

A deep learning-based mobile app system for visual identification of tomato plant disease

Aurelius Ryo Wang, Nabila Husna Shabrina

Department of Computer Engineering, Faculty of Engineering and Informatics, Universitas Multimedia Nusantara, Tangerang, Indonesia

Article Info

Article history:

Received Aug 4, 2022

Revised Sep 15, 2022

Accepted Oct 1, 2022

Keywords:

Android application

Convolutional neural network

Deep learning

EfficientNetB0

Tomato plant disease

ABSTRACT

Tomato is one of many horticulture crops in Indonesia which plays a vital role in supplying public food needs. However, tomato is a very susceptible plant to pests and diseases caused by bacteria and fungus. The infected diseases should be isolated as soon as it was detected. Therefore, developing a reliable and fast system is essential for controlling tomato pests and diseases. The deep learning-based application can help to speed up the identification of tomato disease as it can perform direct identification from the image. In this research, EfficientNetB0 was implemented to perform multi-class tomato plant disease classification. The model was then deployed to an android-based application using machine learning (ML) kit library. The proposed system obtained satisfactory results, reaching an average accuracy of 91.4%.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Nabila Husna Shabrina

Department of Computer Engineering, Faculty of Engineering and Informatics, Universitas Multimedia Nusantara

Scientia Boulevard Street, Gading Serpong, Tangerang, Indonesia

Email: nabila.husna@umn.ac.id

1. INTRODUCTION

Tomato (*Solanum Lycopersicon*) is one of Indonesia's horticultural commodities with high economic value and great export potential [1]. Tomato plants also play a vital role in meeting the needs of the Indonesian people, both in the food processing industry, daily consumption, and the manufacture of a mixture of processed ingredients [2]. In Indonesia, tomato has become an important crop and part of the nation's economy. However, cultivating tomato plants are prone to disease caused by pests, viruses, and fungi [3]. Infected tomato plants cause harvest losses, making their production unprofitable [4], [5]. Early diagnosis can be carried out by monitoring tomato leaves, as diseases and pests can affect the whole of the plants, including the leaves [6]. Identification from the leaves faster the diagnosis process as it is the visible part of the plants. When the tomato plants are infected, the chlorophyll in the leaves will also be affected [7]. Observing the physical change on the leaves, such as spots, damage, necrosis, and discoloration can indicate that the tomato plants were diseased [6]. However, it is vital to know the specific infection as each requires specific treatment [8]. Therefore, a portable, fast, and reliable system for early detection and diagnosis are required to further prevent the spread of the diseases and minimize production losses.

Recent developments in advanced imaging techniques made it possible to perform those tasks reliably [9]–[13]. Those techniques were carried out by retrieving features of the images, such as size, shape, and color [14]. The techniques were implemented both using image processing or machine learning-based methods. In the past few years, deep learning has had tremendous applications for recognizing and classifying plant diseases. In some experiments, deep learning-based methods performed superior to humans

in large-scale tasks. Several recent studies [15]–[20] showed that convolutional neural networks (CNN), a deep learning based method, can be implemented for disease classification in several crops such as mangoes, apples, maize, rice, corn, melons, and wheat.

Deep learning can also be applied in tomato plants related tasks, such as tomato classification, tomato detection, tomato disease identification, and tomato pests detection, as surveyed in [21]. The application of deep learning in identifying tomato plant diseases was proven to provide satisfactory results. CNN method using the SqueezeNet architecture applied to 1,400 tomato plant diseases consisting of seven classes. The applied method produces an accuracy of 86.92% [22]. A CNN-based method was implemented using 4,923 controlled conditions images of diseased and healthy tomato plant leaves. The CNN classification model obtains an accuracy of 95.75 percent, while the F-RCNN detection model produced a confidence score of 80% [23]. Agarwal *et al.* [24] implemented a proposed method along with three different models of pre-trained CNN. The accuracy ranges from 76% to 100%, and the model's average accuracy for the nine disease classes and one healthy class is 91.2%. Pre-trained AlexNet and VGG-16 models were also applied to perform tomato crop disease classification in [25]. The AlexNet and VGG16 models provided an accuracy of 97.49% and 97.23%, respectively.

The previous research mainly focuses on finding the best deep learning model to detect and classify tomato plant diseases. However, they do not include application deployment for real-time detection and classification. This research deploys a lightweight CNN model into a mobile application for real-time early diagnosis of tomato plant disease. The pre-trained EfficientNet B0 model was chosen to be implemented due to its smaller size, outstanding performance, and satisfactory inference time [26]. The applied model will classify images of diseased tomato leaves taken from the PlantVillage datasets [27] with nine tomato plant disease classes. The trained model was then deployed to an Android application to be easily accessible to users, especially farmers, for tomato disease classification. The mobile application will take an image via a smartphone camera and provide classification results with brief information about tomato plant disease.

2. RESEARCH METHOD

2.1. Research workflow

Figure 1 shows an overview of the research workflow for developing a deep learning-based mobile application for tomato plant disease classification. Initially, the PlantVillage datasets were arranged using dataset balancing and were divided into a train, validation, and test datasets. Image rotation, brightness, flip, shifting, shear, and zoom were then applied for augmentation. The datasets are then used as input to train the EfficientNet B0 model. The trained model was then deployed into an android-based application. The evaluation method was conducted on the model results and application performance. The EfficientNet B0 classification results were recorded and evaluated using four metric evaluations. The performance of the deep learning-based mobile application is measured using the processing time the model makes predictions and the time the endpoint processes an incoming request.

2.2. Dataset arrangements

The deep learning model will classify images of diseased tomato leaves taken from The PlantVillage datasets [27], an open-source dataset that can be accessed publicly through the website www.plantvillage.org. This dataset has more than 50,000 data consisting of 14 classes of food plant diseases. Food crops in this dataset include apples, blueberries, cherries, corn, grapes, oranges, peaches, peppers, potatoes, raspberries, soybeans, pumpkins, strawberries, and tomatoes. For tomato plants, there are nine types of diseases consisting of *Alternaria solani* (early blight), *Septoria Lycopersicon* (Septoria), *Corynespora cassiicola* (target spot), *Fulvia fulva* (leaf mold), *Xanthomonas campestris pv Vesicatoria* (bacterial spot), *Phytophthora Infestans* (late blight), Tomato Yellow Leaf Curl Virus, *Tomato Mosaic Virus* (mosaic virus), and *Tetranychus urticae* (spider mites). There is a total of 18,162 images, including healthy tomatoes, which have been researched and validated by the plant pathologist. Figures 2(a) to 2(i) presents the sample of tomato disease images from the dataset late blight, early blight, target spot, bacterial spot, yellow leaf curl virus, leaf mold, Septoria, mosaic virus, and spider mites. Table 1 shows some statistics on tomato plant disease from the PlantVillage dataset.

In this study, ten classes from the PlantVillage dataset and the CIFAR 100 dataset (for representing the unknown images) are used. There is a total of eleven classes implemented in the model. The dataset balancing was then applied to equalize the number of images per class. The image limit per class used in this model was 700 images. After performing dataset balancing, the dataset will be divided into three sets: train, validation, and test with the ratio of 80:10:10. To summarize, the dataset consists of 6,160, 770, and 770 for training, validation, and test, respectively.

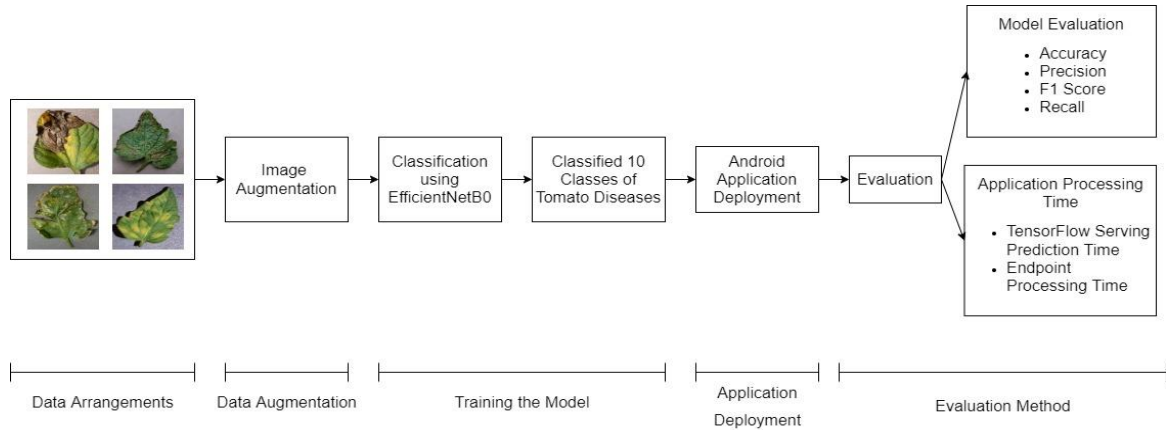


Figure 1. Research workflow

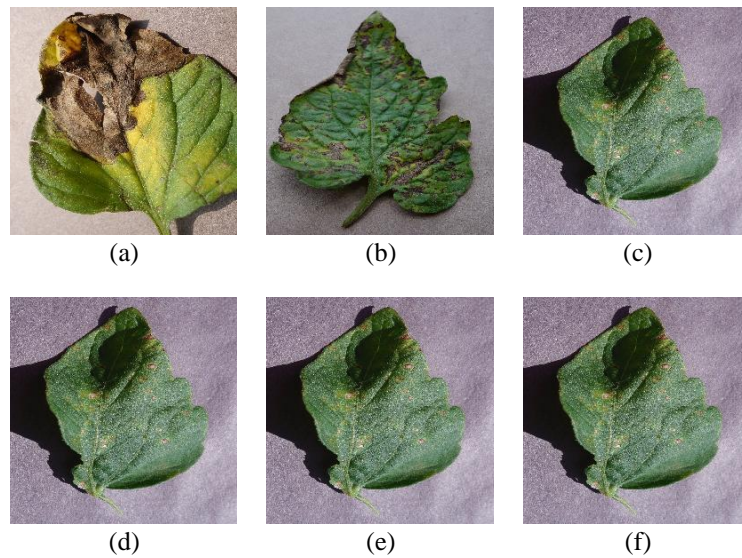


Figure 2. Sample of tomato plant disease from PlantVillage datasets (a) late blight, (b) early blight, (c) target spot, (d) bacterial spot, (e) yellow leaf curl virus, (f) leaf mold, (g) Septoria, (h) mosaic virus, and (i) spider mites [26]

Table 1. Class distribution in the dataset [26]

| Class | No. of image |
|------------------------|--------------|
| Yellow leaf curl virus | 5,357 |
| Bacterial spot | 2,127 |
| Late blight | 1,910 |
| Septoria leaf spot | 1,771 |
| Spider mites | 1,676 |
| Healthy | 1,592 |
| Target spot | 1,404 |
| Early blight | 1,000 |
| Leaf mold | 952 |
| Mosaic virus | 373 |

2.3. Data augmentation

Data augmentation uses image manipulation to increase the diversity and quantity of datasets. The performance of models can be enhanced with the proper augmentation techniques. This research applied augmentation to the train set using several augmentations described below. The sample result of image augmentation is presented in Figure 3.

- Image rotation was applied randomly from 0 to 90 degrees.
- Image brightness was applied with a random brightness parameter varying from 0.4 to 1.
- Image flip was applied using random vertical and horizontal flip.
- Shifting was applied using a random parameter of 0.2 for vertical and horizontal part.
- Shear was applied randomly from 0 to 40 degrees.
- Zoom was applied with a parameter varying from 0 to 0.15.



Figure 3. Sample of augmented image using the combination of rotation, brightness, flip, shifting, shear, and zoom

2.4. Training the model

2.4.1. Convolutional neural networks

One of the most well-known and widely utilized deep learning networks is the CNN [28]. The primary advantage of CNN over its forerunners is that it recognizes the pertinent features automatically without human intervention [29]. CNNs consist of connected neurons, similar to a multi-layer perceptron (MLP) structure. Each neuron and the connections between neurons have a weight, bias, and activation function. The natural structure of neurons inspired by the structure of neurons in human and animal brains. CNN works by performing 2-dimensional convolution using a convolution kernel to an image. CNN does not require as much dataset processing as other algorithms. Feature extraction is done automatically [30]. CNN's ability to perform feature extraction results in a high classification capability. CNN consist of three primary layers [31] described below:

- Convolutional layer: The input image's feature extraction process, such as colors, gradients, and edges, is carried out on a convolutional layer. This extraction process uses a convolution operation using a kernel in the form of a matrix.
- Pooling layer: The pooling layer is a layer that aims to extract information from features that have been taken in the previous layer. Extracting this information uses the portion of the image that the kernel has skipped. In this layer, there are two types of pooling: max pooling, which takes the most significant value, and average pooling, which takes the average value.
- Fully connected layer: The fully connected layer collects the results from the previous layer to determine the input image class. The data in the previous layer will be transformed first so that the model can classify linearly.

2.4.2. EfficientNet B0 architecture

The general CNN model is developed with limited resources (computation and memory), and if resources are available, the model is enlarged by repeating blocks at a specific layer. A study in [32], was proposed a new model capacity enhancement method: changing the model's width, depth, and resolution scale with a combined coefficient. Neural architecture search (NAS) [33], was based on these observations to design a new family model called EfficientNet. One variation of EfficientNet (EfficientNet-B7) achieves state-of-the-art performance on ImageNet datasets with a much smaller model size than the other model. The inference speed is also much faster. This model uses a convolution block named MBConv [32].

EfficientNet has eight models starting from EfficientNet B0 to EfficientNet B7. The difference between each model is the total number of their parameters. The smaller version has smaller parameters compared to the bigger version. For example, The EfficientNet B0 has a total of 5.3 million parameters, while EfficientNet B7 has 66 million parameters. In this study, EfficientNet B0 was chosen because the size is the smallest among the other versions. Despite the small size and parameters, EfficientNet B0 still has satisfactory performance, as it has a top-1 accuracy of 77.1% and top-5 accuracy of 93.3%, which refers to the model's performance on the ImageNet validation dataset [26]. The EfficientNet architecture proposed by [32] is given in Table 2.

Table 2. EfficientNet architecture

| Operator | Resolution | Channel | Total layer |
|----------------------------|------------|---------|-------------|
| Conv3×3 | 224×224 | 32 | 1 |
| MBConv1, k3×3 | 112×112 | 16 | 1 |
| MBConv6, k3×3 | 112×112 | 24 | 2 |
| MBConv6, k5×5 | 56×56 | 40 | 2 |
| MBConv6, k3×3 | 28×28 | 80 | 3 |
| MBConv6, k5×5 | 14×14 | 112 | 3 |
| MBConv6, k5×5 | 14×14 | 192 | 4 |
| MBConv6, k3×3 | 7×7 | 320 | 1 |
| Conv1×1 and Pooling and FC | 7×7 | 1280 | 1 |

The training and optimization of the model is a challenging and time-consuming process. The training needs a powerful GPU and millions of training samples. However, deep learning's use of transfer learning addresses all issues and provides a solution. Transfer learning uses a pre-trained CNN optimized for a single task and spreads information across many modes. Transfer learning uses a model trained with a general dataset, so that the model can specialize in smaller datasets [34]. This method can reduce training time and improve overall model performance.

In this research, the model will be trained using the EfficientNet B0 model with the transfer learning method. The model architecture implemented in this study is shown in Figure 4. When implementing the transfer learning method, there are stages consisting of creating a classification layer, freezing the remaining layer, setting up the learning rate, and compiling the model that should be applied. The new classification layer uses the GlobalAveragePooling2D and dropout layers with a value of 0.5. The purpose of using this layer was to avoid overfitting during training. The Dense layer, a neural network layer, was used as a classification task based on the results of the convolutional layer. The last layer was Dense (11), a final classification layer for the output prediction.

The freeze remaining layer stage aims to freeze the learning layer of the EfficientNet B0 model. Those layers were intended to keep the features previously studied by the EfficientNet B0 model. After freezing the previous layer, the learning rate, optimizer function, and loss function were set on the model. The learning rate of 0.001 and Adam optimizer were implemented in this research. The categorical cross entropy was also applied for multiclass classification. Learning rate aims to adjust the learning speed of the model as accuracy increases in order to avoid decreasing accuracy. The optimizer functions to manage attributes such as the learning rate of the model in learning.

The model is then trained using a configuration consisting of a batch size of 32 and epochs of 30. The model also has a callback function that is executed every epoch. The callbacks are EarlyStopping which stops learning to prevent overfitting, and ReduceLROnPlateau, to reduce the learning rate value if there is no significant change when it has passed an epoch value.

2.5. Application deployment

ML Kit is a machine learning library by Google that can be used openly by mobile application developers [35]. ML Kit provides various features such as barcode scanning, face detection, object detection, and tracking. Each feature can support Android and iOS operating systems. Moreover, each feature is also

equipped with documentation that mobile application developers can apply. In the object detection and tracking feature, there is a streaming mode and a single image mode, where the mode can be set by the developer when creating the application.

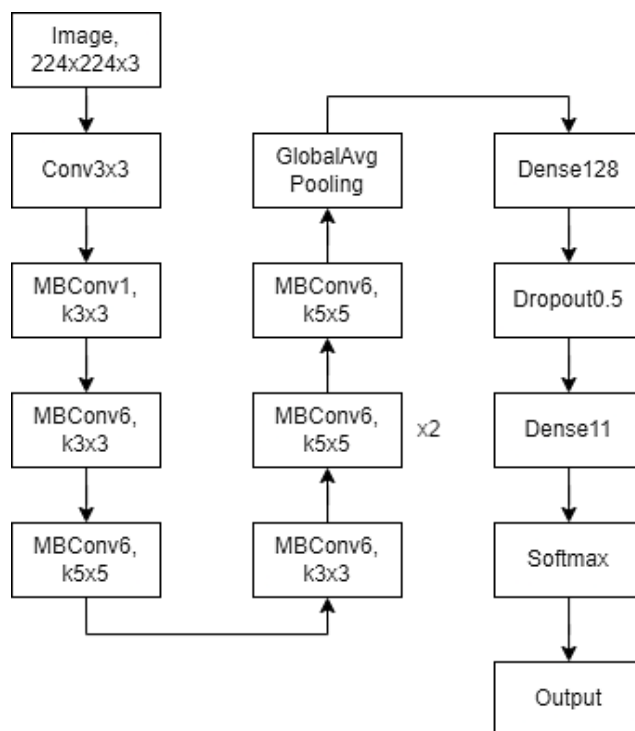


Figure 4. EfficientNet B0 model implemented in this study

The deep learning-based mobile application for identifying tomato plant disease developed in this study consists of three features: live detection, object detection, and image classification. The live detection feature detects tomato plant diseases in real time with the help of object detection using the ML Kit library by Google. This feature aims to look for diseased tomato plants. The multiple object detection feature uses object detection to detect objects and send images of those objects to the application programming interface (API). This feature aims to identify the leaves contained in an image. The image classification image feature performs the classification of the images captured by the camera. This mobile application will take an image via a smartphone camera and provide classification results with brief information about tomato plant disease.

The proposed mobile application system consists of two modules: frontend and backend. The frontend module is an Android application built using the Android Studio IDE, which provides the user view. In the frontend module, object detection using the ML Kit library is applied to extract objects in an image. When an object is detected, the image of the object will be sent to the backend for prediction. The second module is the backend module consists of endpoint and machine learning. The endpoint is used to handle images sent by the frontend module. The endpoint will make predictions by processing the received image and sending the results to machine learning as a backend using TensorFlow Serving. When receiving an image from the endpoint, TensorFlow Serving will make predictions and send back the prediction results to the endpoint, and the results will be forwarded to the frontend. After the frontend module receives the prediction results, the frontend module will display the detected object page specifically for the real-time and multiple object detection features. Each selected object will redirect the page to a results page that displays predictions of tomato plant diseases and a brief of information on how to prevent them using the help of Google Search. When detected in the Still Image feature, the page will change directly to the prediction result page. The complete module flow is given in Figure 5.

The frontend module also contains image processing stages that aim to reduce the image size before sending it to the backend module. The process includes resizing the image to 50% of its previous size, formatting it to JPEG, and converting it to a Base64 String. The backend module will continue the preprocessing steps to change the image channel to red-green-blue (RGB). The machine learning inputs will also adjust the size to 224x224 pixels.

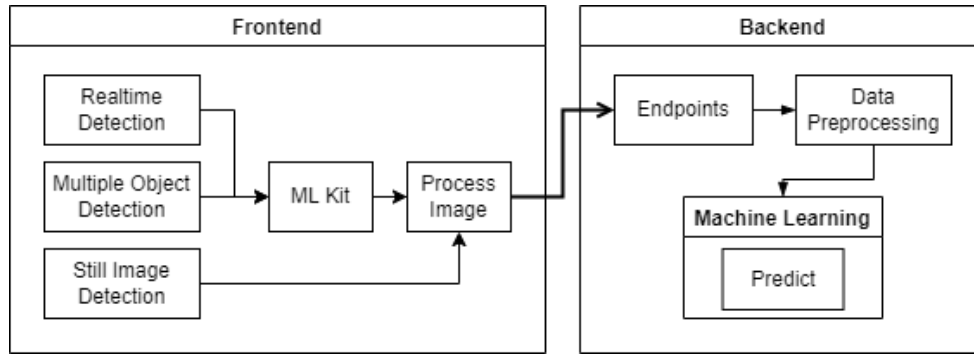


Figure 5. Complete module flow

2.6. Evaluation method

The evaluation processes were employed in model evaluation and the android application's performance. The model evaluation consists of several metrics used to determine the model's performance. The evaluation was applied using the test set. The accuracy was used to determine the proportion of correctly predicted classes in all the analyzed samples. The precision metric was used to correctly determine the positive patterns predicted by every other pattern in a positive class. The recall metric was utilized to determine the percentage of positive patterns that are classified correctly. Then the F1-score metric was applied to analyze the harmonic average between recall and precision. The formula for accuracy, precision, recall, and F1-score are given in (1)-(4), respectively.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

$$F1 \text{ score} = \frac{2 \times Precision \times Recall}{Precision+Recall} \quad (4)$$

The android application's performance was carried out by calculating the processing time between the model makes predictions and the time the endpoint processes an incoming request. The request processing time was tested by measuring when the backend TensorFlow Serving made predictions on an image. The provided image was 256×256, converted into a NumPy Array.

2.7. System implementation

Machine learning code was implemented using Keras and TensorFlow Library, which offer fully built models with pretrained weights on the ImageNet dataset. Model training was performed using Google Colab Notebook Pro, which has a minimum specification of NVIDIA P100 or T4 as GPU, Xeon Processor 2.3 GHz as the CPU, and memory up to 32 GB. The mobile application was built in Realme GT Neo2 phone with Android 12 OS, Chipset Snapdragon 870 5G, GPU Adreno 650, RAM of 12+7 GB extension, and Camera with the specification of 64, 8 and 2 MP. The frontend of the mobile application was implemented using IDE Android Studio with the support of library object detection by ML Kit library by Google. The live preview utilized the library CameraX.

3. RESULTS AND DISCUSSION

3.1. Model performance result

The loss and accuracy progression during training is shown in Figures 6 and 7, respectively. Figure 6 presents the loss during each epoch (number of training cycles through the whole dataset), while the Figure 7 presents the model's accuracy. The model executes the "ReduceLROnPlateau" callback function on epochs 24 and 28. At the end of the training, the model produces a training accuracy of 92.35% and a validation accuracy of 92.221%. The training and validation loss obtained by the model was 0.2302 and 0.2191, respectively.

The test accuracy obtained by the model reached 91.43%. This shows that the model has satisfactory results compared to previous research [22], [24]. The confusion matrix for the model is given in Figure 8, while the complete model performance result for precision, recall, and F1-score for each class is presented in Table 3. The result of the performance metric shows that the model performs prominently.

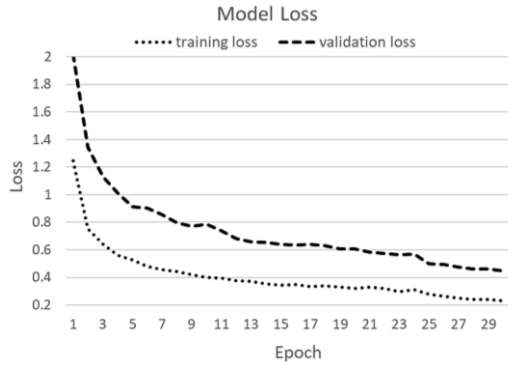


Figure 6. Model loss

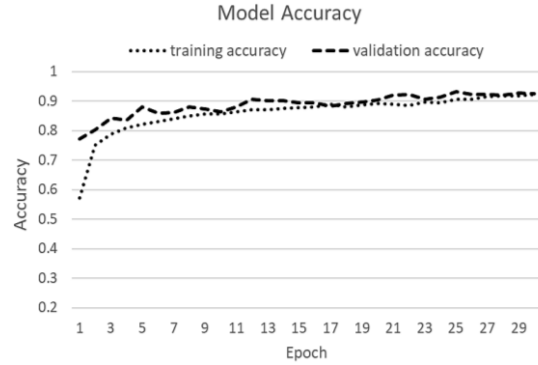


Figure 7. Model accuracy

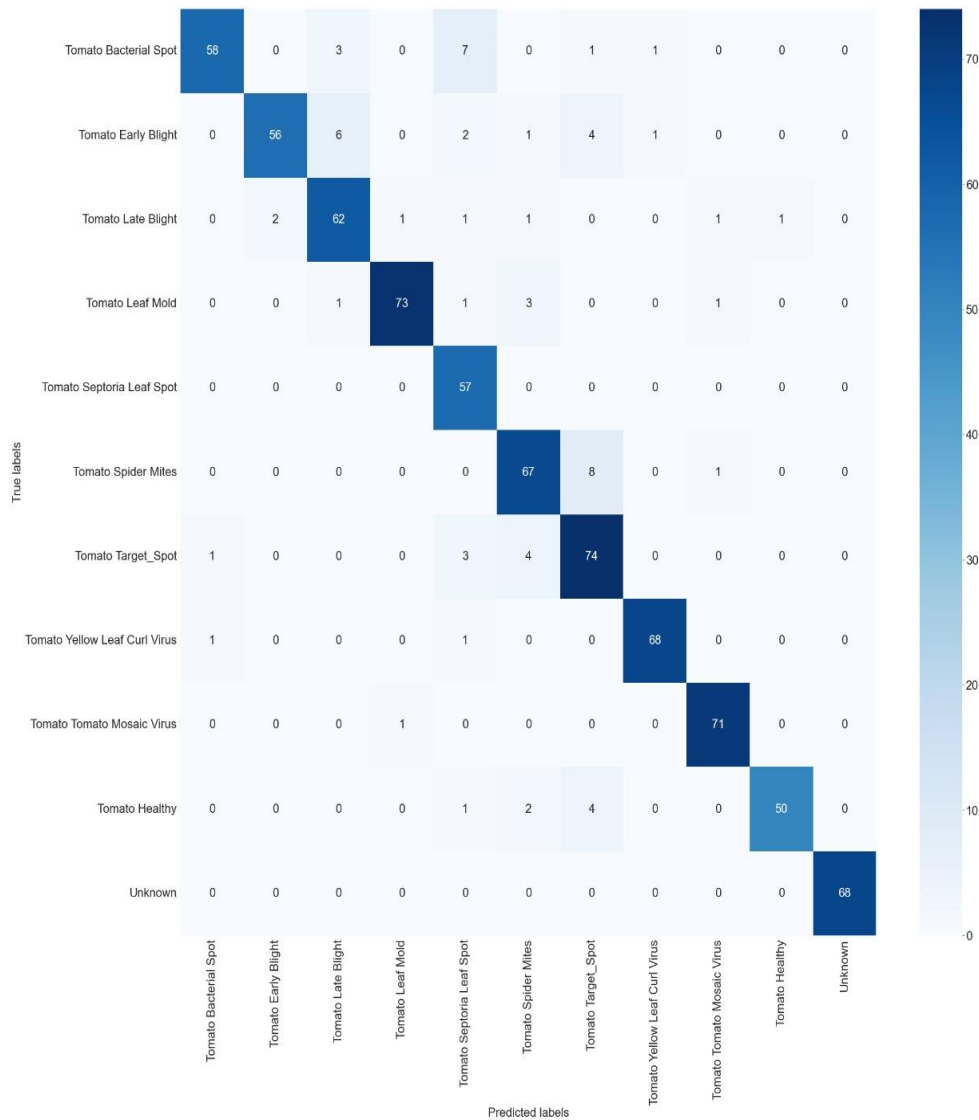


Figure 8. Confusion matrix of the model

Table 3. Model performance

| Class | Precision | Recall | F1-score |
|------------------------|-----------|--------|----------|
| Yellow leaf curl virus | 0.97 | 0.97 | 0.97 |
| Bacterial spot | 0.97 | 0.83 | 0.89 |
| Late blight | 0.86 | 0.90 | 0.88 |
| Septoria leaf spot | 0.78 | 1.00 | 0.88 |
| Spider mites | 0.86 | 0.88 | 0.87 |
| Healthy | 0.98 | 0.88 | 1.00 |
| Target spot | 0.81 | 0.90 | 0.86 |
| Early blight | 0.97 | 0.80 | 0.88 |
| Leaf mold | 0.97 | 0.92 | 0.95 |
| Mosaic virus | 0.96 | 0.99 | 0.97 |
| Unknown | 1.00 | 1.00 | 1.00 |

3.2. Mobile application performance

The complete navigation graph of the mobile application is presented in Figure 9. The application's home page consists of three main menus, as shown in Figure 10. The sample of each menu is given in Figure 11, where in Figure 11(a) shows the real-time detection and Figure 11(b) shows the multiple objects and still images. The sample for the result page is shown in Figure 12. Figure 12 (a) shows the detected objects page for real-time and multiple object features, and Figure 12(b) result page of the detected object.

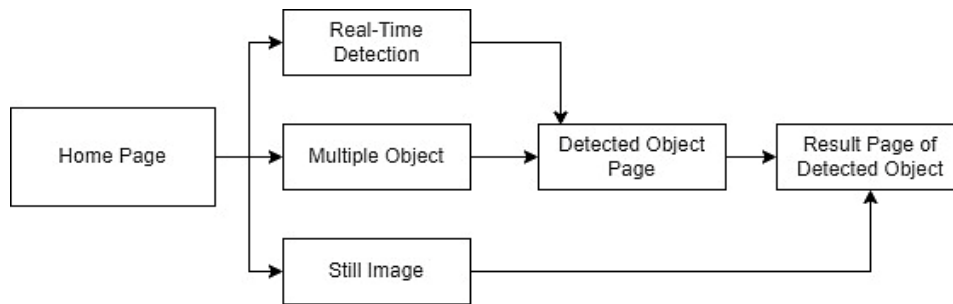


Figure 9. Mobile application navigation graph

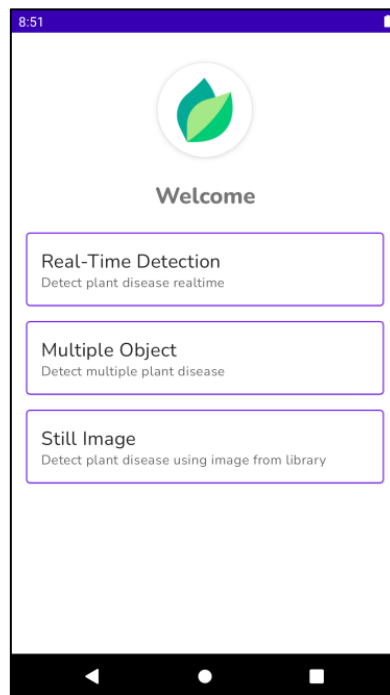


Figure 10. The home page

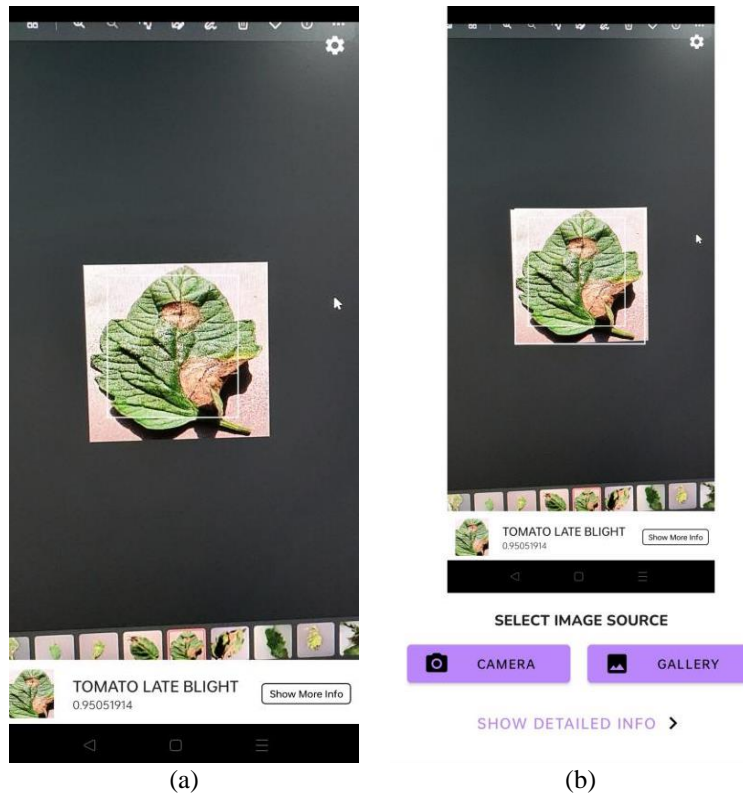


Figure 11. The appearance for each menu: (a) real-time detection and (b) multiple objects and still image

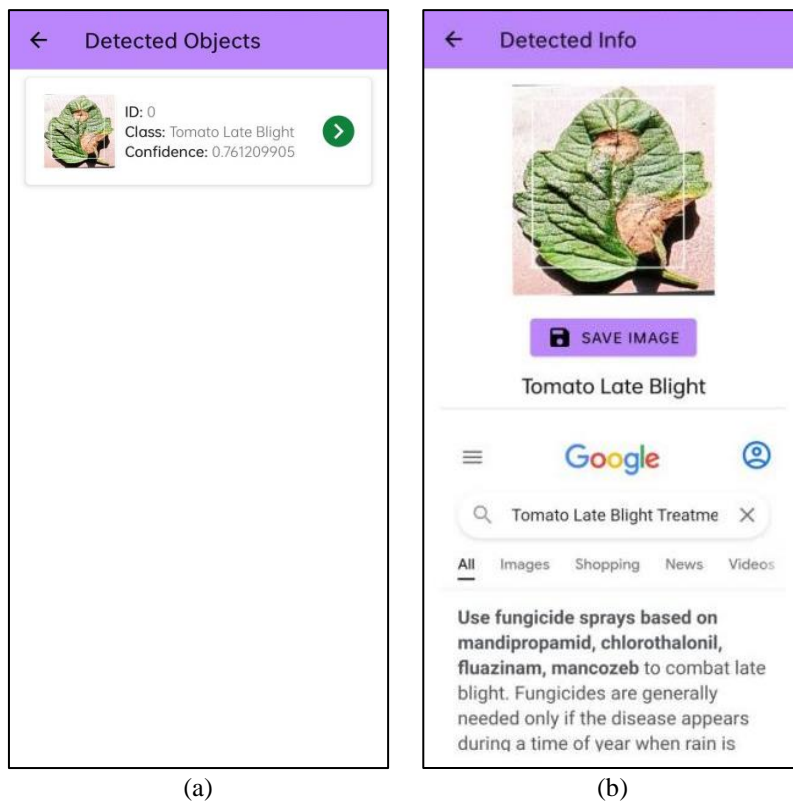


Figure 12. The sample for the result page: (a) detected objects page for real-time and multiple object feature and (b) result page of detected object

The mobile application's performance was carried out by calculating the processing time between the model makes predictions and the time the endpoint processes an incoming request. The test was carried out ten times. The result is given in Table 4.

Based on testing the TensorFlow backend prediction time, the average prediction time was 40.3744 ms, with the longest time of 60,927 ms and the fastest time of 31,835 ms. It was shown that the time when the backend TensorFlow serving predicts an image was considerably fast.

The endpoint processing time was carried out in three different formats of input: JPEG image, with the size of 256×256; string base 64 format, to accommodate other types of the image such as PNG but converted in a smaller size; and string base 64×5 format, a same type with the second format but multiplied by five to accommodate the multiple object detection. The result of the endpoint processing time is given in Table 5.

In the jpeg image data type, the average processing time for a request was 149.289 ms, while for string base 64 and string base 64×5 was 103.183 and 527.875 ms, respectively. The fastest processing time was 91.99 ms in JPEG images, 56.72 ms in string base 64, and 348.85 ms in string base 64×5. The slowest processing time were 185.35, 200.61, and 738.9 ms in JPEG, string base 64, and string base 64×5, respectively. The string base 64×5 format gave the overall longest processing time due to its bigger image, which contains five images at once.

Table 4. Result of TensorFlow serving prediction time

| Data Sent | Response time (ms) |
|--------------|--------------------|
| Test 1 | 41.683 |
| Test 2 | 35.907 |
| Test 3 | 45.016 |
| Test 4 | 39.87 |
| Test 5 | 60.927 |
| Test 6 | 34.96 |
| Test 7 | 31.835 |
| Test 8 | 36.641 |
| Test 9 | 39.621 |
| Test 10 | 37.364 |
| Average time | 40.3744 |

Table 5. Result of endpoint processing time

| Data Sent | Data type | | |
|--------------|-----------------|---------------------|-----------------------|
| | JPEG image (ms) | String base 64 (ms) | String base 64×5 (ms) |
| Test 1 | 104.49 | 108 | 657.82 |
| Test 2 | 185.35 | 58.28 | 671.22 |
| Test 3 | 162.04 | 94.98 | 463.61 |
| Test 4 | 154.86 | 113.76 | 738.9 |
| Test 5 | 172.71 | 97.27 | 523.34 |
| Test 6 | 170.62 | 200.61 | 393.5 |
| Test 7 | 154.09 | 105.55 | 348.85 |
| Test 8 | 152.47 | 97.94 | 509.6 |
| Test 9 | 144.27 | 56.72 | 401.3 |
| Test 10 | 91.99 | 98.72 | 570.61 |
| Average time | 149.289 | 103.183 | 527.875 |

4. CONCLUSION

The deep learning-based mobile application implemented using EfficientNet B0 model is giving promising results. The implementation will support the early diagnosis of tomato plant disease identification. The performance of the EfficientNet B0 was outstanding as it reached an accuracy of 91.4%. The implementation of deep learning in the mobile application also exhibited a satisfactory result with an average prediction time of 40.3744 ms and the fastest average processing time of 103.183 ms. Future work is still necessary to improve the model accuracy and reliability along with the improvement of the mobile application. Future work will focus on further development using other deep learning models to improve the performance of classification results.




ACKNOWLEDGMENTS

We would like to thank the supports given by Universitas Multimedia Nusantara during this research.




REFERENCES

- [1] T. D. Davis and P. Hariyadi, "Horticultural research and education opportunities in Indonesia," *HortScience*, vol. 48, no. 3, pp. 292–295, Mar. 2013, doi: 10.21273/HORTSC.48.3.292.
- [2] P. Amoako Ofori, S. Owusu-Nketia, F. Opoku-Agyemang, D. Agbleke, and J. N. Amisah, "Greenhouse tomato production for sustainable food and nutrition security in the tropics," in *Tomato - From Cultivation to Processing Technology*, IntechOpen, 2022.
- [3] V. K. Singh, A. K. Singh, and A. Kumar, "Disease management of tomato through PGPB: current trends and future perspective," *3 Biotech*, vol. 7, no. 4, Aug. 2017, doi: 10.1007/s13205-017-0896-1.
- [4] M. M. Ali, N. A. Bachik, N. 'Atirah Muhadi, T. N. Tuan Yusof, and C. Gomes, "Non-destructive techniques of detecting plant diseases: a review," *Physiological and Molecular Plant Pathology*, vol. 108, Dec. 2019, doi: 10.1016/j.pmpp.2019.101426.
- [5] I. K. Arah, H. Amaglo, E. K. Kumah, and H. Ofori, "Preharvest and postharvest factors affecting the quality and shelf life of harvested tomatoes: a mini review," *International Journal of Agronomy*, vol. 2015, pp. 1–6, 2015, doi: 10.1155/2015/478041.
- [6] J. B. Jones, T. A. Zitter, T. M. Momol, and S. A. Miller, Eds., *Compendium of tomato diseases and pests, second edition*. The American Phytopathological Society, 2016.
- [7] M. Meena, A. Zehra, M. K. Dubey, M. Aamir, V. K. Gupta, and R. S. Upadhyay, "Comparative evaluation of biochemical changes in tomato (*Lycopersicon esculentum* Mill.) infected by *Alternaria alternata* and its toxic metabolites (TeA, AOH, and AME)," *Frontiers in Plant Science*, vol. 7, Sep. 2016, doi: 10.3389/fpls.2016.01408.
- [8] S. A. Miller, M. L. Lewis Ivey, F. Baysal-Gurel, and Xiulan Xu, "A systems approach to tomato disease management," *Acta Horticulturae*, no. 1069, pp. 167–172, Feb. 2015, doi: 10.17660/ActaHortic.2015.1069.23.
- [9] L. Li, S. Zhang, and B. Wang, "Plant disease detection and classification by deep learning—a review," *IEEE Access*, vol. 9, pp. 56683–56698, 2021, doi: 10.1109/ACCESS.2021.3069646.
- [10] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, pp. 1083–1087, Sep. 2016, doi: 10.3389/fpls.2016.01419.
- [11] V. Singh, N. Sharma, and S. Singh, "A review of imaging techniques for plant disease detection," *Artificial Intelligence in Agriculture*, vol. 4, pp. 229–242, 2020, doi: 10.1016/j.aiaa.2020.10.002.
- [12] A. Abade, P. A. Ferreira, and F. de Barros Vidal, "Plant diseases recognition on images using convolutional neural networks: a systematic review," *Computers and Electronics in Agriculture*, vol. 185, Jun. 2021, doi: 10.1016/j.compag.2021.106125.
- [13] M. A. F. Azlah, L. S. Chua, F. R. Rahmad, F. I. Abdullah, and S. R. Wan Alwi, "Review on techniques for plant leaf classification and recognition," *Computers*, vol. 8, no. 4, Oct. 2019, doi: 10.3390/computers8040077.
- [14] A. K. Jain and A. Vailaya, "Image retrieval using color and shape," *Pattern Recognition*, vol. 29, no. 8, pp. 1233–1244, Aug. 1996, doi: 10.1016/0031-3203(95)00160-3.
- [15] K. Kusriani *et al.*, "Data augmentation for automated pest classification in mango farms," *Computers and Electronics in Agriculture*, vol. 179, Dec. 2020, doi: 10.1016/j.compag.2020.105842.
- [16] P. Jiang, Y. Chen, B. Liu, D. He, and C. Liang, "Real-time detection of apple leaf diseases using deep learning approach based on improved convolutional neural networks," *IEEE Access*, vol. 7, pp. 59069–59080, 2019, doi: 10.1109/ACCESS.2019.2914929.
- [17] A. A. Alfariy, Q. Chen, and M. Guo, "Deep learning based classification for paddy pests and diseases recognition," in *Proceedings of 2018 International Conference on Mathematics and Artificial Intelligence*, Apr. 2018, pp. 21–25, doi: 10.1145/3208788.3208795.
- [18] W. Tan, C. Zhao, and H. Wu, "Intelligent alerting for fruit-melon lesion image based on momentum deep learning," *Multimedia Tools and Applications*, vol. 75, no. 24, pp. 16741–16761, Sep. 2016, doi: 10.1007/s11042-015-2940-7.
- [19] X. Zhang, Y. Qiao, F. Meng, C. Fan, and M. Zhang, "Identification of maize leaf diseases using improved deep convolutional neural networks," *IEEE Access*, vol. 6, pp. 30370–30377, 2018, doi: 10.1109/ACCESS.2018.2844405.
- [20] Z. Lin *et al.*, "A unified matrix-based convolutional neural network for fine-grained image classification of wheat leaf diseases," *IEEE Access*, vol. 7, pp. 11570–11590, 2019, doi: 10.1109/ACCESS.2019.2891739.
- [21] S. M. Saranya, R. R. Rajalaxmi, R. Prabavathi, T. Suganya, S. Mohanapriya, and T. Tamilselvi, "Deep learning techniques in tomato plant - a review," *Journal of Physics: Conference Series*, vol. 1767, no. 1, Feb. 2021, doi: 10.1088/1742-6596/1767/1/012010.
- [22] A. Hidayatuloh, M. Nursalman, and E. Nugraha, "Identification of tomato plant diseases by leaf image using squeezeNet model," in *2018 International Conference on Information Technology Systems and Innovation (ICITSI)*, Oct. 2018, pp. 199–204, doi: 10.1109/ICITSI.2018.8696087.
- [23] R. G. de Luna, E. P. Dadios, and A. A. Bandala, "Automated image capturing system for deep learning-based tomato plant leaf disease detection and recognition," in *2018 IEEE Region 10 Conference*, Oct. 2018, pp. 1414–1419, doi: 10.1109/TENCON.2018.8650088.
- [24] M. Agarwal, A. Singh, S. Arjaria, A. Sinha, and S. Gupta, "ToLeD: tomato leaf disease detection using convolution neural network," *Procedia Computer Science*, vol. 167, pp. 293–301, 2020, doi: 10.1016/j.procs.2020.03.225.
- [25] A. K. Rangarajan, R. Purushothaman, and A. Ramesh, "Tomato crop disease classification using pre-trained deep learning algorithm," *Procedia Computer Science*, vol. 133, pp. 1040–1047, 2018, doi: 10.1016/j.procs.2018.07.070.
- [26] Keras, "Keras applications," *Keras*, Accessed: Feb. 05, 2022. [Online]. Available: <https://keras.io/api/applications>.
- [27] D. P. Hughes and M. Salathé, "An open access repository of images on plant health to enable the development of mobile disease diagnostics," *arXiv preprint arXiv:1511.08060*, Nov. 2015.
- [28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.
- [29] J. Gu *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, May 2018, doi: 10.1016/j.patcog.2017.10.013.
- [30] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537-021-00444-8.
- [31] L. Min, C. Qiang, and Y. Shuicheng, "Network in network," *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, Dec. 2014.
- [32] M. Tan and Q. V. Le, "EfficientNet: rethinking model scaling for convolutional neural networks," in *36th International Conference on Machine Learning*, 2019, pp. 10691–10700.
- [33] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search," in *Journal of Machine Learning Research*, vol. 20, Springer International Publishing, 2019, pp. 63–77.
- [34] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, Sep. 2021, doi: 10.1007/s12525-021-00475-2.
- [35] "Machine learning for mobile developers," *Google Developers*. <https://developers.google.com/ml-kit> (accessed Apr. 04, 2022).

BIOGRAPHIES OF AUTHORS

Aurelius Ryo Wang    is currently an undergraduate student in Computer Engineering at Universitas Multimedia Nusantara. His areas of interest are data science, mobile application, and backend development. He can be contacted at email: ryowang135@gmail.com.



Nabila Husna Shabrina    received her bachelor's degree in Telecommunication Engineering and master's degree In Electrical Engineering from Institut Teknologi Bandung. She is currently a lecturer in Computer Engineering at Universitas Multimedia Nusantara. Her research interest includes communication system, signal processing, image processing, and computer vision. She can be contacted at email nabila.husna@umn.ac.id.