International Journal on Advanced Science Engineering Information Technology

Vol.12 (2022) No. 2
ISSN: 2088-5334

Performance Analysis of Deep Learning-based Object Detectors on Raspberry Pi for Detecting Melon Leaf Abnormality

Hanif Rahmat [a], Sri Wahjuni [a,*], Hendra Rahmawan [a]

[a] *Department of Computer Science, IPB University, Bogor, 16680, Indonesia*
Corresponding author: [*]my_juni04@apps.ipb.ac.id

Abstract*— Melon requires intensive treatment with a high cost of maintenance. Digital image processing with deep learning can help handle diseases in melon plants efficiently. Deep-learning-based object detection has significantly better accuracy than the traditional one. However, the deep-learning-based approach leads to high computational and storage resources consumption. Speed and accuracy become tradeoffs to deal with its implementation on devices with limited computing capabilities like Raspberry Pi. This study comparatively analyzes deep-learning-based object detection algorithm performance implemented on a limited computing device, namely Raspberry Pi. The detected object in this study is melon leaves which are classified into two categories, namely abnormal and normal. The experiment was conducted using Faster R-CNN, Single Shot Multibox Detection (SSD), and YOLOv3. The results showed that Faster R-CNN had the highest mAP (~49 %) that ran ~2.5 seconds for an image but had the highest resource usage. Since accuracy is more important than time complexity in melon leaf detection, Faster R-CNN can be recommended as the best object detection algorithm to implement on Raspberry Pi. However, SSD is a fast algorithm with considerable accuracy for real-time detection. In addition, it had not only fast computational time, but SSD MobileNetV2 also spent the least resource usage. Although YOLOv3 had a significantly better running time (0.5 s) which made YOLOv3 the fastest algorithm, it had too low mAP (below 20%). Therefore, YOLOv3 is not recommended for melon leaf abnormality detection since it can allow more detection errors to occur.

***Keywords*— Faster R-CNN; melon; object detection; Raspberry Pi; SSD; YOLOv3.**

Manuscript received 25 Nov. 2020; revised 12 Jun. 2021; accepted 7 Sep. 2021. Date of publication 30 Apr. 2022.

I. INTRODUCTION

Melon (*Cucumis melo L.*) is one of the most important fruit crops in many countries over the world, like Indonesia [1], China [2], Turkey [3], and Brazil [4]. Melon is a widely well-known plant whose fruit has a great vogue by people worldwide. The melon plant can be found worldwide, especially in subtropical and tropical areas. Melon fruit is rich in fiber, minerals, beta-carotene, and vitamin C, which are beneficial to health and favorable for diet [1]. Furthermore, melon is considered one of the potential horticultural plants for extensive economic benefits [1]–[4].

However, melon is a plant species that is very sensitive to pests and diseases [5]. Pests and diseases damage parts of a melon plant, such as its leaf. Their occurrence leads to a decrease in quantity and the quality of melon fruit production. In the worst case, they can kill the plant itself [1]. Moreover, catastrophic pests and diseases in a melon can be responsible for serious economic losses [2].

Precise identification of plant diseases at the first appearance is a very important effort to manage the diseases efficiently [6]. Pests, diseases, or nutrient deficiencies can cause plant abnormalities. A disease or abnormality in a plant can be indicated by changes in leaf color or a particular pattern, such as spots indicating a disease in the plant. Since melon requires intensive treatment with a high cost of maintenance, it needs an efficient method to prevent or handle the diseases.

One of the methods to identify plant diseases is using digital image processing techniques [7]. In computer vision, deep-learning-based methods with convolutional neural networks (CNNs) have replaced traditional methods with hand-crafted features [8]. Many tasks can be done with a deep learning approach, including image classification, object detection, regression, etc. With large amounts of data availability and high computing hardware, object detection with a deep learning approach has significantly outperformed the traditional approach in terms of accuracy and precision [8].

Deep learning for object detection has recently gained big attention in agriculture [9]–[11]. Artificial intelligence, which leads to system automation, is considered a state-of-the-art

method for efficient solutions to agricultural problems [12]. However, research about object detection with deep learning, especially on agriculture problems, most are still at the model development and evaluation stage, which used high computing devices like GPU to test the model. This paper discusses the performance of deep-learning-based object detection algorithms implemented on a low-cost device, i.e., Raspberry Pi, for detecting melon leaf abnormality.

Several studies concerning object detection problems have been done, including in the fields related to the agricultural sector. Single Shot Multibox Detector (SSD) has been successfully implemented to detect diseases in cassava through leaf images [9]. A study by Arsenovic *et al.* [10] focusing on plant disease detection stated that two-stage detectors such as Faster Region-based CNN (Faster R-CNN) outperformed one-stage methods such as YOLOv3, SSD, and RetinaNet. Arsenovic *et al.* [10] conducted various object detection models to detect plant diseases. However, the two-stage method is mostly computationally slower.

Research related to the implementation of deep-learning-based object detection on Raspberry Pi was conducted by Zhong *et al.* [11] to detect insects. The study aimed to count and recognize insects in a greenhouse using the object detector You Only Look Once (YOLO) and classifier Support Vector Machine (SVM). The recognition system was then implemented on Raspberry Pi. The experiment conducted on six insect species resulted in mean counting accuracy of 92.50% and mean classification accuracy of 90.18%. Raspberry Pi's detection and recognition process took about five minutes [11].

In spite of gaining significantly better accuracy than the traditional approach, the deep learning approach leads to significantly higher time computation and memory consumption. This becomes a challenge when the deep learning method is implemented on devices with limited computing capabilities such as Raspberry Pi, which has limitations in hardware resources like CPU and memory [13]. Therefore, this research aims to conduct a comparative analysis of the performance of deep-learning-based object detection algorithms on Raspberry Pi to detect melon leaf abnormality.

A similar idea has been proposed by He *et al.* [14] using deep learning methods to detect oilseed rape pests. Since there was no significant gap between the models' accuracy, the optimal model (SSD w/Inception) was chosen considering its computational speed for application on the Android platform [14]. However, we employed three popular and well-established algorithms, i.e., Faster R-CNN, SSD, and YOLOv3 [15]–[17]. There might be a big enough variance in the models results when the case at hand detects melon leaf abnormality because of the dataset complexity. For example, Faster R-CNN would improve accuracy since it can extract richer features, but it might run slower. In addition, the implementation of deep learning models on GPU might give different results than that on low-cost devices. Thus, it is important to implement all the models on the low-cost device, i.e., Raspberry Pi, and then analyze their performance considering the tradeoff between accuracy, computational time, and resource usage.

Raspberry Pi is a low-cost computer device suitable for implementation in robotics and the Internet of Things (IoT). Robotics and IoT, along with artificial intelligence and machine learning, support the advancement in smart farming towards the development of agriculture 5.0 [12], [18].

This research is one of the research series concerning surveillance robot development for melon plants. The robot will be assigned to monitor the condition of melon plants during the growth period. The robot's detection task relies on the image taken through a camera mounted on the robot. The image is then analyzed using the object detection method so that the robot can provide information about the leaf abnormality. The robot can follow up the information from relevant treatments like pruning abnormal leaf or sending early warning alerts. The results of this research help suggest deep-learning-based object detection algorithms that are optimal to be embedded in the surveillance robot based on a tradeoff between several aspects such as accuracy, computational speed, and resource usage.

II. Materials and Method

A. Dataset

The dataset used in this study includes images data of a group of melon leaves acquired from iSurf Lab (IoT for Smart Urban Farming Laboratory), Department of Computer Science IPB University, and Agribusiness and Technology Park (ATP) IPB, Indonesia. The images were captured using Raspberry Pi camera v1.3 (5-megapixel image resolution), which produces images with the width and height, respectively, 2,592 and 1,944 pixels. The capturing process was conducted early afternoon (from 11.00 WIB) when the sunlight was sufficient to brighten the environment. At the time of data acquisition, the age of melon plants at iSurf Lab was around 3-4 weeks after planting, while at ATP, the plants were around 7-8 weeks after planting.

B. Data Annotation

In object detection, image data annotation assigns relevant bounding boxes and specific class labels for each object in an image. Each image was annotated using labeling, a tool for image data annotation. Each detected leaf was classified as either abnormal or normal in this study.

Abnormal melon plants usually show several symptoms in the leaves that indicate an abnormality. The abnormality may be caused by pests, diseases, or nutrient deficiency. Generally, among the abnormality symptoms are color changes of the leaves which may be turning yellowish, brownish, pale green or dark green at the whole or some parts of the leaf. For example, melon leaves with downy mildew disease may turn yellow, greasy, and angular dots. Plant with K (potassium) deficiency may have a brownish color of the leaves at the marginal area [5]. In addition, torn or blotchy leaves may indicate abnormal except for white spots caused by spraying liquid such as pesticide. The leaves with such abnormality symptoms were categorized as abnormal. Otherwise, they were labeled as normal if no such abnormality indication was found.

To make it easier, detection considers the position of the leaf objects to be detected. The priority leaf objects to be detected are plants planted on growing media closest to the camera. The plants in the next line from the camera are optional or the next priority to be detected, while plants

located further away are not labeled. Because the characteristic of the objects is dense and piled up, only those visible from the front are detected. Furthermore, every detected object must be clearly identified as an either abnormal or normal leaf.

C. Training Object Detection Model

The dataset was randomly split for training and testing with proportions respectively 0.8:0.2. Thus, from 522, the training and testing dataset was 417 and 105. The object detection process used TensorFlow 1.0 as a machine learning framework. Source code for processing YOLOv3 detector is available at Wizyoung [19], while Faster R-CNN and SSD used TensorFlow Object Detection API, which is available at TensorFlow [20].

The most effective and efficient way to train robust CNN object detectors is to use transfer learning. Transfer learning allows the adaption of a pre-trained deep learning model previously trained from scratch using larger datasets. The pre-trained model is then trained using smaller datasets different from the previous datasets used to train the model from scratch [21]. Several studies have demonstrated that using the transfer learning method to train object detection models for leaf issues improved the performance of the models [22], [23].

In this study, transfer learning was adapted to train object detection models. The pre-trained models were trained on the COCO dataset for Faster R-CNN and SSD [24], while the YOLOv3 pre-trained model is available at Redmon and Farhadi [25]. The pre-trained models were then trained using the melon leaf dataset. The training process was conducted using the facilities of High-Performance Computing (HPC) Indonesian Institute of Sciences (LIPI).

In addition, training object detection models were conducted with the data augmentation technique since it could increase the amount of training data so that the models' accuracy logically rises [26]. The augmentation technique used here was a random horizontal flip.

1) Anchor/Default/Prior Boxes: Different methods defined the initial anchor boxes for each algorithm. Initial anchor boxes were set using scales and aspect ratios for training Faster R-CNN by calculating anchor width (w) and anchor height (h) as Equations 1 and 2. The size of the anchor base (width × height) was 256 × 256. In this study, a combination of 3 scales and aspect ratios (0.5, 1.0, 2.0) was used to obtain 9 initial anchor boxes.

$$w = scale \times \sqrt{aspect\ ratio} \times base\ anchor\ width \quad (1)$$

$$h = scale / \sqrt{aspect\ ratio} \times base\ anchor\ height \quad (2)$$

To make predictions in training SSD, 6 layers of feature maps were used. The experiment used the minimum scale (s_{min}) of 0.2 and the maximum scale (s_{max}) of 0.95. Five aspect ratios were defined for the default boxes defined as $a_r \in \left\{1,2,3,\frac{1}{2},\frac{1}{3}\right\}$. The initial prior boxes for YOLOv3 were obtained from k-means clustering results on the bounding boxes of the training dataset. Intersection over Union (IOU) value was used as distance function instead of Euclidean function, which is commonly used for k-means. In this study, the number of selected clusters was 9, so there were 9 prior boxes to make predictions. The 9 clusters are: (18 × 14), (32 × 21), (39 × 36), (54 × 28), (57 × 51), (84 × 42), (78 × 73), (106 × 66), (129 × 109).

2) Parameter Configuration: Some configurations of training parameters were set up as shown in Table I. However, there are no standard rules in determining the parameter values. This still makes deep learning methods inefficient since it has to experiment with various parameter values for one case. Sometimes one researcher differs from another in determining parameter values according to their respective considerations, even though it is not uncommon for researchers to explain such considerations. Likewise, the same parameter values do not always lead to the same result for several implementations with different kinds of a dataset.

TABLE I
PARAMETER CONFIGURATION

Algorithm	Backbone	Input size	Batch size	Total steps
Faster R-CNN	InceptionV2, ResNet50	Min: 600 Max: 1,024	1	10,000
SSD	InceptionV2, MobileNetV2	300 × 300	16	10,000
YOLOv3	Darknet53	416 × 416	6	~10,000

Faster R-CNN used an input size of 600 pixels for the shorter edge, while the larger edge is resized based on the scale [27]. For example, if the original image size is 800 × 1,000 pixels, the input for Faster R-CNN becomes 600 × 750 pixels. In this study, the minimum input dimension was 600 pixels, while the maximum was 1,024 pixels since the original image was too large. This means that each image will be resized so that the shorter edge is 600 pixels, as used by Ren *et al.* [27]. Meanwhile, if the larger edge exceeds 1,024, then the image will be resized so that the maximum size is 1,024. For instance, if the image size is 2,592 × 1,944 pixels, the input size for Faster R-CNN will become 1,024 × 768 pixels.

Input size for SSD in this study was the same as that used by Liu *et al.* [28], 300 x 300. In Liu *et al.* [28], the experiment of SSD was done using 300 x 300 and 500 x 500 input sizes. The larger input size, as usual, resulted in better accuracy because more information can be extracted from the image, despite leading to longer running time. However, this study used 300 × 300 pixels image size for faster running time regarding the model implementation on a limited computing device, namely Raspberry Pi. Moreover, YOLOv3 used an input size of 416 × 416 pixels as already used in [29].

Minibatch stochastic methods or simply minibatch methods are optimization algorithms that use training set samples to estimate the gradient [30]. In the object detection learning process, batch size, which commonly refers to the minibatch size, can be specified from one onwards. Usually, the selected batch size is more than one but less than the entire training data. Mostly it is the closest to the power of 2, such as 8, 16, 32, 64. Different batch sizes usually take effect on training time. A larger batch size commonly leads to the longer time needed for the training process to converge. However, it allows maintaining stability when the convergence has been reached because of a more accurate gradient estimate.

On the contrary, a small batch size generally speeds up the learning process but with a volatile convergence curve of

training loss or accuracy because of high variance in the gradient estimation. Small learning rate might be required for the training process with a small batch size to keep the learning process stable [30], [31]. For the experiment, Faster R-CNN used minibatch size $m = 1$ taking into account that the training process employed 1 GPU. Moreover, SSD and YOLOv3 models were trained using minibatch sizes 16 and 6, respectively.

The stop condition used for training the models was set up into 10,000 steps. The training loss usually showed no significant changes during the experiment after the 7,000th step. Thus, the number of 10,000 steps was considered to be enough for the stop condition threshold.

D. Model Evaluation

In general, object detection models were evaluated by calculating the precision and recall. In object detection, precision and recall are two important model evaluation metrics used to measure how well a detected object matches a reference object. Precision can be defined as a proportion between correctly detected positive classes among all detected positive classes, while recall refers to the proportion of correctly detected objects among all objects that should be detected [32]. Precision and recall are calculated using Equations 3 and 4 based on Table 2 [14].

TABLE II
CONFUSION MATRIX

	Actual positive	**Actual negative**
Predicted positive	True positive (TP)	False-positive (FP)
Predicted negative	False-negative (FN)	True negative (TN)

$$precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3}$$

$$recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4}$$

Average precision (AP) is used as a metric to measure the performance of the object detection model for a given class. An AP score is defined based on the mean of the precision scores of a set of equidistant recall values (0, 0.1, 0.2, ..., 1) [14]. AP score is calculated with Equations 5 and 6 [14].

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, 0.2, \ldots, 1\}} P_{interp}(r) \tag{5}$$

$P_{interp}(r)$ or interpolated precision is defined as

$$P_{interp} = \max_{\tilde{r}:\tilde{r} \geq r} P(\tilde{r}) \tag{6}$$

where $P(\tilde{r})$ is the precision measured at recall ($\tilde{r}$).

To evaluate multi-class detection, mean average precision (mAP) is used as the average value of all AP values [14]. In this study, the calculation of mAP was processed using GPU.

E. Implementation on Raspberry Pi

Object detection task was carried out on Raspberry Pi to analyze the capability of such device in running object detection algorithms. At this implementation, the computation time of the algorithms when performing detection inference was measured, hereinafter referred to as inference time. The inference phase means a phase when an object detector infers relevant bounding boxes and class labels of objects detected in an image. Furthermore, here resources usage was also measured. Memory consumption and CPU (Central Processing Unit) time are important parameters to measure how many resources a program or task uses. Thus, this study involves those two aspects to analyze the algorithm performance on resources usage.

For efficiency purposes, instead of using the entire test set, the detection process on Raspberry Pi was conducted using 10 images randomly selected from the test data. The computation time was measured each time the detector performed inference for each image. After that, an average computation time was calculated from the obtained inference time data. Moreover, the detection process of one test image was carried out to measure the resources usage of each detector. Parameters measured included CPU (Central Processing Unit) time and RSS (Resident Set Size). The resources usage was monitored and recorded while the detector program was running.

III. RESULTS AND DISCUSSION

Mean average precision (mAP) is an evaluation metric for the object detection task. The mAP score is obtained from averaging the AP scores of all classes. This study used an IOU threshold 0.6 for non-max suppression (NMS) process to calculate the mAP score. Based on Table 3 Faster R-CNN has the highest mAP compared with the other methods, while the lowest mAP belongs to YOLOv3. Faster R-CNN with ResNet50 as the feature extractor is the most accurate model for detecting melon leaves and their class labels with a mAP of 48.85%. This indicates that the two-stage models can obtain more accurate detection results than the one-stage models. As shown in Fig. 1, the two-stage methods gain up to two times higher mAP than the one-stage methods. For the one-stage, SSD with MobileNetV2 achieves higher mAP than that with InceptionV2 by the difference of ~ 6%. MobileNetV2 is probably better than InceptionV2 in feature extraction. The inverted Residual Block proposed by Sandler *et al.* [33] may enrich the features extracted from such a complex image.

TABLE III
AP AND MAP VALUES OF EACH METHOD

Method	**Backbone**	**AP_{normal} (%)**	**$AP_{abnormal}$ (%)**	**mAP (%)**
Faster R-CNN	InceptionV2	**47.26**	50.2	**48.73**
	ResNet50	45.83	**51.86**	48.85
SSD	InceptionV2	23.83	30.25	27.04
	MobileNetV2	30.52	35.81	33.16
YOLOv3	Darknet53	15.26	17.85	16.56

Distinguishing whether a leaf is abnormal or normal from many leaves and locating its position among them is a fairly complex problem. It is necessary to define the leaf object distinguish its class and determine which location must be detected. The background may look similar to the positive objects because there are leaf objects in the background and foreground in the image of a group of leaves. As in the data annotation process, not all leaves were labeled. In addition, a pile of leaves increases the density of the detection objects.

With such data characteristics, two-stage detectors that are powerful in the aspect of extracted features richness can result in better accuracy than one-stage detectors.

Meanwhile, YOLOv3 has the smallest mAP (16.56%) compared to the other methods. This score is two times smaller than SSD MobileNetV2. For cases that require high accuracy, such as leaf abnormality detection, YOLOv3 model is not recommended because it will result in more detection errors. It seems that YOLOv3 is not very good at extracting information (features) from complex and dense data such as leaves. This could be because the prior boxes at each scale perform poorly when retrieving information. Unlike SSD, which used six default boxes in each feature map layer, YOLOv3 used fewer (three) prior boxes at each scale to detect objects so it could not dig up more information. The background looks similar to positive objects in leaf data, which makes it complex, especially for one-stage detectors. In addition, YOLOv3 performs poorly with NMS operation on dense object areas, leading to a higher error rate when detecting dense objects [34]. YOLOv3 produced multiple detections on the same object in this experiment, as seen in the detection results. In Nguyen *et al.* [35], YOLOv3 experienced a dramatic decrease in mAP when IOU was increased from 0.5 to 0.75 because YOLOv3 did not perform well during localization.

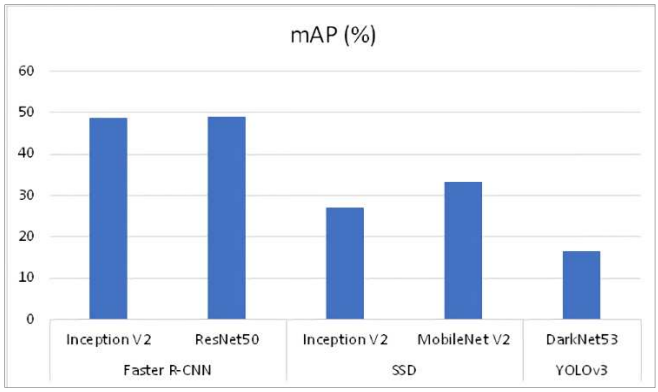

Fig. 1 Comparison of mAP values

Although the two-stage method outperforms the one-stage method in terms of accuracy, the two-stage method is mostly poor in terms of computation time. It can be seen that the computation time required by the two-stage method (Faster R-CNN) to perform detection is longer than the one-stage methods (SSD and YOLOv3). The two-stage method performs the proposal region stage and classification and regression separately, thus increasing the computation time. At each stage, the two-stage method performs classification, the first is to determine the object's existence, and the second is to assign the class label. Table 4 shows the computation time comparison for each method with Faster R-CNN ResNet50 as a benchmark.

For the one-stage method, YOLOv3 has the fastest computation time with a computation speed of 0.5 seconds, up to five times faster than the other methods (Faster R-CNN and SSD MobileNetV2) shown in Table 8. Although YOLOv3 is the fastest, the mAP value of it is still very small (below 20%). Thus, the YOLOv3 method is not recommended for detecting leaf abnormality of melon plants.

SSD with InceptionV2 takes 1.2 times faster inference than MobileNetV2 but has ~6% smaller mAP. Because in case of abnormality detection, accuracy is more important, it can be said that SSD MobileNetV2 is better than SSD InceptionV2 because of higher mAP but with considerable computation time for implementation on limited computing devices like Raspberry Pi.

Faster R-CNN requires the longest computation time. Compared to YOLOv3, Faster R-CNN performs inference five times longer but has mAP value up to three times higher. Each algorithm is stronger in one aspect and weaker in another. However, in this study, accuracy takes precedence over computation time. Faster R-CNN ResNet50 has the highest mAP value with a slightly longer computation time than SSD MobileNetV2 SSD, about 1.2 times longer. Faster R-CNN with ResNet50 and InceptionV2 has nearly the same performance with mAP of ~49% and inference time of ~2.5 seconds, but the Faster's smaller scale is considered R-CNN ResNet50 is better because it takes 84 milliseconds faster. In this case, it seems that the residual block of the ResNet50 network has a role in raising the accuracy [36].

TABLE IV
INFERENCE TIME OF EACH METHOD

Method	Backbone	Inference time (s) per image	Speed up
Faster R-CNN	InceptionV2	2.591	1×
	ResNet50	2.507	1×
SSD	InceptionV2	1.716	1.5×
	MobileNetV2	2.07	1.2×
YOLOv3	Darknet53	**0.522**	**4.8×**

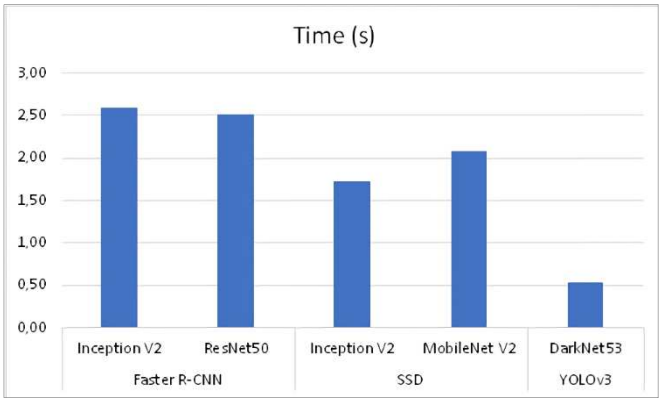

Fig. 2 Inference time comparison

In this case, it can be seen that Faster R-CNN is applicable to limited computing devices such as Raspberry Pi because it has the highest level of accuracy with considerable computation time. However, SSD is preferable for real-time detection such as on video because it ran faster than Faster R-CNN yet had considerable accuracy.

Fig. 3 shows a scatter plot of mAP against computation time of the object detection algorithms. Fig. 3 indicates that computation time is directly proportional to mAP; the higher the mAP value, the longer the computation time required. Faster R-CNN ResNet50 is the model with the best mAP, which is slightly faster than the Faster R-CNN InceptionV2. Faster R-CNN ResNet50 is highly recommended for cases like melon leaf abnormality detection if data transmission and processing, for example, a surveillance robot, is carried out in

no less than five minutes. This does not matter because signs or symptoms of abnormalities in plant leaves can still be detected, even if not in seconds. However, the SSD method is recommended if faster processing is desired, such as for a real-time video.

In summary, for cases that require high accuracy, the two-stage Faster R-CNN method can be the best alternative with considerable computation time for implementation on a limited computing device such as Raspberry Pi. As for real-time detection needs such as video, SSD can be the best alternative to use.

Fig. 4 shows the amount of memory (RAM) used by each object detection program when it was run. The x-axis represents the time during which the program was running, every second, starting from the time the program was executed until it produced object detection output. Meanwhile, the y-axis shows the number of RSS (MiB) allocated to run each detection program. RSS represents non-swap memory or RAM used by the task [37]. In general, in the early seconds, the memory usage is more dominant, indicated by the more volatile movement of RSS graphic than CPU usage, which tends to be stable. At this early stage, the program usually loads modules or packages and loads the object detection model. The process of loading packages and the model is less burdensome on CPU because there are no commands for complex computation operations. Thus, the complexity here is focused on memory.

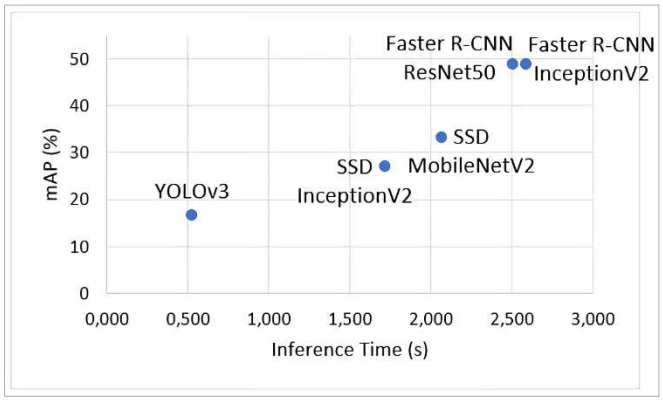

Fig. 3 Scatter plot of inference time against mAP

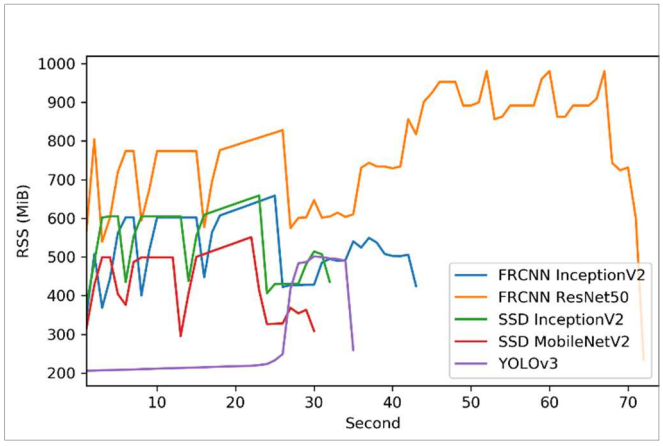

Fig. 4 Memory usage of each method

The largest memory usage is owned by Faster R-CNN ResNet50 where the maximum usage of RSS reaches 981 MiB. This is reasonable because Faster R-CNN is a two-stage model with a complex architecture (namely two stages: proposal region and classification) so that it affects more memory usage, coupled with ResNet50 as a backbone that has a deeper network than InceptionV2 [14]. In addition, the Faster R-CNN ResNet50 detection process requires more memory, as seen in Figure 2, which shows the movement of RSS graphics in the final seconds is higher.

The smallest memory usage belongs to YOLOv3 and SSD MobileNetV2, with a maximum memory usage of 502 MiB and 551 MiB, respectively. YOLOv3 being lighter could be because the design structure of the program algorithm is more efficient than Faster R-CNN and SSDs that use Object Detection API. As seen in Fig. 4, the movement of YOLOv3 RSS significantly increases after the 25th second, which is likely when the program is doing the detection process, in contrast to SSD and Faster R-CNN graphics which have similar RSS fluctuation patterns in the initial seconds (before the 25th or 30th second). In addition, it may be due to the efficient YOLOv3 model as a one-stage model even though from the backbone aspect, the size of Darknet53 is even larger than ResNet50 [38]. Moreover, the lighter SSD MobileNetV2 is possible because of the efficient SSD model as a one-stage object detector and the MobileNetV2 architecture with mobile architecture designed for devices with low computing capabilities [33].

Fig. 5 shows the percentage of CPU usage by object detection programs when run. Percentage of CPU usage means the percentage of CPU time used by the task [37]. In general, after the 25th second, the memory usage starts to decline, followed by the movement of the CPU percentage graph towards the peak. This indicates that the detection process is running. On Raspberry Pi, the image processing is conducted on the CPU, so various mathematical calculations are carried out by deep learning that is burdening the CPU.

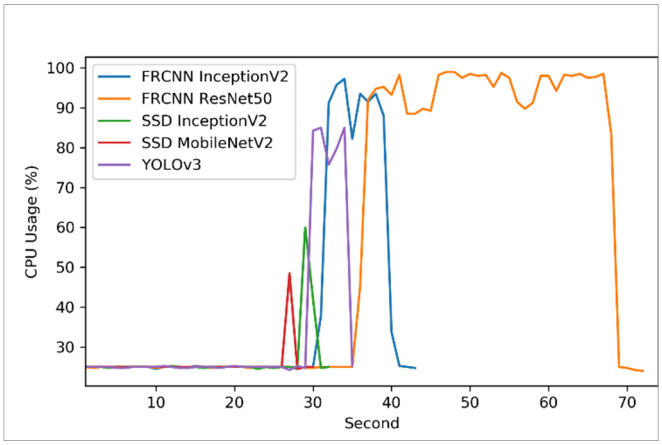

Fig. 5 CPU time of each method

Two-stage detection approach and a deep network backbone make Faster R-CNN ResNet50 have the highest CPU time with CPU usage percentage of 99% (Table 5). SSD MobileNetV2 is the algorithm that consumes the least CPU resources with a percentage of 48.5%. On the other hand, although YOLOv3 consumes the least memory resource, it has a maximum CPU time of 85%, which is greater than the SSD-based methods have. Thus, it can be concluded that from the aspect of resource usage, SSD MobileNetV2 is the best

method to be implemented on limited computing devices such as Raspberry Pi. Apart from having sufficient mAP, SSD MobileNetV2 uses resources more efficiently than the other models.

TABLE V
MAXIMUM USAGE OF CPU AND RSS

Method	Backbone	CPU time (%)	RSS (MiB)
Faster R-CNN	InceptionV2	97.25	659
	ResNet50	99	981
SSD	InceptionV2	60	659
	MobileNetV2	**48.5**	551
YOLOv3	Darknet53	85	**502**

Fig. 6(a) shows a test image with ground truth boxes, while Fig. 6(b)-(f) shows the image with predicted boxes of each method. The detection was done using IOU threshold of 0.6 for NMS operation. Many multiple boxes are assigned to the same objects in YOLOv3 prediction. In case the overlapping boxes are too many, it indicates that the model poorly performs the detection. However, the number of bounding boxes can be reduced by decreasing IOU threshold for NMS operation.

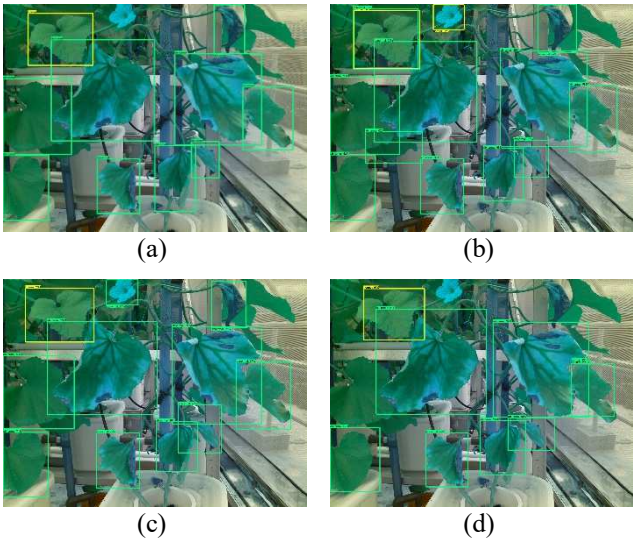

Fig. 6 Image with (a) ground truth boxes, detection results of (b) Faster R-CNN InceptionV2, (c) Faster R-CNN ResNet50, and (d) SSD InceptionV2

IV. CONCLUSIONS

The results showed that the mean average precision (mAP) value is directly proportional to the running time. That is, the algorithm with higher mAP would require a longer running time. Faster R-CNN had the highest mAP (~49 %) that ran ~2.5 seconds for an image, yet with the highest resources usage. Considering the accuracy, Faster R-CNN can be recommended as the best object detection algorithm in case accuracy is more important than time complexity, including melon leaf abnormality. However, for real-time detection such as on video, SSD can be considered a fast algorithm with considerable accuracy that can be implemented on limited computing devices such as Raspberry Pi. Although YOLOv3 had significantly better running time (0.5 s) which made YOLOv3 the fastest algorithm discussed here, it had too low mAP below 20%. Therefore, in this case, YOLOv3 is not recommended for melon leaf abnormality detection since it would lead to more detection errors.

However, future research can concentrate on the detection of more specific types of leaf abnormality in the melon plant. The data annotation process can involve experts to minimize errors when assigning bounding boxes and class labels. In addition, for a better experiment, various training techniques can be applied by increasing the number of datasets, changing the input size, trying more data augmentation techniques, or using different backbones. Adjusting the model networks can also be a good alternative for better accuracy and running time.

ACKNOWLEDGMENT

The computation in this work has been done using the facilities of HPC LIPI, Indonesian Institute of Sciences (LIPI). The dataset used in this study is acquired from the iSurf Lab (IoT for Smart Urban Farming Laboratory) Department of Computer Science FMIPA IPB and Agribusiness and Technology Park (ATP) IPB.

REFERENCES

[1] B. S. Daryono and S. D. Maryanto, *Keanekaragaman dan Potensi Sumber Daya Genetik Melon*. Yogyakarta: Gadjah Mada University Press, 2017.

[2] Q. Zhao *et al.*, "Biocontrol of Fusarium wilt disease for Cucumis melo melon using bio-organic fertilizer," *Appl. Soil Ecol.*, vol. 47, no. 1, pp. 67–75, Jan. 2011, doi: 10.1016/j.apsoil.2010.09.010.

[3] S. Dinc *et al.*, "the Rootstock Effects on Agronomic and Biochemical Quality Properties of Melon Under Water Stress," *Fresenius Environ. Bull.*, vol. 27, no. 7, pp. 5008–5021, 2018.

[4] M. J. de Santana, G. de A. Bocate, M. A. Sgobi, S. S. de Souza, and T. T. B. Valeriano, "Irrigation management of muskmelon with tensiometry," *Rev. Agrogeoambiental*, vol. 9, no. 3, 2017, doi: 10.18406/2316-1817v9n32017965.

[5] A. Balliu and G. Sallaku, "Early production of melon, watermelon and squashes in low tunnels," in *Good Agricultural Practices for greenhouse vegetable production in the South East European countries*, 230th ed., W. Baudoin, A. Nersisyan, A. Shamilov, A. Hodder, and D. Gutierrez, Eds. Rome: Food and Agriculture Organization of the United Nations, 2017, pp. 341–351.

[6] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection," *Front. Plant Sci.*, vol. 7, Sep. 2016, doi: 10.3389/fpls.2016.01419.

[7] S. D. Khirade and A. B. Patil, "Plant Disease Detection Using Image Processing," in *2015 International Conference on Computing Communication Control and Automation*, Feb. 2015, pp. 768–771, doi: 10.1109/ICCUBEA.2015.153.

[8] D. S. Trigueros, L. Meng, and M. Hartnett, "Face Recognition: From Traditional to Deep Learning Methods," Oct. 2018, Accessed: Nov. 04, 2020. [Online]. Available: http://arxiv.org/abs/1811.00116.

[9] A. Ramcharan *et al.*, "A Mobile-Based Deep Learning Model for Cassava Disease Diagnosis," *Front. Plant Sci.*, vol. 10, Mar. 2019, doi: 10.3389/fpls.2019.00272.

[10] M. Arsenovic, M. Karanovic, S. Sladojevic, A. Anderla, and D. Stefanovic, "Solving Current Limitations of Deep Learning Based Approaches for Plant Disease Detection," *Symmetry (Basel).*, vol. 11, no. 7, p. 939, Jul. 2019, doi: 10.3390/sym11070939.

[11] Y. Zhong, J. Gao, Q. Lei, and Y. Zhou, "A Vision-Based Counting and Recognition System for Flying Insects in Intelligent Agriculture," *Sensors*, vol. 18, no. 5, p. 1489, May 2018, doi: 10.3390/s18051489.

[12] L. Ahmad and F. Nabi, *Agriculture 5.0: Artificial Intelligence, IoT, and Machine Learning*. Florida: CRC Press, 2021.

[13] D. Foley and R. O'Reilly, "An evaluation of convolutional neural network models for object detection in images on low-end devices," in *CEUR Workshop Proceedings*, 2018, vol. 2259, pp. 350–361, [Online]. Available: http://ceur-ws.org/Vol-2259/aics_32.pdf.

[14] Y. He, H. Zeng, Y. Fan, S. Ji, and J. Wu, "Application of Deep Learning in Integrated Pest Management: A Real-Time System for Detection and Diagnosis of Oilseed Rape Pests," *Mob. Inf. Syst.*, vol. 2019, 2019, doi: 10.1155/2019/4570808.

[15] U. Alganci, M. Soydas, and E. Sertel, "Comparative research on deep learning approaches for airplane detection from very high-resolution satellite images," *Remote Sens.*, vol. 12, no. 3, p. 458, Feb. 2020, doi: 10.3390/rs12030458.

[16] N. D. Nguyen, T. Do, T. D. Ngo, and D. D. Le, "An Evaluation of Deep Learning Methods for Small Object Detection," *J. Electr. Comput. Eng.*, vol. 2020, 2020, doi: 10.1155/2020/3189691.

[17] M. Li, Z. Zhang, L. Lei, X. Wang, and X. Guo, "Agricultural greenhouses detection in high-resolution satellite images based on convolutional neural networks: Comparison of faster R-CNN, YOLO v3 and SSD," *Sensors (Switzerland)*, vol. 20, no. 17, pp. 1–14, 2020, doi: 10.3390/s20174938.

[18] V. Saiz-Rubio and F. Rovira-Más, "From smart farming towards agriculture 5.0: A review on crop data management," *Agronomy*, vol. 10, no. 2, p. 207, Feb. 2020, doi: 10.3390/agronomy10020207.

[19] Wizyoung, "Complete YOLOv3 TensorFlow implementation." [Online]. Available: https://github.com/wizyoung/YOLOv3_TensorFlow.

[20] TensorFlow, "Models and examples built with TensorFlow." [Online]. Available: https://github.com/tensorflow/models/tree/r1.13.0.

[21] Y. Wang, C. Wang, and H. Zhang, "Combining a single shot multibox detector with transfer learning for ship detection using sentinel-1 sar images," *Remote Sens. Lett.*, vol. 9, no. 8, pp. 780–788, Aug. 2018, doi: 10.1080/2150704X.2018.1475770.

[22] V. Ponnusamy, A. Coumaran, A. S. Shunmugam, K. Rajaram, and S. Senthilvelavan, "Smart Glass: Real-Time Leaf Disease Detection using YOLO Transfer Learning," in *Proceedings of the 2020 IEEE International Conference on Communication and Signal Processing, ICCSP 2020*, Jul. 2020, pp. 1150–1154, doi: 10.1109/ICCSP48568.2020.9182146.

[23] M. Buzzy, V. Thesma, M. Davoodi, and J. M. Velni, "Real-time plant leaf counting using deep object detection networks," *Sensors (Switzerland)*, vol. 20, no. 23, pp. 1–14, Dec. 2020, doi: 10.3390/s20236896.

[24] Github, "TensorFlow detection model zoo," *GitHub*, 2020. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tfl_detection_zoo.md (accessed Mar. 07, 2021).

[25] J. Redmon and A. Farhadi, "YOLO: Real-Time Object Detection." https://pjreddie.com/darknet/yolo/ (accessed Mar. 07, 2021).

[26] B. Zoph, E. D. Cubuk, G. Ghiasi, T.-Y. Lin, J. Shlens, and Q. V. Le, "Learning Data Augmentation Strategies for Object Detection," 2019, Accessed: May 31, 2020. [Online]. Available: http://arxiv.org/abs/1906.11172.

[27] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[28] W. Liu *et al.*, "SSD: Single shot multibox detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, Dec. 2016, doi: 10.1007/978-3-319-46448-0_2.

[29] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018, Accessed: May 27, 2020. [Online]. Available: http://arxiv.org/abs/1804.02767.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge: MIT Press, 2016.

[31] J. Brownlee, *Better Deep Learning. Train Faster, Reduce Overfitting, and Make Better Predictions*, V1.8., vol. 1, no. 2. 2018.

[32] A. Godil, R. Bostelman, W. Shackleford, T. Hong, and M. Shneier, "Performance Metrics for Evaluating Object and Human Detection and Tracking Systems," Gaithersburg, MD, Jul. 2014. doi: 10.6028/NIST.IR.7972.

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 4510–4520, Jan. 2018, doi: 10.1109/CVPR.2018.00474.

[34] J. Lei, C. Gao, J. Hu, C. Gao, and N. Sang, "Orientation Adaptive YOLOv3 for Object Detection in Remote Sensing Images," Springer, Cham, 2019, pp. 586–597.

[35] K. Nguyen, N. T. Huynh, P. C. Nguyen, K. D. Nguyen, N. D. Vo, and T. V. Nguyen, "Detecting objects from space: an evaluation of deep-learning modern approaches," *Electron.*, vol. 9, no. 4, p. 583, Mar. 2020, doi: 10.3390/electronics9040583.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Dec. 2016, vol. 2016-Decem, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[37] M. Kerrisk, "Pidstat(1) – Linux manual page," 2020. https://man7.org/linux/man-pages/man1/pidstat.1.html.

[38] M. Elgendi *et al.*, "The Performance of Deep Neural Networks in Differentiating Chest X-Rays of COVID-19 Patients From Other Bacterial and Viral Pneumonias," *Front. Med.*, vol. 7, Aug. 2020, doi: 10.3389/fmed.2020.00550.