International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-2, December 2019

Plant Disease Classification using Lite Pretrained Deep Convolutional Neural Network on Android Mobile Device

Burhanudin Syamsuri, Gede Putra Kusuma

Abstract: ***The implementation of image recognition in agriculture to detect symptoms of plant disease using deep learning Convolutional Neural Network (CNN) models are proven to be highly effective. The computational efficiency by using CNN, made possible to run the application on mobile device. To optimize the utilization of mobile device and choosing the most effective CNN model to run as detection system in mobile device with the highest accuracy and low resource consumption is proposed in this paper. In this study, PlantVillage dataset which extended to coffee leaf, were tested and compared using three CNN models, two models which specifically designed for mobile, MobileNet and Mobile Nasnet (MNasNet), and one model that recognized for its accuracy on personal computer (PC), InceptionV3. The experiment executed on both mobile and PC found a slightly degradation on accuracy when the application is running on mobile. InceptionV3 experienced the most persistence model compares to MNasNet and MobileNet. Yet, InceptionV3 had biggest latency time. The final result on mobile device recorded InceptionV3 achieved highest accuracy of 95.79%, MNasNet 94.87%, and MobileNet 92.83%, while for time latency MobileNet achieved the lowest with 394.70 ms, MNasnet 430.20 ms, and InceptionV3 2236.10 ms respectively. It is expected that the outcome of this study will be of great benefit to farmers as mobile image recognition would help them analyze the condition of their plants on site simply by taking a picture of the leaf and running the experiment on their mobile device.***

Keywords**: Deep learning, CNN, Pretrained model, TFLite, Mobile application.**

I. INTRODUCTION

Indonesia is among the biggest coffee producers in the world[1], as stated by the Ministry of the Industry of the Republic of Indonesia in Table- I, which shows countries with highest productivity in the world.

Table- I: Biggest four world coffee producers

No	Country	Production(ton/year)
1	Brazil	2,900,000
2	Vietnam	1,600,000
3	Colombia	840,000
4	Indonesia	639,000

Revised Manuscript Received on December 30, 2019.
***** Correspondence Author
Burhanudin Syamsuri, Department of Computer Science, BINUS Graduate Program - Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia, 11480.
Gede Putra Kusuma*, Department of Computer Science, BINUS Graduate Program - Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia, 11480.

Although being one of the biggest coffee producer, statistically compared with the other three countries, the productivity of Indonesian coffee is considered low[2]. Coffee leaf rust contributes significantly in reducing production and is reported to be the most destructive disease in coffee plant[3]. This fact leads to a research that can contribute in detecting early symptoms of leaf disease in coffee plant.

Object recognition has accomplished significant outcomes in detecting pattern in image recognition by consolidating computer vision with an artificial intelligence techniques[4], that can be used as a tool in agriculture especially for plant leaf disease classification.

Some studies in leaf disease early detection, especially in coffee plants, were developed using Hyperspectral Image System(HIS)[5]–[7]. These technique are widely used in the last decade[8].

HIS technique obtains a hyperspectral image from the sensor system in form of camera and spectrograph, by capturing data visible across near-infrared wavelengths and supplying narrow spectral channels from the same surface area[9].

Paper review on advance hyperspectral image techniques have been published to compare the benefits and limitations of this potential method[10]. The need of fast computers, sensitive detectors, and large data storage capacity to analyze hyperspectral data made HIS costly and complex.

Different techniques in plant disease detection are carried out by utilizing deep learning CNN model[11]–[15]. The model has advantages in computational efficiency by using convolution operations, special integration, and sharing parameters which made it possible to operate on any mobile devices[16]. Review paper on deep learning has been published, concentrated on the size impact and variety of datasets to oversee how effective deep learning by using transfer learning methods for plant disease detection[17]-[18].

Several studies oversee the optimization technic to execute CNN efficiently on mobile device[19]–[21]. Others uses Offloading technique by running MobiRNN locally and resulting significant decrease on latency[22]. It is quite encouraging to be able to run a plant disease detection system by running CNN-based model[23].

In plantation sites with limited internet connectivity, an application that can run on mobile device to perform as a plant disease detection system, which can be executed locally without the need to go through the process of sending images to the server for identification processes, avoiding limitation on internet connections,

and to avoid processing latency the servers in Cloud[24] is highly benefited for farmers.

In this study, testing and comparison were carried out through two stages. First, the process of retraining using the python script from TensorFlow to generate a graph file as protobuf file. Then the file is reprocessed using TensorFlow Lite converter to generate a TensorFlow lite flat buffer file (.lite). The second stage is copying tflite files into mobile and run the test on the application installed on mobile.

In order to do the performance test on CNN models, this research utilizes the latest TensorFlow-lite framework, which optimize implementation of the three CNN architectures, MobileNet, MNasNet, and InceptionV3 on mobile device. Android application were built to run the deep learning application to read tflite file and make predictions.

The two main contributions of this paper are:

1. Evaluating and comparing pre-trained models on PC and mobile to get the best model conclusions and whether there is a decreased in accuracy. Also recognize the resources needed to run applications on mobile such as memory, CPU, power consumption, and the average time needed in detection process.
2. Extending a coffee leaf rust dataset by collecting images both from the internet and in collaboration with Indonesian Coffee and Cacao Plant Research Centre.

II. RELATED WORKS

Studies with the topic of plant disease classification and detection utilizing the most common CNN models such as AlexNet, GoogleNet, CIFAR-10, VGG, and ResNet, using the Plant Village dataset and focusing on one type of plant have been conducted by many researchers.

Classification research of diseases in maize plants, using fine-tuned CNN models GoogLeNet and Cifar10, each achieved accuracy of 98.9% and 98.8%[11]. The research which advantageous from deep learning visualization method to provide transparent information to get explanation and details of the classification mechanism[13] has been done with highest accuracy of 99.76% using inceptionV3 model within 5.64 hours training time, while the shortest time in the learning process is 0.85 hours with shallow learning type on squeeznet model, but with lower accuracy of 96.26%.

In research that contributed to the provision of datasets collected by downloading from the internet and searched by diseases and plant names in various sources, the images in dataset classified into 15 classes. After perfecting network parameters, recorded an overall accuracy 96.3%[14].

Research done by Mohanty, Hughes, and Salathé[15] evaluated images of plant leaves which distributed in multi-class labels, resulted overall accuracy varies from 85.53% using AlexNet to 99.34% with experiments on GoogLeNet variations. The research concluded that GoogLeNet performs better than AlexNet with the transfer learning training method.

Research by Brahimi et al[12] compared different learning methods, shallow models collaborated with their own models (SVM, Random forest) with deep CNN architectural models (GoogLeNet and AlexNet). In result, deep model (CNN) performed better with accuracy 99.19% and macro f1-score 98.52% compare to shallow model with accuracy 95.48% and macro f1-score 94.19%.

Studies on the impact of data sets and their variations on the effectiveness of deep learning to detect plant diseases, lead to many studies using similar tools in a dataset which do not reproduce the variety of expected conditions in the field. This study explains why most of the studies have succeeded in showing near-perfect accuracy such as [13], [15], and [18].

A research named CNNdroid[24] work on GPU-accelerated execution of trained deep CNN on Android, Caffe, Theano, and Torch models achieved maximum speed of 60X and energy saving 130x on mobile devices.

A study based on mobile deep learning on cassava dataset[23], focused on CNN based model, found that different input data (image or video) in varied performance play important part for design in real world applications.

Comparative studies on models such as VGG16, Inception v4, ResNet, and DensNet[26] using fine-tuning methods in plant disease detection resulting accuracy tests of 81.83%, 99.66%, 99.75%, and 98.08% respectively.

Research on plant detection using the CNN deep learning method on mobile devices and specifically for coffee plants has not been conducted. This study evaluated specific models of deep learning CNN that can be explicitly run on mobile devices to help classify plant diseases particularly in coffee plants, to optimize computing needs and save resources without reducing the accuracy and privacy of the mobile device itself.

III. RESEARCH METHODOLOGY

Methodology of this research comprised two primary stages: retraining and testing. The first stage is the retrain on pretrained CNN models and the second stage is testing on both PC and mobile. The illustration in Fig. 1 shows the order of working process in this research.

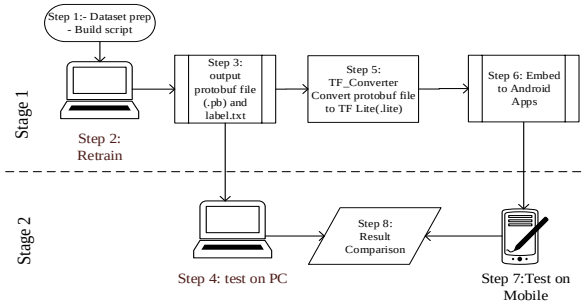

Fig. 1. Research methodology

The retraining steps begin at stage 1 with the preparation of data, collecting image data, and then implementing deep learning with the method of transfer learning. Stage 2 is the process of testing the output file from retraining.

A.Retrain Stage

In the retrain process, the system design and fabrication illustrated in flow chart in Fig. 2.

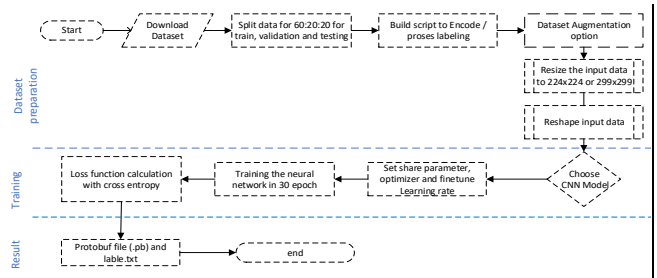

Fig. 2. Retrain stage process

The training process of CNN's state of the art at this research conducted by utilizing the TensorFlow open-source framework, TensorFlow-lite deep learning model, Python 3.6, PyCharm community edition, and Android developer. This research used a PC with the specifications as stated in Table- II.

Table- II: PC configuration and specification

No	Hardware	Specification
1	Memory	32Gb
2	CPU	Intel Core i7-8750, 2.20 GHz Gen 8
3	Graphics Processor Unit (GPU)	GeForce GTX 1050, 4GB
4	Operating System	Ubuntu 18.04 64 bits

A.1. Dataset Preparation

Engineered dataset the PlantVillage done by D. P. Hughes and M. Salathe[25], with additional two classes of coffee plant leaves were collected from the internet and in collaboration with the Indonesian Coffee and Cocoa Research Center. Data physically separated from the training composition, validation, and testing with composition 60% (32,382 Images), 20% (10,756 images), and 20% (10,765 images). Since the testing process in mobile by picking the image manually, in this research, 5% (522 images) of data is separated to be tested on PC and mobile with precisely the same data.

Table- III: PlantVillage dataset with additional two coffee leaf classes

No	Class Name	Train	Val	Test
1	Apple-blackrot	363	120	120
2	Apple-cedar-rust	161	53	53
3	Apple-healthy	956	318	318
4	Apple-scab	367	122	122
5	Blueberry-healthy	873	290	291
6	Cherry-includingsour-healthy	497	165	165
7	Cherry-includingsour-powderymildew	612	203	204
8	Coffee-healthy	116	38	38
9	Coffee-leafrust	683	227	227
10	Corn-cercospora-leafspot-grayleafspot	300	99	99
11	Corn-healthy	676	225	225
12	Corn-commonrust	693	230	231
13	Corn-northern-leafblight	573	191	191
14	Grape-blackrot	687	228	228
15	Grape-esca-blackmeasles	804	267	267
16	Grape-healthy	247	82	82
17	Grape-leafblight-isariopsis-Leafspot	627	208	208
18	Orange-haunglongbing-citrusgreening	3197	1065	1065
19	Peach-bacterialspot	1335	444	444
20	Peach-healthy	210	70	70
21	Pepper-bell-bacterialspot	580	193	193
22	Pepper-bell-healthy	859	286	285
23	Potato-earlyblight	582	193	194
24	Potato-healthy	90	29	30
25	Potato-lateblight	582	193	194
26	Raspberry-healthy	216	71	72
27	Soybean-healthy	2955	984	985
28	Squash-powdery-mildew	1078	350	348
29	Strawberry-healthy	266	88	88
30	Strawberry-leafscorch	645	214	214
31	Tomato-bacterialspot	1236	411	411
32	Tomato-earlyblight	582	193	194
33	Tomato-healthy	924	308	308
34	Tomato-lateblight	1110	369	370
35	Tomato-leafmold	554	184	184
36	Tomato-mosaic-virus	217	72	72
37	Tomato-septoria-leafspot	1029	342	343
38	Tomato-spidermites-twospotted	974	324	324
39	Tomato-targetspot	816	271	272
40	Tomato-yellowleaf-curl-virus	3110	1036	1036
	Total	**32382**	**10756**	**10765**

A.2.Transfer Learning

This research introduced a method of transfer learning from pretrained models. The concept behind transfer learning is a model trained on a large-scale and general dataset, namely Imagenet. The aim of this model is to be functioning effectively as a generic template for the visual world by taking advantage of this feature map without starting from the beginning of training models on large scale datasets. The choice of model used is specifically designed for mobile and one ordinary model. This study also fine-tuned the pre-trained models. Fine tuning is a term of transfer learning where information acquired during training is used to perform assignments or other similar domains[26]. Transfer learning consists of two phases, bottleneck creation and training. At first phase, all images on the disk will be analyzed and calculated. Then the bottle neck value of each image will be stored. “Bottleneck” is unofficial term which often used to name layers right before the last layer that does the classification. When the bottleneck creation is completed, the actual training from the top layer of the network begins. The output will show cross entropy, accuracy, and validation of accuracy.

A.3. CNN Architecture

All CNN models generally follow the same architecture, as illustrated in Fig. 3, using images as input, follow by convolutional operations, pooling operations, and several fully connected layers.

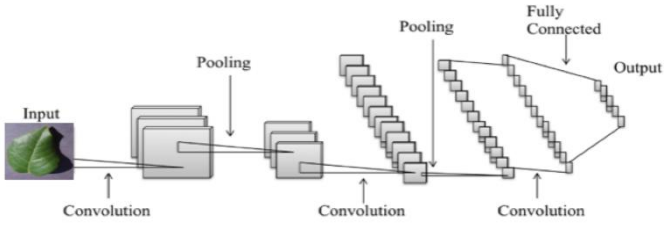

Fig. 3. CNN architecture

Three pre-trained CNN architectures were chosen on this study, a pre-trained model stored in Cloud, so that it is reusable and downloadable from the framework provider.

- **MobileNet**

The paper[27] explained that MobileNetV2 architecture contains an initial full convolution layer with 32 filters, 19 residual bottleneck layers. ReLU6 used in low-precision measurements as non-linearity due to its robustness. MobilNet used kernel size 3x3, batch normalization, and dropout.

MobileNet is an architecture created to meet the application design needs of mobile and embedded devices.

MobileNet has an efficient network and is a collection of networks consisting of two hyper-parameters to build a model with minimal and low latency. MobileNets are made with a depthwise semi-convolutional method to reduce the calculation of the first few layers.

Depthwise separable convolutional explained in paper[28] as a form of factored convolution, using factorizes standard convolution into deep convolution and with stride 1×1 convolution namely pointwise convolution, to collaborate output with deep convolutional.

This method will reduce the calculation and size of the model drastically. The pretrained mobilenet_v2_1.4 model was used in this study, with a top 5 accuracy of 92%.

- **MNasNet**

Described in the paper[29], MNAS (Mobile Neural Architecture Automated Search) with low latency as the main goal so the search can identify a model that achieves a good exchange between accuracy and latency.

MNasNet is designed for mobile devices that are resource efficient with an approach, incorporates latency model by using the factorized hierarchical search space method where the network layer is classified into several predefined frameworks, called block.

Each block contains several repetitive identical layer variables, which has stride 2 and stride 1. Stride 2 at the first layer only if the input/output resolution is different and stride 1 for other layers. MNasNet pretrained produced top-5 accuracy of 92.55%, with latency of 389 ms.

- **InceptionV3**

The first concept introduced by Szegedy et al. in 2015[30], proposed InceptionV3 architecture with an update to the inception module to increases the accuracy of the ImageNet classification in the GoogLeNet architecture. GoogleNet's next version linked to Inception vN with N stand for version number. In the paper outline[31], InceptionV3 added factors in the third iteration. Every module's output size is the next input size. InceptionV3 used variants of reduction techniques to minimize grid sizes between the Inception blocks where applicable. This architecture collaborates design activation with residual connection to accelerate initial network training. Inception modules consist of layered integration and convolution layers multiple of sizes such as 1x1, 3x3, and 5x5.

The aspect that stands out from the Inception module is the use of bottleneck layer, which is convolution 1x1. Bottleneck layers reduce calculation requirements. Additionally, a merging layer is used in modules to reduce dimensions.

B. Testing Stage

The testing stage on mobile device and PC illustrated in Fig. 4.

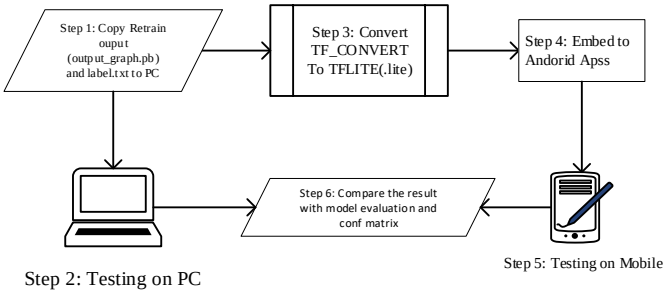

Fig. 4. Testing stage process

After completing retraining the model which generate a graph file defined as google protobuf file (.pb) and output text label (.txt). The file will be tested on PC during the training. For mobile testing, the graph file will be converted to TFLite file and copied into mobile device.

The conversion process to optimize files running on mobile devices, files generated from the training process, for example, "retrained_graph.pb" will be converted to "retrained_graph.lite" using the "TensorFlow Lite Optimizing Converter" program or tflite_convert which is a new graphics converter now included with the TensorFlow installation.

TensorFlow uses the Buffer Protocol while TFLite uses FlatBuffers. The main benefit of FlatBuffers is that it can be mapped in memory and used directly from disk without being loaded and parsed. So that startup time is much faster and gives the operating system the choice to load and unload pages needed from the model file to avoid application shutdown when it runs out of memory. The converted file will then be copied to the assets folder on mobile.

For the evaluation purposes on mobile, a mobile application was built by modifying the application sample from Tensorflow, as illustrated in Fig. 5.

Fig. 5. Mobile application testing

To determine resources consumption, this study used an Android profiler, a build-in tool from Android Studio, to assess the memory and CPU load requirements during peak time on the prediction process. While for the battery resource requirements measured using the AccuBatery application. The evaluation process will use mobile devices with specifications shown in Table- IV.

Table- IV: Mobile device spec

No	Hardware	Specification
1	Memory	4 GB
2	Processor (CPU)	Quad-core Snapdragon 820
4	Operating system	Android 8.0 Marshmallow
5	Rear Camera	12MP

IV. EXPERIMENTS

A. Experimental Design

The experiment in this study is divided into two stages, retraining and testing.

In retraining stage, the evaluation and comparison of accuracy and loss time consumed of pretrained architecture is counted during the training. The output file generated is graph file (.pb) which will be converted to tflite file (.lite) for the purpose of mobile evaluation.

The second stage is running the evaluation of testing data on PC and testing performance on mobile. The result is measured on both devices for resource requirements such as peak memory size, CPU load at peak, and battery usage.

The result is used to set the minimum requirements for mobile hardware specification and which model is the most effective to be used in mobile devices.

The set-up environment of this study is a PC or workstation equipped with tools to run deep learning, python 3.6, TensorFlow, OpenCV, Keras, Pandas, NumPy, and seaborn modules. Meanwhile for mobile device, a mobile application was built using Android developer. The application offers two choices to determine the image source, whether to take pictures using the camera or to get pictures from image directory. Output generated time consumption per image, confidence percentage prediction, and predicted label.A comparison test between PC and mobile is performed using the same separated image. This experiment aims to assess accuracy, latency, and mobile resource consumption.

Testing on PC performed by running image label_image.py script from TensorFlow over images and manually checked on mobile using built plant disease application. The result will be sent to online matrix calculator[32]. Until beginning of the test, the mobile device battery will be fully charged and restarted to get the baseline and to ensure no other application are running.

B. Experimental Result

B.1. Retraining on PC

The results of retraining stage are summarized in Table- V. Run in 30 epochs, by equating all hyperparameters, using learning rate 0.01 for MobileNet and MNasNet and learning rate 0.1 for InceptionV3, resulting final training and validation.

Table- V: Retraining result accuracy, loss, output, and time

Model and Configuration			Accuracy (%)			Loss		Output File (MB)		Time/epoch (Minutes)
			Train	Valid	Test	Train	Valid	.pb	.lite	
MobileNet	0.01	60:20:20	95.41%	94.99%	95.10%	0.1179	0.1855	24.10	23.96	3.2
MNasNet	0.01	60:20:20	97.55%	95.78%	97.00%	0.1067	0.1205	24.64	24.27	3.7
InceptionV3	0.1	60:20:20	96.20%	96.17%	94.80%	0.0257	0.0208	85.69	85.39	8.6

The final test and categorical cross entropy calculation in this study showed MobileNet leading over two other models with slightly different points.

At the final evaluation, MNasNet recorded 97% highest overall accuracy followed by 95.10% of MobileNet, and 94.80% of InceptionV3. While loss on validation recorded InceptionV3, MNasNet, and MobileNet respectively from the lowest to the highest. From output file MobileNet generated 24.10 MB, MNasNet 24.64 MB, and InceptionV3 85.69MB.

The training time difference between the two models specifically designed for mobile device (MobileNet and MNasNet) and InceptionV3 model is quite significant which InceptionV3 took 3x longer than the other two. Performance of each model in training process on dataset training, validation, and testing graphically illustrated by grouped on Fig. 6 for accuracy and Fig. 7 on cross entropy performance.

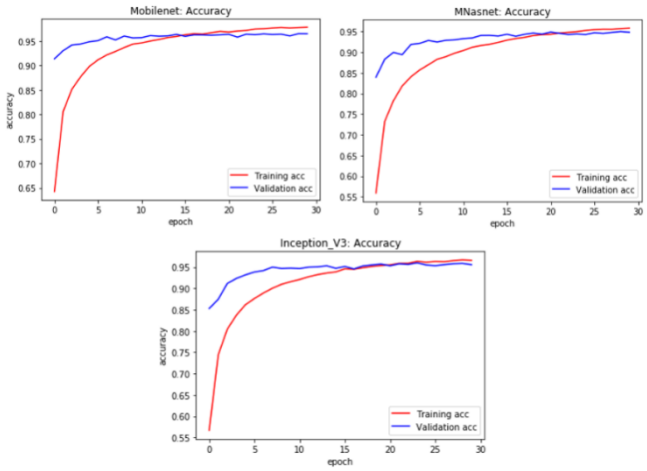

Fig. 6. Accuracy performance on each model

In Fig. 6 accuracy of MNasNet and InceptionV3 indicates better performance than MobileNet. MNasnet and InceptionV3 more consistent between training and validation. Meanwhile in Fig. 7, loss value InceptionV3 is the lowest compared to MNasNet and MobileNet. Both graphic images show that models performed no overfitting indicated.

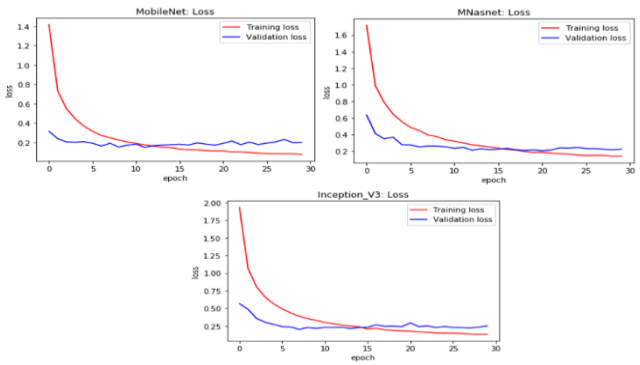

Fig. 7. Categorical-cross entropy loss on each model

The second experiment performed after the retraining phase is the testing of models by separating 20 percent of the data. This experiment aims to determine the level of accuracy per plant class, presented classification metrics to display the reliability, recall, and F1-score. The result presented in Table- VI.

Table- VI: Classification performance summary

Models	Weighted Average		
	Precision	Recall	F1-score
MobileNet	0.95	0.94	0.95
MNasNet	0.97	0.96	0.96
InceptionV3	0.96	0.96	0.96

B.2. Test Comparison PC vs Mobile

▪ **Accuracy and Latency on Mobile vs PC**

An experiment on performance comparison between PC and mobile using 522 images indicated in Table- VII with overall accuracy and average latency.

Table- VII: Accuracy and latency by models on PC and mobile

Models	Accuracy (%)		Average Latency (ms)	
	PC	Mobile	PC	Mobile
MobileNet	95.32	92.83	220.90	394.70
MNasNet	97.32	94.87	348.30	430.20
InceptionV3	96.93	95.79	376.83	2236.10

Figures in Table- VII showed assessment on mobile recorded some loss of accuracy while MobileNet and MNasNet had a slightly higher latency level. InceptionV3 encountered the lowest loss of 1.14% with MobileNet 2.45%, and MNasNet 2.5%. Meanwhile, InceptionV3 showed significantly different latency levels between mobile and PC.

▪ **Confusion Matrix on PC**

Performance of classification output for each class and result test model on PC illustrated with confusion matrix in Fig. 8 – Fig. 10. Y axis represent true labels and Y axis predicted labels.

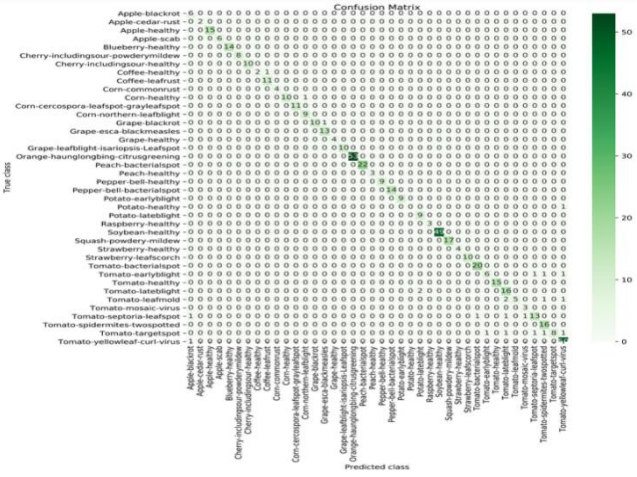

Fig. 8. MobileNet confusion matrix on PC

Fig. 8 shows good performance on most of the class, while two classes, coffee healthy and tomato, showed slightly poor in overall performance.

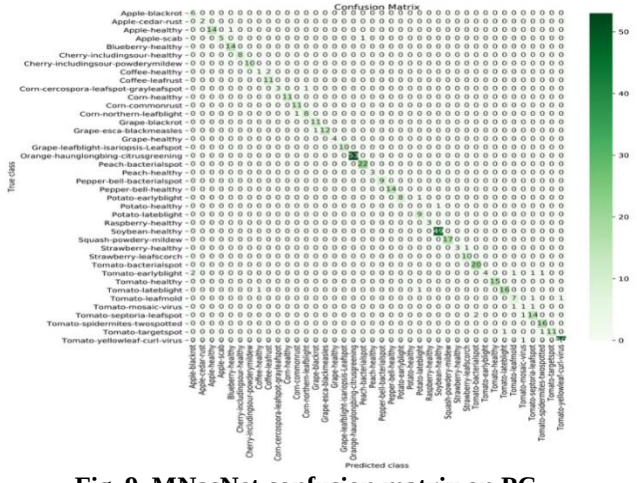

Fig. 9. MNasNet confusion matrix on PC

Fig. 9 shows that class tomato get better performance on MNasNet, while class coffee healthy shows slightly decreased performance.

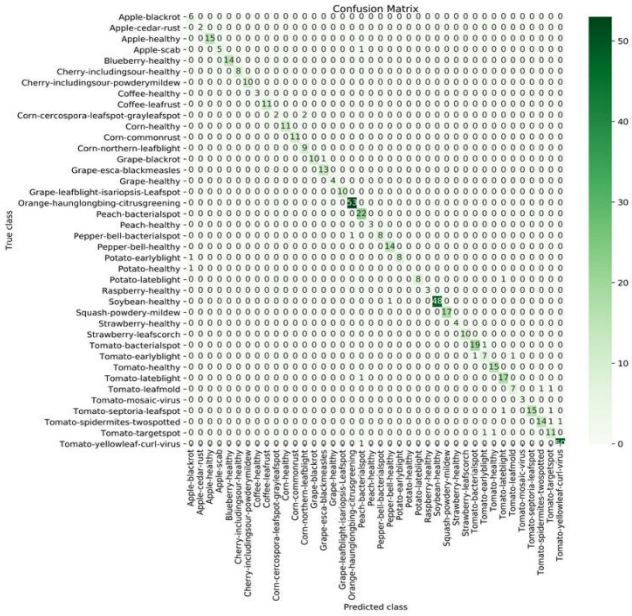

Fig. 10. InceptionV3 confusion matrix on PC

From Fig. 8 – Fig. 10 shows that the classification performance of each class has a similar pattern, tomato classes recorded poor performance results while the other classes recorded good performance.

▪ **Classification Performance Comparison**

Experiment to test and compare the output file generated from retraining process been done with result on Table- VIII to Table- X.

Table- VIII: Comparison performance each class on MobileNet

No	Class	Mobile			PC			
		Precision	Recall	F1 Score	Precision	Recall	F1	Support
1	Apple-blackrot	1.0	1.0	1.0	1.0	1.0	1.0	6
2	Apple-cedar-rust	1.0	1.0	1.0	1.0	1.0	1.0	2
3	Apple-healthy	0.94	1.0	1.0	1.0	1.0	1.0	15
4	Apple-scab	0.9	1.0	0.9	1.0	1.0	1.0	6
5	Blueberry-healthy	1.0	1.0	1.0	1.0	1.0	1.0	14
6	Cherry-includingsour-healthy	1.0	1.0	1.0	1.0	1.0	1.0	8
7	Cherry-includingsour-powderymildew	1.0	1.0	1.0	1.0	1.0	1.0	10
8	Coffee-healthy	1.0	1.0	1.0	1.0	1.0	1.0	3
9	Coffee-leafrust	1.0	1.0	1.0	1.0	1.0	1.0	11
10	Corn-cercospora-leafspot-grayleafspot	1.0	0.75	0.86	0.75	0.75	0.75	4
11	Corn-commonrust	0.9	1.0	1.0	1.0	1.0	1.0	11
12	Corn-healthy	1.00	1.0	1.0	1.0	1.0	1.0	11
13	Corn-northern-leafblight	1.00	1.0	1.00	0.89	0.89	0.89	9
14	Grape-blackrot	0.9	1.0	0.96	0.91	0.91	0.91	11
15	Grape-esca-blackmeasles	1.0	0.92	0.96	0.92	0.92	0.92	13
16	Grape-healthy	1.0	1.0	1.0	1.0	1.0	1.0	4
17	Grape-leafblight-isariopsis-Leafspot	1.0	0.70	0.82	1.0	1.0	1.0	10
18	Orange-haunglongbing-citrusgreening	1.0	1.0	1.00	1.0	1.0	1.0	53
19	Peach-bacterialspot	1.0	0.91	0.95	1.0	1.0	1.0	22
20	Peach-healthy	0.5	1.0	0.67	1.0	1.0	1.0	3
21	Pepper-bell-bacterialspot	0.78	0.78	0.78	0.69	1.0	0.82	9
22	Pepper-bell-healthy	0.93	0.93	0.93	1.0	1.0	1.0	14
23	Potato-earlyblight	0.75	0.67	0.71	0.82	1.0	1.0	9
24	Potato-healthy	1.0	1.00	1.00	1.0	1.0	1.0	1
25	Potato-lateblight	0.67	0.89	0.76	0.90	1.0	0.95	9
26	Raspberry-healthy	1.0	1.00	1.0	1.0	1.0	1.0	3
27	Soybean-healthy	1.0	1.00	1.0	1.0	0.96	0.98	49
28	Squash-powdery-mildew	1.0	1.00	1.0	1.0	1.0	1.0	17
29	Strawberry-healthy	1.0	1.00	1.0	1.0	1.0	1.0	4
30	Strawberry-leafscorch	0.88	0.70	0.78	1.0	0.80	0.89	10
31	Tomato-bacterialspot	0.95	0.90	0.92	1.0	1.0	1.0	20
32	Tomato-earlyblight	0.80	0.44	0.57	0.86	0.67	0.75	9
33	Tomato-healthy	0.94	1.00	0.97	1.0	0.87	0.93	15
34	Tomato-lateblight	0.89	0.89	0.89	1.0	0.89	0.84	18
35	Tomato-leafmold	0.75	1.00	0.86	0.8	0.89	0.84	9
36	Tomato-mosaic-virus	1.0	0.67	0.80	1.0	1.0	1.0	3
37	Tomato-septoria-leafspot	0.79	0.88	0.83	0.84	0.94	0.90	17
38	Tomato-spidermites-twospotted	0.57	1.00	0.7	0.93	0.88	0.90	16
39	Tomato-targetspot	0.50	0.077	0.13	0.80	0.92	0.86	13
40	Tomato-yellowleaf-curl-virus	1.0	0.98	0.99	1.00	0.98	0.99	51

From the Table- VIII, some class experienced accuracy degradation on mobile such apple healthy and tomato class in overall, and very poor performance on tomato-targetspot while coffee, corn, grape and soybean experienced better performance on mobile.

Table- IX: Comparison performance each class on MNasNet

No	Class	Mobile			PC			
		Precision	Recall	F1 Score	Precision	Recall	F1	Support
1	Apple-blackrot	1.0	0.83	0.91	1.0	1.0	1.0	6
2	Apple-cedar-rust	1.0	1.0	1.0	1.0	1.0	1.0	2
3	Apple-healthy	1.0	1.0	1.0	1.0	1.0	1.0	15
4	Apple-scab	1.0	1.0	1.0	1.0	1.0	1.0	6
5	Blueberry-healthy	1.0	1.0	1.0	1.0	1.0	1.0	14
6	Cherry-includingsour-healthy	1.0	0.88	0.93	1.0	1.0	1.0	8
7	Cherry-includingsour-powderymildew	1.0	1.0	1.0	1.0	1.0	1.0	10
8	Coffee-healthy	1.0	1.0	1.0	1.0	1.0	1.0	3
9	Coffee-leafrust	1.0	1.0	1.0	1.0	1.0	1.0	11
10	Corn-cercospora-leafspot-grayleafspot	1.0	0.75	0.86	1.00	0.75	0.86	4
11	Corn-commonrust	1.0	1.0	1.0	1.0	1.0	1.0	11
12	Corn-healthy	1.0	1.0	1.0	1.0	1.0	1.0	11
13	Corn-northern-leafblight	0.90	1.0	0.95	0.90	1.00	0.95	9
14	Grape-blackrot	0.91	0.91	0.91	0.92	1.00	0.96	11
15	Grape-esca-blackmeasles	0.92	0.92	0.92	1.00	0.92	0.96	13
16	Grape-healthy	0.75	1.0	0.9	1.0	1.0	1.0	4
17	Grape-leafblight-isariopsis-Leafspot	1.0	1.0	1.0	1.0	1.0	1.0	10
18	Orange-haunglongbing-citrusgreening	1.0	1.0	1.0	1.0	1.0	1.0	53
19	Peach-bacterialspot	1.0	1.0	1.0	1.0	1.0	1.0	22
20	Peach-healthy	0.75	1.0	0.86	1.0	1.0	1.0	3
21	Pepper-bell-bacterialspot	1.00	0.78	0.88	0.82	1.0	0.90	9
22	Pepper-bell-healthy	0.88	1.00	0.93	1.0	1.0	1.0	14
23	Potato-earlyblight	1.0	1.0	1.0	1.00	0.89	0.94	9
24	Potato-healthy	1.0	1.0	1.0	1.0	1.0	1.0	1
25	Potato-lateblight	1.0	1.0	1.0	1.0	1.0	1.0	9
26	Raspberry-healthy	1.0	1.0	1.0	1.0	1.0	1.0	3
27	Soybean-healthy	1.0	1.0	1.0	1.0	0.98	0.99	49
28	Squash-powdery-mildew	0.94	1.0	0.97	1.0	1.0	1.0	17
29	Strawberry-healthy	1.0	1.0	1.0	1.0	1.0	1.0	4
30	Strawberry-leafscorch	1.0	1.0	1.0	0.91	1.0	0.95	10
31	Tomato-bacterialspot	1.0	0.75	0.86	1.0	0.95	0.97	20
32	Tomato-earlyblight	1.0	0.56	0.71	0.73	0.89	0.80	9
33	Tomato-healthy	0.94	1.0	0.97	1.0	1.0	1.0	15
34	Tomato-lateblight	1.0	0.89	0.94	1.0	0.89	0.94	18
35	Tomato-leafmold	0.58	0.78	0.67	0.89	0.89	0.89	9
36	Tomato-mosaic-virus	1.0	0.67	0.80	1.0	1.0	1.0	3
37	Tomato-septoria-leafspot	0.71	1.00	0.83	0.82	0.82	0.82	17
38	Tomato-spidermites-twospotted	0.75	0.94	0.83	0.89	1.00	0.94	16
39	Tomato-targetspot	0.91	0.770	0.83	1.00	0.92	0.96	13
40	Tomato-yellowleaf-curl-virus	1.0	0.96	0.98	1.00	0.98	0.99	51

Table- IX show that the result on MNasNet evenly distributed slight degradation on every class compared to MobileNet which had one class very poor performance. Hence in overall MNasNet performance is better than MobileNet.

Table X: Comparison performance each class on InceptionV3

No	Class	Mobile			PC			
		Precision	Recall	F1	Precision	Recall	F1	Support
1	Apple-blackrot	1.0	1.0	1.0	1.0	1.0	1.0	6
2	Apple-cedar-rust	1.0	1.0	1.0	1.0	1.0	1.0	2
3	Apple-healthy	1.0	1.0	1.0	1.0	1.0	1.0	15
4	Apple-scab	1.0	1.0	1.0	1.0	1.0	1.0	6
5	Blueberry-healthy	1.0	1.0	1.0	1.0	1.0	1.0	14
6	Cherry-includingsour-healthy	1.0	1.0	1.0	1.0	1.0	1.0	8
7	Cherry-includingsour-powderymildew	1.0	1.0	1.0	1.0	1.0	1.0	10
8	Coffee-healthy	1.0	1.0	1.0	1.0	1.0	1.0	3
9	Coffee-leafrust	1.0	1.0	1.0	1.0	1.0	1.0	11
10	Corn-cercospora-leafspot-grayleafspot	1.0	0.50	0.67	1.0	0.5	0.67	4
11	Corn-commonrust	1.0	1.0	1.0	1.0	1.0	1.0	11
12	Corn-healthy	0.82	1.0	0.9	1.0	1.0	1.0	11
13	Corn-northern-leafblight	0.85	1.0	0.92	0.82	1.0	0.90	9
14	Grape-blackrot	1.0	0.85	0.92	0.85	1.00	0.92	11
15	Grape-esca-blackmeasles	1.0	1.0	1.0	1.0	0.85	0.9	13
16	Grape-healthy	1.0	1.0	1.0	1.0	1.0	1.0	4
17	Grape-leafblight-isariopsis-Leafspot	1.0	1.0	1.0	1.0	1.0	1.0	10
18	Orange-haunglongbing-citrusgreening	1.0	1.0	1.0	1.0	1.0	1.0	53
19	Peach-bacterialspot	1.0	1.0	1.0	1.0	1.0	1.0	22
20	Peach-healthy	1.0	1.0	1.0	1.0	1.0	1.0	3
21	Pepper-bell-bacterialspot	0.90	1.0	0.95	0.82	1.0	0.90	9
22	Pepper-bell-healthy	1.0	1.0	1.0	1.0	1.0	1.0	14
23	Potato-earlyblight	1.0	0.89	0.94	1.0	1.0	1.0	9
24	Potato-healthy	1.0	1.0	1.0	1.0	1.0	1.0	1
25	Potato-lateblight	1.0	1.0	1.0	1.0	1.0	1.0	9
26	Raspberry-healthy	1.0	1.0	1.0	1.0	1.0	1.0	3
27	Soybean-healthy	1.0	1.0	1.0	1.0	1.0	1.0	49
28	Squash-powdery-mildew	1.0	1.0	1.0	1.0	1.0	1.0	17
29	Strawberry-healthy	1.0	1.0	1.0	1.0	1.0	1.0	4
30	Strawberry-leafscorch	1.0	1.0	1.0	1.0	1.0	1.0	10
31	Tomato-bacterialspot	0.94	0.80	0.96	0.95	0.90	0.92	20
32	Tomato-earlyblight	0.75	0.67	0.71	0.78	0.78	0.78	9
33	Tomato-healthy	1.0	0.87	0.93	0.93	0.93	0.93	15
34	Tomato-lateblight	1.00	0.89	0.94	1.00	0.89	0.94	18
35	Tomato-leafmold	0.82	1.00	0.90	1.0	1.0	1.0	9
36	Tomato-mosaic-virus	1.0	1.0	1.0	1.0	1.0	1.0	3
37	Tomato-septoria-leafspot	0.70	1.0	0.82	0.75	0.88	0.81	17
38	Tomato-spidermites-twospotted	1.00	0.81	0.90	1.00	0.94	0.97	16
39	Tomato-targetspot	0.73	0.85	0.79	0.92	0.85	0.88	13
40	Tomato-yellowleaf-curl-virus	1.0	1.0	1.0	1.0	1.0	1.0	51

Table- X, shows that InceptionV3 performance both mobile and PC experienced same performance in each class, it made InceptionV3 considered the most robust model but with trade-off in resource and latency.

Comparison performance score for each class and model summarized on Table- XI.

Table- XI: Classification performance comparison summary on PC and mobile by average

Model	PC			Mobile		
	Avg Precision	Avg Recall	Avg F1-score	Avg Precision	Avg Recall	Avg F1-score
MobileNet	0.95	0.96	0.95	0.92	0.91	0.90
MNasNet	0.97	0.97	0.97	0.95	0.93	0.94
InceptionV3	0.97	0.96	0.96	0.96	0.95	0.96

Above figures shows average precision, recall, and F1-score on PC higher than mobile, which mean there is slightly degradation on performance in mobile.

- **Mobile Device Resource Consumption**

The experiment to determine resource consumption needed can be seen in Table- XII.

Table- XII: Resource consumption

Model	Resource Consumption		
	Memory (peak)	CPU Load (peak) %	Battery
MobileNet	134	25.1	14.2
MNasNet	132	23.2	16.6
InceptionV3	331	43.5	34.3

The figure above shows MobileNet and MNasNet require fewer resources, while InceptionV3 uses nearly half of the other two models. An experiment to detect resource consumption on Android profiler-captured mobile devices, capture CPU load on green at the top, memory usage on blue in the middle, and energy usage graphs at the bottom, shown in figures below using a PC and the Mobile is in Fig. 11 – Fig. 13 and summarized in Table- XII above.

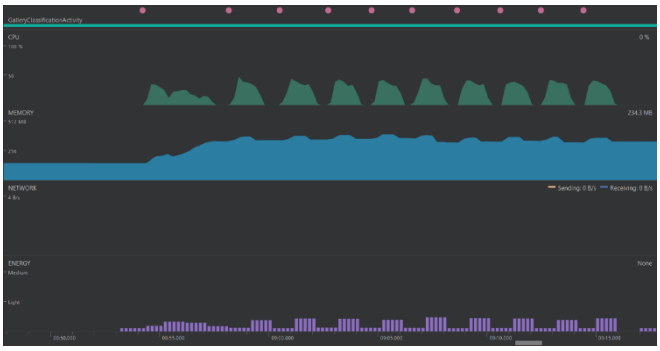

Fig. 11. Inception v3 resource profile

On Fig. 11, illustrated CPU load per image consumed highest CPU load and memory usage on InceptionV3 but with low energy consumed.

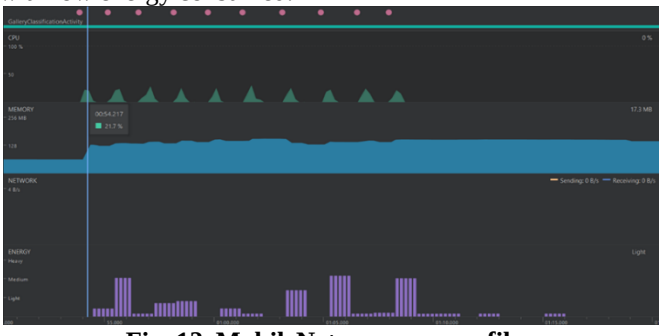

Fig. 12. MobileNet resource profile

Fig. 12 illustrated CPU load per image consumed lower CPU load and memory usage on MobilNet compare to InceptionV3. While energy consume in heavy state.

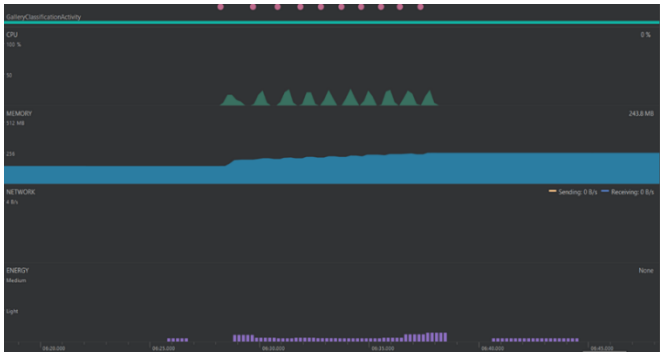

Fig. 13. MNasNet resource profile

Fig. 13 illustrated CPU load per image consumed similar to MobilNet on CPU load and memory usage on MNasNet. While energy consume is low.

Battery usage on mobile grouped on Fig. 14, which top left is MobileNet, top right is MNasNet, and at the bottom is InceptionV3.

Fig. 14. Battery consumption on Mobilenet, MNasNet, and InceptionV3

Evaluated by conducting experiments on 100 images times, recorded the consumption of battery 34.4 mAH for InceptionV3, 14.2 mAH for MobileNet, and 16.6 mAH for MNasNet. Battery usage is below 1% but in addition to the request itself, taking pictures will be added to the amount.

V. CONCLUSION

This study evaluated pretrained model with transfer learning method that can possibly retraining process much lighter at computational cost, so retraining can be performed on medium specification workstations with reasonable time consumed.

Time consume to run retraining process with bottleneck creation process is only on the first run, conclude InceptionV3 processing time required the longest time compared to MobileNet and MNasNet, both of which require nearly 0.25 times of InceptionV3.

The output file generated from this retraining process showed InceptionV3 is four times bigger than comparing the other 2 models.

The experiments result showed a slight decrease in overall accuracy in certain classes. InceptionV3 build a more reliable network, with the lowest accuracy loss, with trade-off resource consumption and latency far lower than the other two networks. Although two other architectures display very high performance with a latency level that is not too different when tested on a PC or Mobile device <500ms, lacking accurate degradation between PCs and mobile devices.

VI. FUTURE WORK

The accuracy of each model experienced minor degradation from experimental results, whereas InceptionV3 demands higher resources and latency than others. The challenge for future work is how to create a model with the same stability, but less resource requirements and less latency.

REFERENCES

1. KOMINFO, “Rayakan Hari Kopi, Kemenperin Terus Tingkatkan Ekspor Kopi Nasional,” 2017. [Online]. Available: https://kominfo.go.id/content/detail/10775/rayakan-hari-kopi-kemenperin-terus-tingkatkan-ekspor-kopi-nasional/0/artikel_gpr. [Accessed: 15-Dec-2018].
2. M. Mahfud, N. Siti, Ismiyati, and Ardiansyah, “Kajian penerapan teknologi produksi pada usahatani kopi robusta di lokasi prima tani kabupaten pasuruan,” J. Pengkaj. dan Pengemb. Teknol. Pertan., vol. 13, no. 2, 2010, pp. 141–147.
3. G. N. Agrios, Plat Pathology, Third. Academic Press, 1988.
4. D. I. Patrício and R. Rieder, “Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review,” Comput. Electron. Agric., vol. 153, no. June, pp. 69–81, 2018.
5. W. Castro, J. Oblitas, J. Maicelo, and H. Avila-George, “Evaluation of Expert Systems Techniques for Classifying Different Stages of Coffee Rust Infection in Hyperspectral Images,” Int. J. Comput. Intell. Syst., vol. 11, no. 1, p. 86, 2018.
6. A. Chemura, O. Mutanga, and T. Dube, “Separability of coffee leaf rust infection levels with machine learning methods at Sentinel-2 MSI spectral resolutions,” Precis. Agric., vol. 18, no. 5, pp. 859–881, 2016.
7. A. Chemura, O. Mutanga, M. Sibanda, and P. Chidoko, “Machine learning prediction of coffee rust severity on leaves using spectroradiometer data,” Trop. Plant Pathol., vol. 43, no. 2, pp. 117–127, 2018.
8. K. Golhani, S. K. Balasundram, G. Vadamalai, and B. Pradhan, “A review of neural networks in plant disease detection using hyperspectral data,” Inf. Process. Agric., vol. 5, no. 3, pp. 354–371, 2018.
9. P. Ghamisi, J. Plaza, Y. Chen, J. Li, and A. J. Plaza, “Advanced Spectral Classifiers for Hyperspectral Images: A review,” IEEE Geosci. Remote Sens. Mag., vol. 5, no. 1, pp. 8–32, 2017.
10. S. Sankaran, A. Mishra, R. Ehsani, and C. Davis, “A review of advanced techniques for detecting plant diseases,” Comput. Electron. Agric., vol. 72, no. 1, pp. 1–13, 2010.
11. 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.
12. M. Brahimi, K. Boukhalfa, and A. Moussaoui, “Deep Learning for Tomato Diseases: Classification and Symptoms Visualization,” Appl. Artif. Intell., vol. 31, no. 4, pp. 299–315, 2017.
13. M. Brahimi, M. Arsenovic, S. Laraba, S. Sladojevic, K. Boukhalfa, and A. Moussaoui, “Deep Learning for Plant Diseases: Detection and Saliency Map Visualisation,” Comput. Econ., pp. 1–21, 2018.
14. S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic, “Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification,” Comput. Intell. Neurosci., vol. 2016, p. 11, 2016.
15. S. P. Mohanty, D. P. Hughes, and M. Salathé, “Using Deep Learning for Image-Based Plant Disease Detection,” vol. 7, no. September, pp. 1–10, 2016.
16. Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” vol. 521, 2015.
17. J. Garcia and A. Barbedo, “Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification,” Comput. Electron. Agric., vol. 153, no. August, 2018, pp. 46–53.
18. K. P. Ferentinos, “Deep learning models for plant disease detection and diagnosis,” Comput. Electron. Agric., vol. 145, no. September 2017, pp. 311–318
19. K. Yanai, R. Tanno, and K. Okamoto, “Efficient Mobile Implementation of A CNN-based Object Recognition System,” pp. 362–366, 2016.

20. L. N. Huynh, Y. Lee, and R. K. Balan, "DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications," Proc. 15th Annu. Int. Conf. Mob. Syst. Appl. Serv. - MobiSys '17, pp. 82–95, 2017.
21. M. Ham et al., "NNStreamer: Stream Processing Paradigm for Neural Networks, Toward Efficient Development and Execution of On-Device AI Applications," 2019.
22. Q. Cao, N. Balasubramanian, and A. Balasubramanian, "MobiRNN: Efficient Recurrent Neural Network Execution on Mobile GPU," pp. 1–6, 2017.
23. A. Ramcharan et al., "A Mobile-Based Deep Learning Model for Cassava Disease Diagnosis," Front. Plant Sci., vol. 10, no. March, pp. 1–8, 2019.
24. S. Salar, L. Oskouei, H. Golestani, and M. Hashemi, "CNNdroid : GPU-Accelerated Execution of Trained Deep Convolutional Neural Networks on Android Comparing Mobile and Desktop GPUs," in Proceedings of the 24th ACM international conference on Multimedia, 2016, pp. 1201–1205.
25. D. P. Hughes and M. Salathe, "An open access repository of images on plant health to enable the development of mobile disease diagnostics," ArXiv, 2015.
26. E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, "A comparative study of fine-tuning deep learning models for plant disease identification," Compututer and Electronic in Agriculture, vol. 161, no. February, 2019, pp. 272–279.
27. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2018.
28. A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017.
29. M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," 2018.
30. C. Szegedy et al., "Going Deeper with Convolutions," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
31. C. Szegedy, V. Vanchouke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826.
32. M. Vanetti, "Confusion Matrix Online Calculator," Confusion Matrix Online Calculator, 2007. [Online]. Available: http://www.marcovanetti.com/pages/cfmatrix/?noc=8%0A. [Accessed: 27-Apr-2019].

AUTHORS PROFILE

Burhanudin Syamsuri received bachelor's degree from Duta Wacana Christian University, Yogyakarta, Indonesia in 2002. He is currently pursuing a master's degree program in Bina Nusantara University, Indonesia.

Gede Putra Kusuma received PhD degree in Electrical and Electronic Engineering from Nanyang Technological University (NTU), Singapore, in 2013. He is currently working as a Lecturer and Research Coordinator in Computer Science Department, Bina Nusantara University, Indonesia. Before joining Bina Nusantara University, he was working as a Research Scientist in I2R – A*STAR, Singapore. His research interests include pattern recognition, machine learning, face recognition, appearance-based object recognition, mobile learning, and gamification of learning.