing 300 samples for each class. The outcomes revealed that the DenseNet121 model achieved the highest accuracy rate of 91% compared to other base architectures. The VGG19 and MobileNetV2 models demonstrated similar accuracy rates of 84.67% and 85% respectively. On the other hand, the ResNet50 model exhibited the lowest accuracy level among the different base architectures, reaching 66.33%. The number of parameters served as the primary consideration when selecting the base

model for implementation in Android applications. This is because a lower number of parameters necessitates less computation and memory. Among the examined base architectures, MobileNetV2 possessed the fewest parameters and the smallest model size with around 2.4 million, making it the chosen architecture for implementation in Android application. Once the chosen base architecture is obtained, additional training is conducted using two scenarios during the classification model training process.

In the first scenario, all layers of the pre-trained model are initially unfrozen, allowing their weights to be updated throughout the training process. This enables the model to continuously learn and adapt its weights based on the training dataset. However, there are potential drawbacks to this approach. One is the risk of overfitting, where the model becomes too specialized to the training data and struggles to generalize to unseen data. Additionally, since all layers' weights are updated, the training time required may be longer.

In the second scenario, the weights of the pre-trained model's layers are frozen at the beginning of the training process, except for the classification layer. This means that the pre-trained model serves as an initial feature extractor, while the classification layer is updated to learn specific features relevant to the classification task. By freezing the pre-trained model's layers, the previously learned features at lower levels are retained, while the classification layer can adapt its weights to the new dataset. This approach is beneficial when the dataset is limited or when leveraging existing knowledge from a pre-trained model for a particular classification task. Consequently, the classification layer progressively learns the feature representation specific to the classification task. After several epochs, certain layers in the pre-trained model can be unfrozen to allow for updates and the learning of additional features at lower levels.

D. Android Application

During this phase, development of an Android based device application is undertaken, utilizing the previously constructed classification model. The resulting model is initially converted into a readable format, which is subsequently integrated into the application. The application allows users to input images either from the camera or the gallery. Subsequently, it displays the classification results based on the provided image. The application consists of various pages that serve different functionalities, including the homepage, plant selection page, camera page, gallery page, and classification results page. The specific functional requirements of the application can be found in Table II.

TABLE II. APPLICATION FUNCTIONAL REQUIREMENTS

ID	Description
FR-001	The application can select the type of plant.
FR-002	The application can capture images from the camera.
FR-003	The application can retrieve images from the gallery.
FR-004	The application can display the classification results of diseases on the selected plant from the camera.
FR-005	The application can display the classification results of diseases on the selected plant from the gallery.

IV. EXPERIMENT AND RESULT

A. Experimental Setup

The DCGAN model and classification model were implemented using the TensorFlow and Keras libraries in Python, while the mobile applications were implemented using the Flutter library in the Dart programming language. The implementation and testing environments for the DCGAN model and classification model are specified in Table III. Likewise, the implementation and testing environments for the device applications are outlined in Table IV.

TABLE III. ENVIRONMENT SPECIFICATION FOR IMPLEMENTATION AND TESTING OF DCGAN MODEL AND CLASSIFICATION MODEL

Hardware	**Specification**
CPU	Intel Xeon E5-2698 v3 (16 core, Haswell-EP)
GPU	Nvidia Tesla V100 32 GB RAM
Software	**Specification**
Operating System	Ubuntu 20.04.2 LTS
CUDA	CUDA Toolkit 11.8, cuDNN SDK 8.6.0
TensorFlow	TensorFlow 2.12.0

TABLE IV. ENVIRONMENT SPECIFICATION FOR IMPLEMENTATION AND TESTING OF ANDROID APPLICATION

Hardware	**Specification**
CPU	Qualcomm SDM730 Snapdragon 730 (8 core)
GPU	Adreno 618
Memory	8 GB RAM
Software	**Specification**
Operating System	Android 13
Flutter	Flutter 3.7.12

B. Evaluation Metrics

In this paper, the evaluation process includes testing both the model and the android application. Model testing is conducted to identify the optimal model for implementation in the android application. Similarly, android application testing is performed to ensure the proper functionality of the application, aligning with the specified test scenarios. The classification model was tested by evaluating its accuracy, precision, recall, and f1-score. Accuracy measures the model's ability to correctly classify tomato plant leaf diseases and serves as the primary metric for evaluating its performance. Precision indicates how well the model classifies positive samples while minimizing false positive errors. Recall measures the model's ability to correctly classify a class and minimize false negative errors. The F1-score provides an overall view of the balance between precision and recall. The model with the highest accuracy, considering precision, recall, and f1-score, was selected. Android application testing followed the functionality testing method, where test cases were executed on the application to achieve the goals specified in Table II. Functionality testing ensures that each test case can be successfully executed and evaluated.

C. Experiment on Scenario 1

The experiment was conducted on the model trained under scenario 1, which involved training the model with the pre-trained model layer unfrozen from the beginning of the training process. The evaluation results for the model trained with the original data set, without any augmentation, can be observed in Table V. Similarly, the evaluation results for the model trained with the original data set, augmented using DCGAN, can be found in Table VI. The following parameters were employed during the training of the classification model for scenario 1:

1) The batches were set to 16, 32, and 64.
2) The initial number of epochs consisted of 30 epochs, early stop if there was no validation loss improvement within 10 epochs.
3) Optimizer Adam with a learning rate of 0.0001, a multiplier factor of 0.1 applied if there was no improvement in validation accuracy within 5 epochs.

TABLE V. MODEL EXPERIMENT FOR SCENARIO 1 ON THE INITIAL DATA SET

Batch Size	Accuracy	Precision (Average)	Recall (Average)	F1-Score (Average)
16	0.9733	0.9737	0.9733	0.9733
32	0.9533	0.9541	0.9533	0.9529
64	0.9633	0.9638	0.9633	0.9632

TABLE VI. MODEL EXPERIMENT FOR SCENARIO 1 ON THE INITIAL DATA SET WITH DCGAN AUGMENTATION

Batch Size	DCGAN Batch	Accuracy	Precision (Average)	Recall (Average)	F1-Score (Average)
16	32	0.9717	0.9718	0.9717	0.9716
	64	0.9783	0.9786	0.9783	0.9781
32	32	0.9767	0.9771	0.9767	0.9766
	64	0.9750	0.9761	0.9750	0.9748
64	32	0.9667	0.9671	0.9667	0.9667
	64	0.9550	0.9550	0.9550	0.9549

The model trained on the initial dataset without augmentation achieved a high accuracy rate of 97.37% in scenario 1. With DCGAN augmentation, the model achieved a slightly higher accuracy of 97.83%. Augmentation increased the data set size and improved the model's adaptability to unseen variations. Both models demonstrated good performance with average precision, recall, and f1-score exceeding 90% for each class in the test data.

D. Experiment on Scenario 2

The experiment was conducted on a model trained for scenario 2, which involved freezing the pre-trained model layer at the start of the training process and gradually activating it layer by layer. The test outcomes for the model trained using the initial data set without augmentation are presented in Table VII. Additionally, the test results for the model trained with the DCGAN augmentation data set can be observed in Table VIII. The following parameters were employed during the training of the classification model for scenario 2:

1) The batches were set to 16, 32, and 64.
2) The initial number of epochs consisted of 30 epochs with all pre-trained model layers frozen, followed by 15 epochs with some of the pre-trained model layers activated, and finally 5 epochs with all pre-trained model layers activated.
3) Optimizer Adam with a learning rate of 0.0001, a multiplier factor of 0.1 applied if there was no improvement in validation accuracy within 5 epochs.

TABLE VII. MODEL EXPERIMENT FOR SCENARIO 2 ON THE INITIAL DATA

Batch Size	Accuracy	Precision (Average)	Recall (Average)	F1-Score (Average)
16	0.8917	0.8919	0.8917	0.8912
32	0.8750	0.8778	0.8750	0.8747
64	0.8800	0.8822	0.8800	0.8793

TABLE VIII. MODEL EXPERIMENT FOR SCENARIO 2 ON THE INITIAL DATA SET WITH DCGAN AUGMENTATION

Batch Size	DCGAN Batch	Accuracy	Precision (Average)	Recall (Average)	F1-Score (Average)
16	32	0.8967	0.8967	0.8967	0.8961
	64	0.9233	0.9350	0.9233	0.9241
32	32	0.9367	0.9381	0.9367	0.9362
	64	0.9033	0.9046	0.9033	0.9034
64	32	0.9533	0.9575	0.9533	0.9528
	64	0.8933	0.8933	0.8933	0.8929

The model trained on the initial dataset without augmentation achieved 89.17% accuracy in scenario 2, while the model trained with DCGAN augmentation reached 95.33% accuracy. Adding DCGAN augmentation significantly improved the model's accuracy. Average precision, recall, and f1-score exceeded 80% for each class in the test data, indicating strong predictive performance.

E. Experiment on Android Application

Based on the experiment outcomes, the implemented Android application successfully fulfills several tested functionalities. The features including plant type selection, camera capture, and gallery image capture work as intended. However, there are limitations in the application's ability to accurately classify diseases on tomato plant leaves. It performs well on test data from the PlantVillage dataset, but struggles with out-of-distribution (OOD) data. OOD refers to situations where a model trained on a specific dataset fails to generalize to data from a different distribution not encountered during training. When faced with significantly different characteristics and unseen data, the transfer learning-trained model may produce inaccurate classification results. Further research is necessary to improve the model's capability to address OOD problems and enhance its generalization abilities. The Android application user interface can be seen in Fig 3.

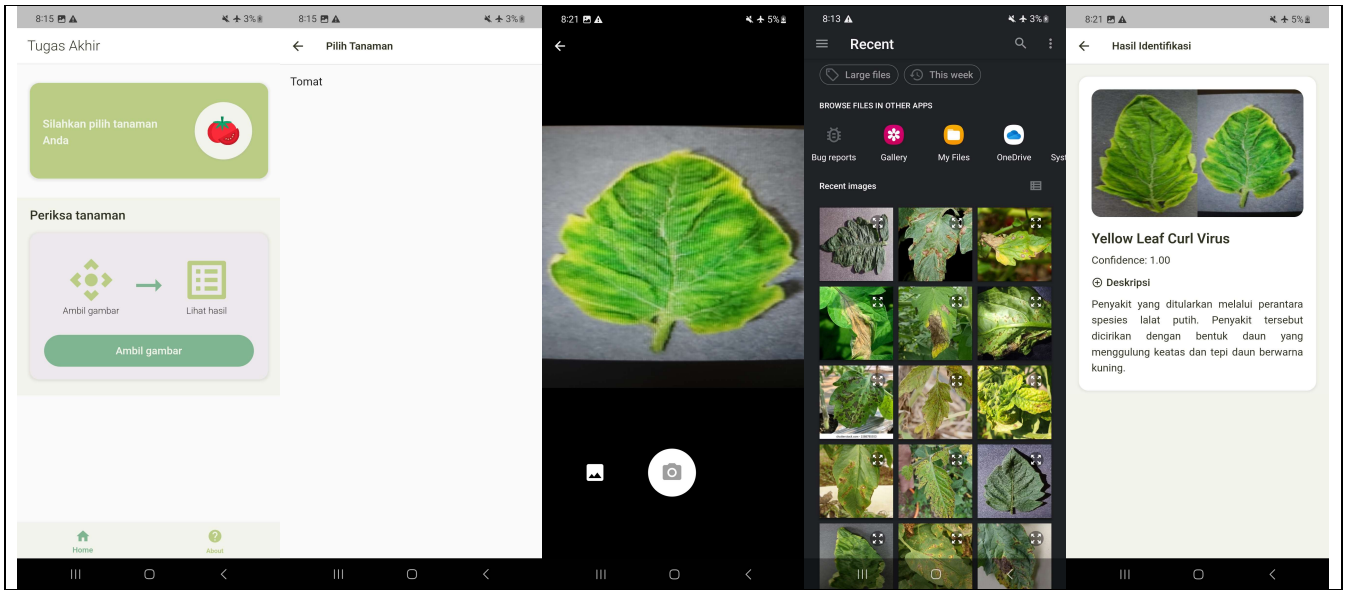

Fig. 3. Android Application User Interface for Plant Disease Detection.

V. CONCLUSION

This paper focuses on implementing a tomato plant leaf disease classification model in Android based application. The model is trained using the transfer learning method and DCGAN data augmentation technique. Through experimentation, several conclusions have been drawn from the preparation of this project. Firstly, the MobileNetV2 architecture is chosen for its smaller size and fewer parameters compared to other architectures. The trained model achieves a highest accuracy rate of 97.83%. Additionally, incorporating DCGAN augmentation data improves the model's accuracy. In scenario 1, the highest accuracy increases from 97.33% to 97.83%, while in scenario 2, it rises from 89.17% to 95.33%. Lastly, the successfully trained MobileNetV2 model is integrated into an Android application, allowing users to select the plant type, capture images from the camera or gallery, and obtain disease classification results. However, it should be noted that there are limitations to the application's consistency when classifying images outside the PlantVillage dataset.

There are two suggestions that can be implemented. Firstly, consider employing a similarity-based approach instead of classification to detect diseases on tomato plant leaves. This approach is less affected by data variations and does not require prior class information. Secondly, assess the quality of images generated by DCGAN using the Frechet Inception Distance (FID) method. This evaluation helps select high-quality DCGAN images that can serve as valuable additional data. Implementing these suggestions will improve the project's robustness and overall performance.

REFERENCES

[1] Pusat Data dan Sistem Informasi Pertanian. (2021). Statistik Pertanian 2021. Kementerian Pertanian Republik Indonesia.

[2] Oerke, E. C. (2005). Crop losses to pests. The Journal of Agricultural Science, 144(1), 31–43. https://doi.org/10.1017/s0021859605005708.

[3] Ahmed, S., Hasan, M. B., Ahmed, T., Sony, M. R. K., & Kabir, M. H. (2022). Less is More: Lighter and Faster Deep Neural Architecture for Tomato Leaf Disease Classification. IEEE Access, 10, 68868–68884. https://doi.org/10.1109/access.2022.3187203.

[4] Ahmad, M., Abdullah, M., Moon, H., & Han, D. (2021). Plant Disease Detection in Imbalanced Datasets Using Efficient Convolutional Neural Networks With Stepwise Transfer Learning. IEEE Access, 9, 140565–140580. https://doi.org/10.1109/access.2021.3119655

[5] Wu, Q., Chen, Y., & Meng, J. (2020). DCGAN-Based Data Augmentation for Tomato Leaf Disease Identification. IEEE Access.