The Smart Image Recognition Mechanism for Crop Harvesting System in Intelligent Agriculture

Gwo-Jiun Horng, *Member, IEEE*, Min-Xiang Liu, and Chao-Chun Chen, *Member, IEEE*

Abstract—**This study proposed a harvesting system based on the Internet of Things technology and smart image recognition. Farming decisions require extensive experience; with the proposed system, crop maturity can be determined through object detection by training neural network models, and mature crops can then be harvested using robotic arms. Keras was used to construct a multilayer perceptron machine learning model and to predict multiaxial robotic arm movements and position. Following the execution of object detection on images, the pixel coordinates of the central point of the target crop in the image were used as neural network input, whereas the robotic arms were regarded as the output side. A MobileNet version 2 convolutional neural network was then used as the image feature extraction model, which was combined with a single shot multibox detector model as the posterior layer to form an object detection model. The model then performed crop detection by collecting and tagging images. Empirical evidence shows that the proposed model training had a mean average precision (mAP) of 84%, which was higher than that of other models; a mAP of 89% was observed from the arm picking results.**

Index Terms—**Internet of Things (IoT), multiaxial robotic arm control, deep learning, object detection in smart agriculture**

I. Introduction

FACTORS such as competition brought on by globalization, extreme climate change, and population aging have indirectly affected crop harvest and crop profits. Coupled with aggressive changes in agriculture in recent years, fierce competition between markets has led to demands for higher product quality. Accordingly, an increasing number of farmers are being forced to adapt to new technologies to meet rising quality demands and achieve lower costs. During the 1950s–1960s, economic development was achieved through industrial development, which was in turn supported by agriculture. Now that the industrial sector has matured, it should in turn support agricultural development by applying industry 4.0 technologies such as cloud technology, big data analysis, Internet of Things (IoT), smart machinery, and sensor applications to the agricultural sector. Like the industrial sector, the agricultural sector is being forced to adopt modern methods, among which the combined use of sensors with IoT technologies is the most common method. This method enables farmers to quickly understand the conditions of vast farmlands and immediately implement appropriate measures.

Agriculture is probably one of the oldest industries. However, with the development and use of agricultural machinery, population aging in the agricultural sector has aggravated, as shown in Fig. 1; the average age of the population engaging in agriculture is 50–64 years, and the size of the population of this age group is also much greater than that of groups younger than 49 years. In addition, the increase in the proportion of the younger population joining the agricultural industry is insufficient for replacing the older generation of agricultural workers in the future. Furthermore, the population of those aged 40–49 years exhibits a diminishing trend, resulting in a larger gap between younger and older age groups. This indicates that increasingly fewer people want to participate in agriculture.

Not everyone must grow their own food. Currently, less than 2% of the US population is engaged in agriculture. However, this 2% of the population supplies an amount of food that far exceeds that required by the remaining 98% of the population. Estimates indicate that at the end of the twentieth century, the food grown by one farmer in the United States was sufficient for 25 persons; this ratio has increased to 1:130. This shows that one farmer on a modern food crop farm can produce an amount of food sufficient to feed thousands of people. With the continual advancement of agricultural machinery, farmers are playing increasingly specialized roles.

Smart agriculture is a future trend of technology. Drones, a type of agricultural robot, are conducive to successful agriculture, and their application potential is limitless. Farmers may fail to notice crops situated in the middle of a field during the initial stage of a plant disease. Drones can be helpful for such a situation by preliminarily monitoring the field, reporting and recording crop conditions, and performing disease control. This saves farmers time and energy while achieving effective plant growth control.

This work was supported by Ministry of Science and Technology (MOST) of Taiwan under Grants MOST 107-2221-E-218-024. This work was also supported by the Allied Advanced Intelligent Biomedical Research Center, STUST under Higher Education Sprout Project, Ministry of Education, Taiwan.

Gwo-Jiun Horng is with the Department of Computer Science and Information Engineering, Southern Taiwan University of Science and Technology, Tainan, Taiwan (Corresponding author e-mail: grojium@gmail.com).

Min-Xiang Liu, Chao-Chun Chen, are with Institute of Manufacturing Information and Systems, Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan (e-mail: 98081077@gs.ncku.edu.tw, chencc@imis.ncku.edu.tw)

In recent years, computer vision technology has attained major breakthroughs, enabling the categorization of objects detected from an image as well as determining the positions of these objects in the image. This technique is referred to as object detection and has been prevalently used in various applications, including pedestrian detection, stock inventory, and self-driving vehicles. Various object detection techniques have been developed. However, not all such techniques are applicable to every situation. Factors such as object attributes, application environments, and equipment factors must be considered. Therefore, this study analyzed and compared different object detection models to determine the model most suitable for this study. Robotic arms are programable and feature functions similar to those of a human arm. A robotic arm is composed of various hardware components and enables various rotation and parallel movements through the interconnection between arm levels and joints. Computation of such movements can be executed using a kinematic chain algorithm, which requires actual parameters for the hardware components of each robotic arm; hence, measurements must be conducted for all arms. Additionally, such an algorithm is difficult to understand and design. Therefore, this study adopted a neural network to predict the movement of a robotic arm and resolve problems concerning robotic arm designs.

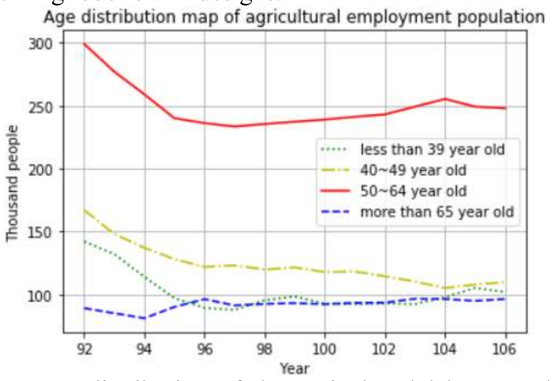


Fig. 1. Age distribution of the agricultural labor population (2003–2017)[1]

In view of the booming development and future needs of today's object detection technology, IoT technology, data analysis, and various smart technologies, this study explored the prospects, practicality, and applicability of various technologies. By conceptualizing smart image recognition and IoT as one entity, this study enabled computers to perform automated harvesting of fruits or targets through natural vision and device communication achieved with wireless communication.

In agricultural technology, most automated machinery is designed to operate in a fixed manner and requires human operation or regular supervision to avoid mechanical failures or errors. The machines are not only expensive but also feature a narrow range of applications and use flexibility; each workflow requires additional setup or development. Thus, such large machines may be relatively difficult for older farmers to operate because learning ability, memory, and physical strength are prone to deterioration over time.

Machines that use low-cost microprocessors embedded in hardware for capturing images in the work area can employ network models that have been pretrained using deep learning and installed in the server machines in order to improve the accuracy and speed of image recognition. Additionally, this reduces the computational burden for processors. Most importantly, such machines cost much less than conventional agricultural machinery. By using artificial intelligence–based learning technologies and image recognition technologies, neural network models applied in robotic arms are capable of analyzing known and unknown objects. Furthermore, by employing already optimized algorithms that can quickly mine and analyze the information and characteristics of various objects, robotic arms can precisely manipulate a target object through correct judgment. This is similar to how humans are able to analyze the characteristics of an object according to its weight, texture, and size and is applicable to the learning and operation of unknown objects. In addition, on the basis of IoT technology, this study designed a human–machine interface for operator use. Wireless network transmission technology was used to establish device communication, including the real-time transmission of images from the work area, reception of object detection results from the server side, and transmission of movement control commands. Furthermore, location information collected using a GPS module was displayed on the webpage to provide farmers with orientation when controlling devices remotely.

This study focused on the application of a system that uses smart recognition, smart IoT, and harvesting using robotic arms. The system aimed to provide information corresponding to farmers' needs, and smart systems were integrated accordingly. In its initial stage of development, smart recognition referred to the conversion of spatial information captured by the human eyes into that for computerized machines and training of the machines to have recognition skills similar to that of humans. Ultimately, the developed system can be applied to humanoid robots, machine devices, and diversified smart computational and recognition systems. Smart IoT, through the prevalence of wireless environment installations, has reduced the gap between actual machinery movement and user-end smart system installations. With technological development, farmers can employ technologies to monitor farm conditions from their homes by using merely a few fingers and connecting to a network. Harvesting using robotic arms is a semiautomated method that reduces the amount of time spent by farmers in performing repetitive tasks; with machine learning, new robotic movements can be explored. Furthermore, the extensibility, scalability, and usefulness of the system facilitate its integration into various industries.

This study hopes to equip a new generation of young people with the ability to quickly take over work in the agricultural industry. Farming requires extensive experience, and every agricultural skill is the outcome of a farmer's dedicated labor. However, the trends of modern technology prevent some of these skills from being documented and passed on to the next generation; as a result, many are being lost. Accordingly, this study endeavored to develop a system that can be easily adopted by the next generation of farmers. Through smart recognition models trained using neural networks, the proposed system can determine whether crops are ready for harvest, after which robotic arms can be employed to harvest the crops.

II. Related Work

This section introduces technologies related to those employed in the current study that have facilitated the introduction of subsequent systems, including object detection and robotic arm movement prediction. Finally, the main experimental objectives of this study are summarized in detail.

Image classification has become one of the most exciting areas of computer vision and artificial intelligence. With the development of convolutional neural network (CNN) architecture, computers can now perform exceptionally well in certain recognition systems, such as those for facial recognition. AlexNet [2], which is a CNN based on LeNet with some additions [3], won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by amplifying training data, using Rectified Linear Units (ReLUs), and increasing computational speed by employing graphics processing units. This subsequently sparked interest in object detection.

ZFNet [4], winner of the 2013 ILSVRC, was basically AlexNet with some minor modifications. To store additional information in images, it used 7×7 kernels instead of 11×11 kernels in the first convolutional layer. Although the VGGNet [5] did not win the ILSVRC in 2014, it remains one of the most common CNN used today because of its simplicity and effectiveness. Its primary concept is replacing large convolution cores by stacking several small cores, which increases the depth of the neural network and improves its accuracy. In 2014, the year it was introduced, GoogLeNet won first place in the ILSVRC. The model, which was named Inception [6], featured a sparse network structure design and enhanced neural network accuracy. The Inception model has several versions (e.g., V1, V2, V3, and V4) and is close to being optimized.

ResNet [7], introduced in 2015, constitutes a milestone in CNN image classification. It won multiple competitions at the ILSVRC with its 3.57% error rate. The high accuracy rate of ResNet is due to its network depth, which is greater than that of its predecessors. Additionally, to solve the degradation problem of the deep web, residual learning is used for implementing the network structure. The idea was that in the situation when subtracting the input features from the original learning features yields zero features, the stacked layers merely perform identity mapping. In 2017, a Google team proposed the CNN architecture MobileNet [8]. Because CNN architectures are generally very large and not suitable for use by low-end devices such as mobile phones, the Google team proposed new CNN computation methods, namely, depthwise convolution and pointwise convolution. In a large network model, most computations occur in the CNN; H, W, and N represent the height, width, and depth of the input data, respectively, which are multiplied by N_k number of $k \times k$ size convolution kernels of depth N of the input data, yielding $H' \times W' \times N\_k$.

Google proposed a second version of the MobileNet [9] in 2018. Its computational load decreased from 4.24 million parameters to 3.47 million parameters, its reported accuracy increased from 89.9% to 91%, and it was included in the ImageNet classification set. Thus, the new version not only substantially reduced computational load but also improved recognition accuracy. This study used the latest MobileNet neural network model as the infrastructure and network layer for achieving a favorable recognition accuracy rate.

Faster region–based CNN (R-CNN) [10] is an improved version of the R-CNN [11] and Fast R-CNN [12]. Both the R-CNN and the Fast R-CNN models require the use of the Selective Search method to generate region proposals before calculating the feature map. This part of the algorithm derives the result by using the loop method; thus, the computational speed at this part of the algorithm is low. The Faster R-CNN model proposed in 2016 is a faster version of its two predecessors.

Faster R-CNN is highly intuitive; it does not generate region proposals through the slow selective search method but instead inputs an image and captures a feature map after CNN computation. Subsequently, numerous region proposals are obtained through the Region Proposal Network (RPN). The final result is obtained from a layer similar to the RoIPooling layer of a Fast R-CNN.

The feature map output by the CNN includes the bounding box (b-box) and the probability of a b-box object. Because the RPN is a type of CNN, the RPN can directly input a previous CNN output. The sliding window calculation method is adopted in the RPN, and the central point of the sliding window is called the anchor point. The k number of boxes of varying sizes set in the pretraining model parameters is calculated, and the box with the highest probability score of containing the object is taken as the box. Through RPN calculation, numerous b-boxes are derived, although they may not all be accurate. However, through RoIPooling, the boxes are quickly classified to obtain the most accurate b-box.

The name YOLO, most popularly an acronym for “You Only Live Once,” represents “You Only Look Once” in the context of in object detection. Accordingly, a YOLO detection model must only perform one CNN calculation for an image to determine the type and position of the object inside, thus improving detection and recognition speed. YOLO is a neural network algorithm implemented in a niche darknet architecture. Joseph Redmon, the creator of the architecture, did not use any deep learning architecture but achieved YOLO through the characteristics of light weight, low dependency, and high computational efficiency. Hence, it has high application value in industries such as pedestrian detection and industrial image detection. Redmon released three versions of YOLO that sequentially offered improved detection accuracy.

YOLOv1 [13] was proposed in 2015. Its core concept is similar to that of R-CNN, which operates by proposing regions first and then making determinations. In R-CNN, the scope examined is limited only to the region; because the scope is smaller, background patches can easily be mistaken for the object. However, YOLOv1 examines the complete image during both the training and detection stages and calculates the positions and classification type of b-boxes directly in the input layer using regression algorithms. Hence, its background error rate is only half that of Fast R-CNN. YOLOv1 divides an image into S×S number of grid cells. If the central point of an object is located in a particular grid cell, that specific grid cell becomes responsible for predicting the particular object.

YOLOv2, proposed in 2016, improved upon YOLOv1. It offered higher accuracy, a faster detection rate, and the ability to recognize more than 9000 types of objects. YOLOv1 had numerous shortcomings that its creator aimed to improve, including recall, the accuracy of the b-box, and the classification accuracy. The update content is listed in Table 2. YOLOv3 did not feature specific innovations but demonstrated improvement compared with models proposed in other studies. Darknet-53 was used as the new network architecture, which had more layers and thus improved detection accuracy compared with the previously developed Darknet-19. Furthermore, YOLOv3 used ResNet architecture, which is often used in general neural networks for deepening. To enhance the detection ability for small objects, feature pyramid networks (FPNs) were used and the single feature layer was replaced with multiple feature layers. These layers had sizes of 13 × 13, 26 × 26, and 52 × 52 sizes, and the number of b-boxes was changed from five in each layer to three in each layer, yielding a total output of 3 × 3 types. Consequently, the FPN architecture enabled the b-box of subsequent layers to merge semantic information with that of previous layers, thus yielding excellent results in the detection of small objects.

In 2015, a Google team proposed a neural network detection model called a single shot multibox detector (SSD). The model did not have an accuracy rate as high as that of Faster R-CNN, but it did have a shorter detection execution time. Compared with YOLO, it did not have a faster detection execution time, but it did have higher detection accuracy.

According to the aforementioned description, the procedure of an object detection model can be divided into two parts: (1) a fundamental convolutional network layer (e.g., Inception, ResNet, and MobileNet) that extracts features from the original image and (2) an object detection network layer (e.g., Faster R-CNN, YOLO, and SSD) that categorizes image features and determines selection scopes. However, detection accuracy and computation speed must be balanced. MobileNet is a highly suitable fundamental convolutional network layer for SSD because the primary focus of these two networks is to reduce the computation volume while sustaining a specific level of detection accuracy. Therefore, we adopted a MobileNet–SSD model as the research method.

III. System Model

Because the ages of farmers vary considerably and young farmers have more opportunities to change careers, the inability of young farmers to acquire the skills of previous generation continues to be a problem. This study used IoT technology with deep learning algorithms to design a remote harvesting system with smart crop identification ability for use by young farmers.

Fig. 2 shows that the architecture proposed in the present study can be divided into three parts, namely object detection, arm control, and a communication system. The object detection system is largely responsible for image detection and sends detection results to the arm control system for arm movement prediction. Finally, the communication system sends image information to the object detection model for detection and enabling arm control.

Object detection is performed using the MobileNet SSD CNN model constructed using TensorFlow, and system tolerance is attained primarily by adjusting training images and training different numbers of images in batches to test the model network's detection accuracy.

Arm control includes the assembling of hardware. The hardware employs Raspberry Pi, which is responsible for image streaming and receiving webpage commands; robotic arms, constituted by six servomotors; webcams; a GPS module; an ultrasonic sensing module; and a tracked mobile vehicle that carries all of the various objects.

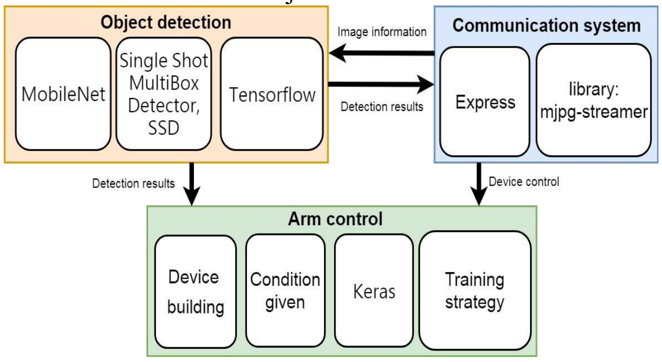


Fig. 2. System model

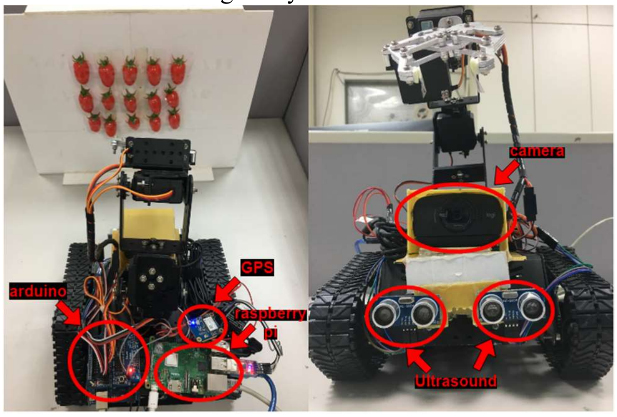


Fig. 3. Modular design of the hardware prototype

The communication system mainly serves as a bridge for communication between the front and back ends of the system and performs some complex computations, including image streaming, detection, transmission of detection results to the webpage, and the reception or issuance of movement commands from the webpage. The user webpage enables farmers to perform remote control and displays various information that influences harvesting decisions, such as streamed images, postdetection results, the distance determined by ultrasonic sensors, arm control schematics, arm control buttons, and the GPS position of tracked mobile vehicles.

As shown in Fig. 3, in consideration of the unevenness of agricultural land, this study used a tracked mobile vehicle as a carrier for the robotic arms and installed webcams to capture real-time images. Raspberry Pi transferred data through a wireless network, controlled movement of the tracked mobile vehicle, and received ultrasonic measurement data and the geographic coordinates of the system from the GPS. Arduino was used to control the robotic arm. The ultrasonic sensors measured the distance between the vehicle and the objects, and these measurements served as the reference value; the GPS module sent current coordinates to the server.

A. Object Detection Model

Many types of object detection neural network models currently exist. However, this study required a system that features both detection accuracy and detection speed. Hence, performance in these two aspects needed to meet this study's requirements. By referring to the results of various object detection neural network models, this study ultimately selected MobileNet [8], [9], a base CNN architecture with low computational load, and paired the system with SSD [17], a b-box algorithm that accommodates objects of various sizes.

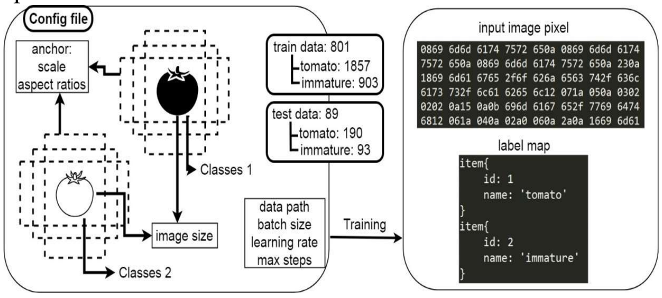
Usually, different sets of training data are collected during machine training to avoid obtaining a model with overfitting that yields results that appear to be but are not actually accurate. Hence, this study searched for object type, image capture angle, camera distance, ambient lighting, and ambient complexity on the Internet to increase the neural network model's understanding. Images were captured from farmlands to simulate actual settings and thereby improve the model's accuracy. In addition, image size and resolution were considered; during online search, this study used images with an 800 × 600 resolution captured using smartphones as the training data.

Two classifications were used, namely, tomato and immature. The proposed system determined whether a fruit was sufficiently ripe for harvesting. Training images of two sizes were used to test the effect of training image size on model accuracy. The zoom level of the b-box was determined according to the size presented in the images. The present study adopted farm crops as the detection objects and expected them to be detected at varying sizes. Hence, the zoom level was set in accordance with the satisfactory level used in [35].

The number of training batches was determined according to the system memory of the graphics card; specifically, for the GTX 1080 8G graphics card, a batch size of 24 images was set. To achieve a stable state of the neural network model posttraining, the number of training steps was set as 200 000; the initial training rate was set as 0.003.

This study prepared a total of 890 training images that were randomly dispersed in the main data set at a ratio of 1:9. Of the 890 images, 491 were obtained from online sources, whereas 399 were captured in real farmland settings. A total of 442 online images were used in the first round of training, which included 812 ripe fruits and 528 unripe fruits ; 49 images were tested, which included 78 ripe fruits and 51 unripe fruits.

A total of 801 images were used in the second round of training, which included 1857 ripe fruits and 903 unripe fruits; 89 images were tested, which included 190 ripe fruits and 93 unripe fruits. When training the neural network model, pixel values and tags that were converted before training were automatically introduced into the model for training in accordance with the settings in the config file. After training, changes to the total loss value were determined from the files specific to TensorFlow.



```
0869 6d6d 6174 7572 650a 0869 6d6d 6174
7572 650a 0869 6d6d 6174 7572 650a 230a
1869 6d61 6765 2f6f 626a 6563 742f 636c
6173 732f 6c61 6265 6c12 071a 050a 0302
0202 0a15 0a0b 696d 6167 652f 7769 6474
6812 061a 040a 02a0 060a 2a0a 1669 6d61
```

```
item{
    id: 1
    name: 'tomato'
}
item{
    id: 2
    name: 'immature'
}
```

Fig. 4. Schematic of the Training Files Setup

B. Arm Movement Prediction Model

This study aimed to use robotic arms to harvest target fruits semiautomatically. Hence, the arms needed to be modelled to execute harvesting movements smoothly, integrate with the detection results of the neural network model, and execute various appropriate movements. Specifically, control commands were issued from the user webpage end; upon receiving the commands, Raspberry Pi sent the commands to the robotic arm through serial ports, and the arm executed the appropriate movement. In addition, to compensate for the unevenness of agricultural land, this study used a tracked mobile vehicle as a carrier for hardware equipment.

We constructed a semiautomated smart robotic arm mobile vehicle. Commercially available robotic arms are generally large industrial arms that are dexterous and precise but costly or those developed by Maker that are small and lightweight but limited to picking small objects. This study assembled a robotic arm and modified the specifications and performance according to our needs. Regarding the arm design, we prioritized the appropriateness of the arm size and weight for mounting on a tracked mobile vehicle. Moreover, we aimed to reduce development cost and apply our research to everyday life; thus, the arms needed to satisfy most needs in everyday life, feature mobility and safety, and be lightweight and sensitive. The electronic potential of the robotic arm for increasing torque was also required to fall between 12–24 V to meet the requirements of current battery designs.

To enable modifications and control over robotic arm precision, this study used Maker arms for the robotic arms, which were composed of six servomotors. However, the precision and torque differed among servomotors, so the servomotors had to be replaced with ones suitable for this study: TR213 servomotors were selected (Fig. 5). The motor featured fully metallic gear wheels, 13-kg torque, and robot-specific steering gears with a precision of 0.5 degree.

Arduino was used for arm control. The design includes a process flow for arm movement and requires users to input commands to control the arms. The design of a robotic arm with smooth movements prevents collisions, thereby extending the arm's lifespan (Fig. 7). The design serves as a reference for users when operating the robotic arm.

Fig. 5. TR213 servomotors

Fig. 6. Diagram of the HN-35GBE-1640Y Gear Motor

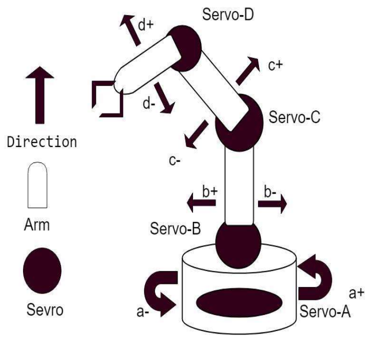


Fig. 7. Schematic of the Robotic Arm Axis System

Regarding the carrier, this study selected an unmanned ground vehicle that was suitable for installing the designed arm, fitted with Raspberry Pi and control panels, and that had electrical circuits and motors that could be modified and controlled. Confronted with the lack of a suitable vehicle, this study self-designed a vehicle to construct the semiautomated robotic arm and mobile vehicle. Because the tracked mobile vehicle needed to carry various types of equipment, the motor of choice needed to be capable of driving the entire vehicle. Ultimately, this study selected HN-35GBE-1640Y (Fig. 6), a direct current motor with a high torque. Because a motor with high revolutions per minute (rpm) was not suitable for this study, a speed of 60 rpm was selected for the motor.

This study used the HC-SR04 ultrasonic sensor module to perform onsite environment measurements. This module is easy to operate, and its simple design can be integrated with most hardware. Although the sensor's measurement distance ranges from 2 to 450 cm, error values were returned during actual measurements with a distance of less than 5 cm. An NEO-6M GPS module was used to track the location of the mobile vehicle. This module was a U-BLOX NEO-6M module featuring a high-performance ceramic passive antenna, a rechargeable backup battery, and warm start capability. The backup battery provides approximately 30 min of backup power for storing GPS information in the event that the main power supply was cut off. For faster application, a plug-and-play webcam that did not require additional driver installation was used, namely Logitech C525. The webcam had a field of view (FOV) of 69°, featured autofocus, and had built-in automatic light correction, enabling a clear view of objects even in low-light environments.

Regarding arm movement prediction, this study did not correct the axis coordinates of the camera with those of the robotic arm because the camera had an autofocus function that would interfere with value correction. In addition, the robotic arm had six axes, rendering the design of sequence detection difficult. Therefore, Multilayer Perceptron (MLP) Machine Learning was used to predict arm movement and position.

Because each servomotor formed an angle when the robotic arm moved to a specific position, this study set various conditions beforehand. Fig. 8 shows how the arm collects the pixel coordinates of the object detection results as the input and output data for MLP under three conditions. This study used a webcam as the image-capturing device. However, when the object size is not constant, depth information cannot be easily obtained using a single-lens reflex camera, thereby rendering it difficult to obtain the precise distance between the robotic arm and the detection object. Hence, as indicated by Condition 1 in Fig. 8, the object must first be within the camera's FOV; therefore, an ultrasonic sensor module was installed in front of the mobile vehicle to measure distance and apply the measurement results as depth information. Because of the wide measurement range of ultrasonic waves, we set the sensor distance to 20 cm in accordance with Condition 2, as shown in Fig. 8.

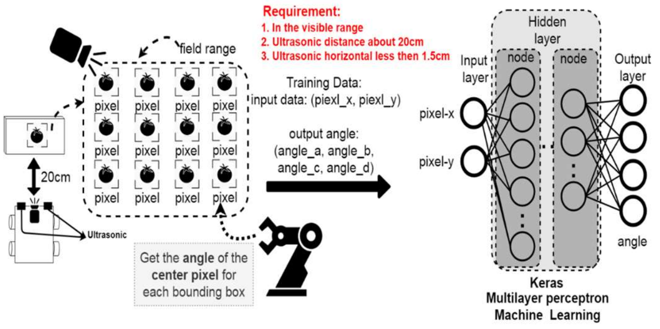


Fig. 8. Schematic Depicting the Conditions Set for Collecting the Training Data for the Arm Prediction Model

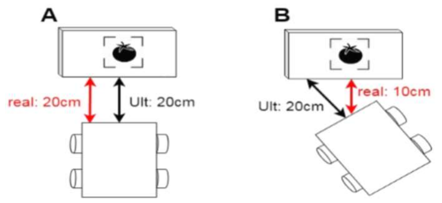


Fig. 9. Schematic Depicting Various Problems in the Relationship between the Vehicle and Detection Objects

Fig. 9. depicts another problem that occurs when a mobile vehicle is not level with the detection object. Measurements using ultrasonic waves are obtained by calculating the time taken by an emitted signal to reach a vertical target and return to its source. Therefore, when an angle was generated between the mobile vehicle and the detection object, the distance measured by the ultrasonic sensors was not the actual distance. To solve this problem, two ultrasonic waves were used to measure whether the vehicle was level with the object in front of it. As indicated by Condition 8 in Fig. 8, the difference between the two ultrasonic waves should be less than 1.5 cm; this was set as the level standard for detecting the objects in front of the vehicle.

Model training data were collected after the three conditions were satisfied. Following object detection, b-boxes were generated on the object shown in the images. Next, the pixel coordinates of the central point of each b-box were used as the input, whereas the angle formed when the arm actually moved to each object was used as the output. Finally, a model was generated using Keras, which was trained using supervised learning.

The blue dot in Fig. 10 represents the pixel coordinates of a b-box's central point. To improve prediction accuracy, the collected coordinates were evenly distributed among the two-dimensional FOV coordinates. Fig. 10 depicts the three-dimensional space of the central point's pixel coordinates corresponding to each angle. Although the robotic arm was composed of six servomotors, the use of only four servomotors was sufficient to move the arm to required positions; the remaining two servomotors performed the object picking action. The predicted angles are named A, B, C, and D. Fig. 11 shows a linear relationship between angle A and the central point's pixel coordinates but a nonlinear relationship between the central point's pixel coordinates and angles B, C, and D. According to these relationships, this study achieved the required positions through the use of neural networks for predicting the four angles.

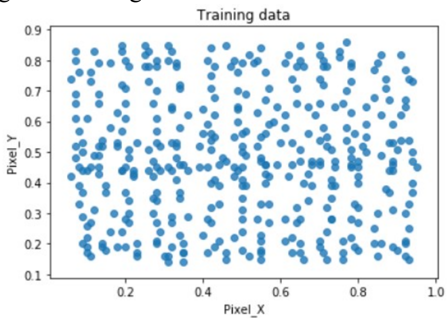


Fig. 10. Visualization of the Input Data of the Robot Arm Prediction Model

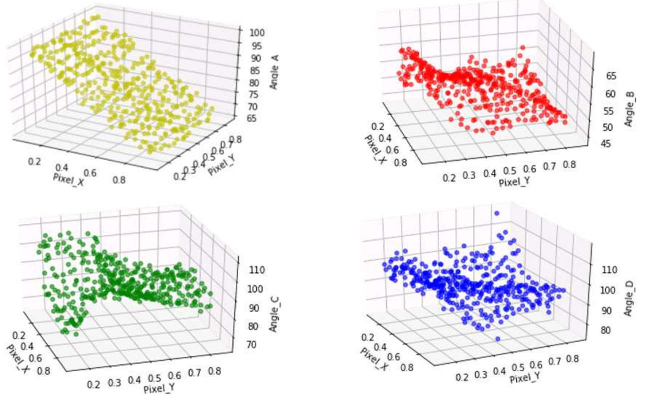


Fig. 11. Visualization of the Output Data of the Robot Arm Prediction Model

C. *Establishing the Model Using Keras*

Keras is an open-source neural network library that is written in Python and can be run on TensorFlow. Keras is designed for the fast realization of neural networks and focuses on user-friendliness, modulization, and extensibility. Program-writing in Keras is much simpler and clearer compared with that in TensorFlow and uses a stacking approaching for model construction. Thus, its models are extremely versatile.

This study used a neural network to predict angle values. The prediction range was 0°–180°, which was excessive as an input. Therefore, to increase the network's convergence rate, the predicted values were normalized using sklearn in the Python package. This reduced the range to ±1.5. Because the predicted output values were normalized, the values were small. To avoid convergence failure due to an excessively small training loss, the cost function of the mean absolute value error was used, which is presented as follows:

$$cost\ function = \frac{1}{n}\sum_{i=1}^{n}|(y_{true}^{(i)} - y_{pred}^{(i)})| \qquad (1)$$

where n = 4 and represents the four angles. Various problems may occur in machine learning. As depicted in Fig. 12, Situation A may occur when the learning becomes stuck in a local minimum and fails to obtain the optimal solution. Situation B may also occur when the optimal solution cannot be determined because of an excessively slow convergence rate. Finally, in Situations C and D, failure to derive the optimal solution may occur because of an excessively fast convergence rate. Hence, training a satisfactory model in machine learning is challenging. Using neural network training is an excellent method; however, it does not guarantee that the value it converges to is the optimal solution or that the neural network can converge to the optimal solution.

To compensate for the aforementioned deficiency, this study adopted the optimization training method used by Adam [23-26] and provided by Keras. This method has been proven to be an excellent optimization method and modifies the learning rate to find the lowest point in the overall gradient change. The initial learning rate in this study was set to 0.001 in anticipation that the model would learn at an appropriate speed.

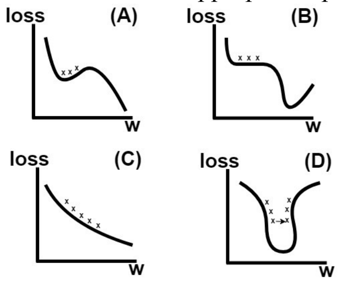


Fig. 12. Possible Problems Encountered by Machine Learning

The commonest approach to determining the number of iterations required for deriving the optimal solution when training neural networks is setting a maximum iteration limit. Subsequently, changes to the training loss value can be observed in order to identify the stage with the most favorable number of iterations. However, this procedure is time consuming. There are two types of training loss values; the first is the loss that occurs as a result of training data, and the second is the validation loss that occurs as a result of testing data. Excessive iterations in neural networks results in overfitting. When this occurs, the loss value decreases while the validation loss increases. Keras provides a smart solution that facilitates the observation of changes in the validation loss value during training. This study set the number of iterations to 100; training was terminated if there was no improvement in validation loss.

As depicted in Fig. 13, we used a four-layered perceptron to perform prediction. The number of nodes in each layer differed and was set by referencing the inverted residuals [9] concept of MobileNet. By applying the concept of amplification first and diminishing later, the feature extraction amount of the first layer was improved, thus enhancing the accuracy of the neural network model. The nodes of each layer were the weight values of the features. The ReLu method, a common method in machine learning, was used for the activation function. For the final layer, a linear activation function directly predicted the four angles.

During model training, some of the training data are used as testing data to verify model performance. However, this means that all modifications that the trained model undergoes are performed according to the same testing data set. The developed model would therefore develop dependency on a specific set of testing data. To prevent this result, we used k-fold cross-validation and compared the model results ten times using ten folds (Fig. 14). The advantage of using this method is that for each model, a different set of testing data is used during training and a prediction accuracy result is generated. The ultimate model performance is determined according to the collected accuracy results. To stop the model at the optimal number of iterations, an early termination condition was set. A validation set was used to calculate the validation loss of the entire model, which was monitored continually during the training iterations. Training was terminated when the validation loss value showed no improvement after 100 iterations.

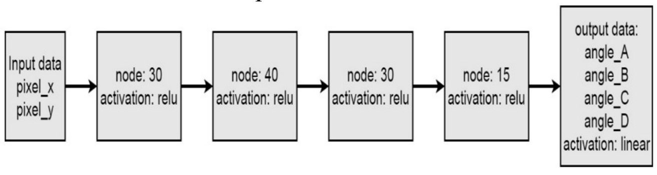


Fig. 13. Architecture of the Robotic Arm Prediction Model

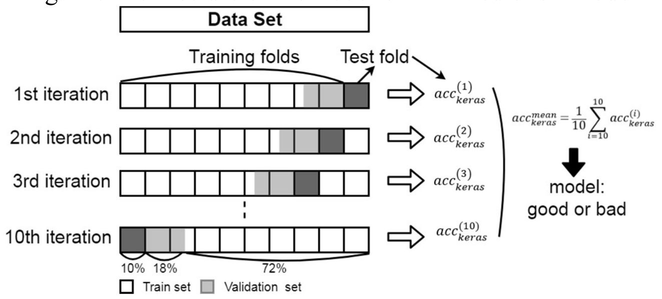


Fig. 14. Schematic of k-Fold Cross-validation

During model training, some of the training data are used as testing data to verify model performance. However, this means that all modifications that the trained model undergoes are performed according to the same testing data set. The developed model would therefore develop dependency on a specific set of testing data. To prevent this result, we used k-fold cross-validation and compared the model results ten times using ten folds (Fig. 14). The advantage of using this method is that for each model, a different set of testing data is used during training and a prediction accuracy result is generated. The ultimate model performance is determined according to the collected accuracy results. To stop the model at the optimal number of iterations, an early termination condition was set. A validation set was used to calculate the validation loss of the entire model, which was monitored continually during the training iterations. Training was terminated when the validation loss value showed no improvement after 100 iterations.

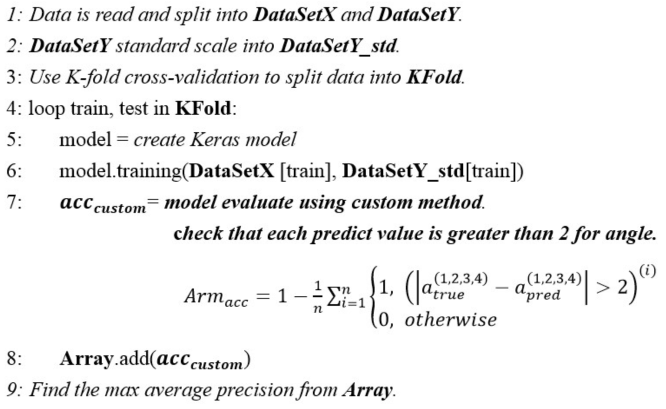
Equation (2) was used for calculating the model accuracy of Keras. The calculated loss value was a continuous interval. Therefore, using the results with the highest accuracy obtained using this equation was not appropriate for prediction in actual practice because the robotic arm's picking operation would fail when the predicted value of any of the arm's four servomotors exceeded ±2°. This system designed a more rigorous method to validate model accuracy, as shown in (3) for assessing actual object picking performance. The condition entailed deriving the absolute value of the difference between actual values and predicted values; a prediction was treated as an error when any of the four angles exceeded 2°. The mean total error was then calculated to obtain the model's accuracy.

$$acc_{keras} = 1 - \frac{1}{n}\sum_{i=1}^{n}\left(y_{true}^{(i)} - y_{pred}^{(i)}\right) \quad (2)$$

$$Arm_{acc} = 1 - \frac{1}{n}\sum_{i=1}^{n}\begin{cases}1, & \left(\left|a_{true}^{(1,2,3,4)} - a_{pred}^{(1,2,3,4)}\right| > 2\right)^{(i)} \\ 0, & otherwise\end{cases} \quad (3)$$

Fig. 15 presents the pseudo code for the arm prediction model. The first three lines mainly preprocess the training data, including dividing the data into model input and output, normalizing output data, and using k-fold cross-validation to divide the training data into ten sets. The ten sets of data are then divided into separate training models through looping. After training completion of the sixth line, the model is incorporated into the arm prediction accuracy equation and saved into the arrays. When all ten sets had completed training, the set with the highest relationship score in the array was selected as the optimal model.

This study adopted stagewise training because robotic arm control is cumbersome. The process flow is shown in Fig. 16. First, a neural network is established. Data are then collected to train the model. Next, the model's predicted results are tested. If the predicted angle is not a desired position, then data obtained postadjustment are logged.



```
1: Data is read and split into DataSetX and DataSetY.
2: DataSetY standard scale into DataSetY_std.
3: Use K-fold cross-validation to split data into KFold.
4: loop train, test in KFold:
5:      model = create Keras model
6:      model.training(DataSetX [train], DataSetY_std[train])
7:      $acc_{custom}$= model evaluate using custom method.
                check that each predict value is greater than 2 for angle.
                $Arm_{acc} = 1 - \frac{1}{n}\sum_{i=1}^{n}\begin{cases}1, & \left(\left|a_{true}^{(1,2,3,4)} - a_{pred}^{(1,2,3,4)}\right| > 2\right)^{(i)} \\ 0, & otherwise\end{cases}$
8:      Array.add($acc_{custom}$)
9: Find the max average precision from Array.
```

Fig. 15. Model Training Pseudo Code

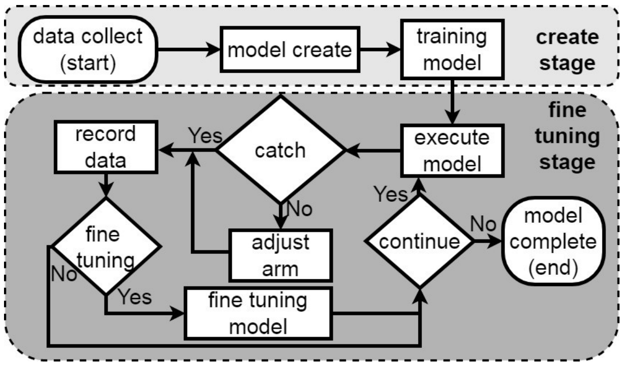


Fig. 16. Flow Process of Model Training and Adjustment

To increase the amount of training data in order to obtain a stable neural network model, data are logged even when the model derives the optimal angle. This study proposes a neural network system with real-time adjustment. As indicated by the adjustment stage of system development (Fig. 15), users can decide whether model adjustment is necessary upon collecting new angles.

D. Full-Stack Communication System

This study used the MIAT system design. According to the system design requirements, hierarchical modular design was conducted. Fig. 17 depicts the Integrated Computer-Aided Manufacturing DEFinition for Function Modeling (IDEF0) architecture. The system's parts were divided into numerous groups of independent modules, which assisted in clarifying and defining the system and facilitated team support and codesign. Subsequently, GRAFCET, a visualization tool, was used to construct the discrete event model of each module group, representing the module design of its subsystems.

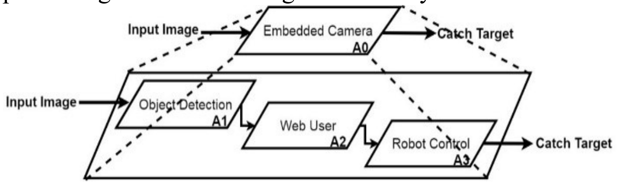


Fig. 17. IDEF0 Hierarchy Structure of the Proposed System

The A1 subsystem could be further divided into two subsystems, as shown in Fig. 18. Images are sent to the user's webpage through a wireless network, enabling the user to decide whether the images should be captured and sent to the server. Upon receiving the images, the server inputs them into the trained neural network model for object detection, which is completed within seconds. The detection results are then sent to the webpage and displayed.

The A2 subsystem could be further divided into four subsystems, as shown in Fig. 19. Different screens are displayed according to the input source. The webpage continually monitors and displays the streaming video. This enables the user to view current conditions and decide whether images should be captured and sent to the server for object detection. The detection results are then sent back to the webpage for user decision-making. If an object is detected, further actions through the use of robotic arms can be conducted on the webpage, and the actions can be monitored through video streaming.

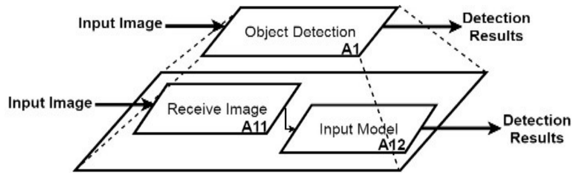


Fig. 18. IDEF0 Hierarchy Structure for Object Detection

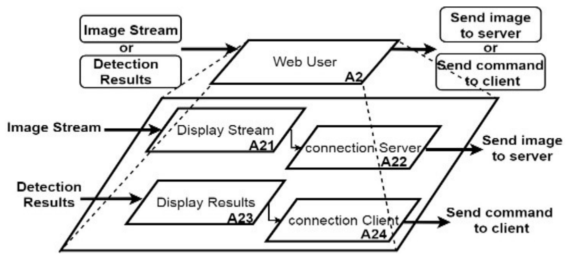


Fig. 19. Web User IDEF0 Hierarchy Structure

The A3 subsystem could be further divided into two subsystems, as shown in Fig. 20. This study selected Raspberry Pi as the relay station because its low power consumption facilitates continual monitoring of messages sent from the network side. When control commands are received from the user webpage, the commands are issued to the robotic arm for action through serial ports.

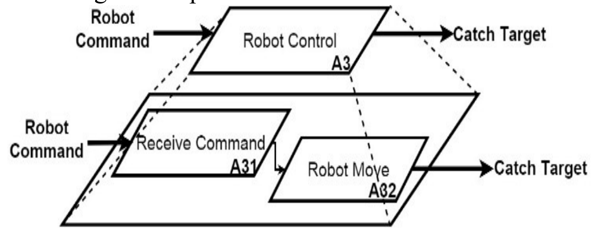


Fig. 20. Robot Control IDEF0 Hierarchy Structure

The development environment of the present study can be divided into two parts (Fig. 21). On the server side, the Ubuntu operating system was used for construction because of its low power consumption; it also delivers favorable speed in training neural networks and in detection operations. The core program was written using Python, which is a cross-platform program language. Because of current trends in the artificial intelligence industry, it has numerous supporters; thus, support could be easily obtained through online searches.

The server is mainly responsible for executing relatively complex computations and processing, including object detection and arm movement prediction. A human–machine interface was also installed in the server. Responsible for implementing control hardware operations, the control side includes the Raspberry Pi–setup mobile vehicle paired with a robotic arm, which receives commands through serial ports. A wireless network serves as the means of communication between the server side and the user side; thus, operating the proposed system requires both sides to have Internet access.

The webpage server was created using the Express module of Node.js, which is an open source code that can execute JavaScript on a server and across platforms. Originally, Node.js did not support other common web development tasks. For example, if using a different HTTP method, such as GET, POST, DELETE, or other specific requests; providing static files for different uniform resource locators; or using templates or generating dynamic responses, related programming needed to be completed manually or by using a web framework.

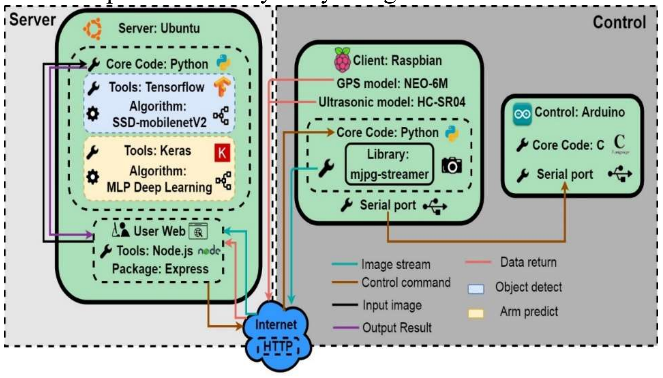


Fig. 21. System Development Tools and Techniques

Express is the most popular Node.js web framework. It features simple and flexible operations, provides a powerful set of tools for creating a variety of web applications, and offers rich HTTP tools. A full-featured website can be quickly constructed using Express.

The flow process of communication between the server side and the human–machine interface can be divided into front, middle, and back operations (Fig. 22). In front operations, users first connect to the server and then decide what requests to send (e.g., object detention, moving equipment in order to view video streaming). In middle operations, the server receives requests, converts them into a format compatible with the back operations, and sends them accordingly. Back operations involve the mobile vehicle; the vehicle immediately executes a relevant response upon receiving information that must be sent back to the server, whereas upon receiving data, the server converts them into the format required by the user. Finally, the information is rendered on the user's webpage for browsing.

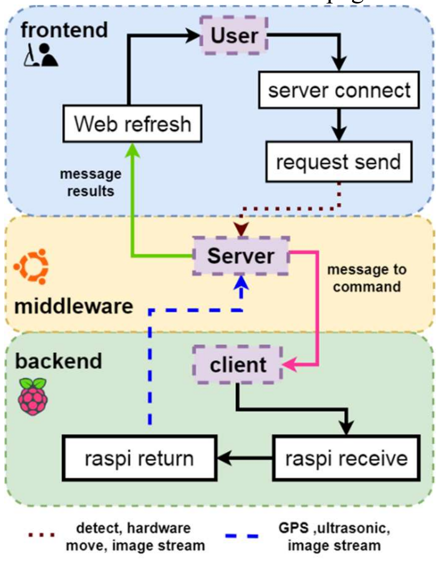


Fig. 22. Server Communication Process Flow

The proposed system uses HTTP protocols as the method of communication between the front-end user and the back-end vehicle. The primary purpose of the system is downloading the images onto the server operating system to facilitate the neural network model's detection operations and display streaming images. Front-end users can consequently determine the real-time status of matters on the mobile vehicle's side. The secondary purpose of the system is sending arm movement commands from the webpage side to the back-end vehicle to implement required actions and send commands for vehicle control. In the proposed system, customization is enabled such that the user can select the required movement distance.

IV. Results and Discussion

A. *Results of Object Detection Model Trasining*

This proposed system uses object detection neural network models to detect whether a crop has matured. To apply the proposed harvesting system, differences between the various neural network models had to be analyzed and compared. To compare the neural network models, two different numbers were used for the numbers of images in the training and testing data and two image sizes (i.e., 320 × 320 and 512 × 512) were adopted. The aim was to observe the influence of compressing images on the proposed application. This study implemented three object detection models, namely an SSD model developed using MobileNet version 2 as the base network model, a faster R-CNN model using ResNet 152 as the base network model, and a YOLOv3-based model.

The parameters trained in this study are listed in Table 1. Using the three aforementioned methods, stagewise training was employed to train data sets with various numbers of images and image sizes. In the training data sets, 491 images were images retrieved from the Internet and 399 were images captured from real farmlands. Combined, they comprised 890 images, which were retrained. The training results are presented as follows.

Two classifications were used, namely "tomato" for ripe tomatoes and "immature" for unripe tomatoes. When training the SSD model, the average precision (AP) and mean AP (mAP) predicted for each classification were logged; all had intersection over union (IOU) values greater than 0.5.

TABLE I
Parameter Setup of the Neural Network Model

Name	Parametric values
Training image size	320 × 320 or 512 × 512
Number of training images	442 or 801
Number of classification types	2
Learning rate	0.003
Maximum training steps	200000
First-round training numbers	Total images: 442, Ripe objects: 817, Unripe objects: 528
First-round testing numbers	Total images: 49, Ripe objects: 78, Unripe objects: 51
Second-round training numbers	Total images: 801, Ripe objects: 1857, Unripe objects: 903
Second-round testing numbers	Total images: 89, Ripe objects: 190, Unripe objects: 93

To identify the most suitable object detection model, this study constructed three different detection models and trained them under four different conditions for comparison (Table 2). The MobileNet version 2 SSD model was selected as the ultimate model. Fig. 23 and Fig. 24 illustrate its training process. As shown in the figure, extreme changes occurred prior to the 60 000th training step and gradually stabilized only after the 60 000th step. The model training reached near saturation at approximately the 100 000th training step, and no major changes occurred until the 200 000th step. The entire training spanned 37 hours; however, the training can be terminated after 18 hours.

Some data for the two classification types (namely, tomato and immature) of the various models are displayed in Fig. 25, which shows that in the MobileNet version 2 SSD model, the AP for both classification types was greater than that of other models. In addition, training image size influenced the object detection models; the AP of training 512×512 images was slightly greater than that of training 320 × 320 images (Fig. 24).

To increase the training image complexity, additional images captured at actual farmlands were incorporated. The images mainly captured cherry tomatoes, which are characteristically small in appearance. The distance at which the images were captured was not set, and they were not taken as close-up as those retrieved from the Internet, which increased the training image complexity.

TABLE II
COMPARISON OF THE OBJECT DETECTION TRAINING RESULTS

Model	Number of training	Image size	tomato AP	immature AP	(mAP)	(GTX 1080 ti 8G) FPS
ResNet152 Faster RCNN	442	320 × 320	78%	77%	78%	6
ResNet152 Faster RCNN	801	320 × 320	85%	64%	75%	6
ResNet152 Faster RCNN	442	512 × 512	80%	84%	82%	5
ResNet152 Faster RCNN	801	512 × 512	83%	61%	72%	5
MobileNet ver2 SSD	442	320 × 320	81%	80%	81%	48
MobileNet ver2 SSD	801	320 × 320	89%	74%	81%	47
MobileNet ver2 SSD	442	512 × 512	82%	81%	82%	36
MobileNet ver2 SSD	801	512 × 512	**91%**	**76%**	**84%**	**40**
YOLO ver3	442	320 × 320	75%	68%	71%	49
YOLO ver3	801	320 × 320	76%	58%	67%	49
YOLO ver3	442	512 × 512	77%	68%	73%	32
YOLO ver3	801	512 × 512	79%	58%	68%	32

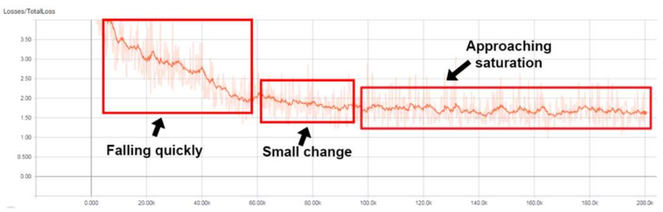


Fig. 23. Training Loss of the MobileNet ver2 SSD Model

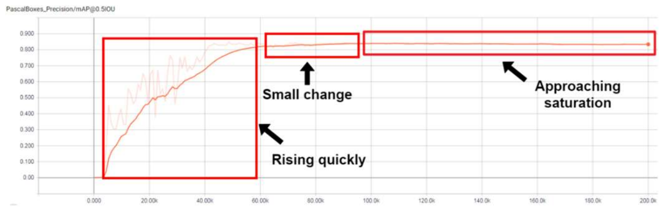


Fig. 24. Training mAP of the MobileNet ver2 SSD Model

Fig. 25 shows that except for the YOLOv3 model, the other two models had AP values that were only satisfactory after training with 442 Internet images. However, the detection performance of the YOLOv3 model was unsatisfactory for immature classification; the model sometimes misjudged an object as part of the image background. After incorporating 359 images captured at actual farmlands and performing retraining using 801 images (i.e., the new greater number of training images), the AP increased for "tomato" classifications but decreased for "immature" classifications (Fig. 25). This phenomenon indicates that the recognition performance of object detection models for objects that are relatively small and have a similar color to that of the background is relatively poor, with objects tending to be mistaken for the image background. This can be attributed to two causes; first, the training for the immature classification type may be insufficient, resulting in the model's inability to perform accurate classifications. Second, background images were not incorporated into the training, resulting in the model's inability to correct its results on the basis of background weight values.

Fig. 26 presents a visualization of the data from Table 2, which compares the mAP and frames per second (FPS) of the various models. The mAP of the MobileNet version 2 SSD model was markedly higher than that of the other two models. In addition, training image size influenced the object detection models; the mAP of training 512 × 512 images was slightly greater than that of training 320 × 320 images (Fig. 26). However, the image detection speed decreased as FPS increased. The FPS value of the ResNet152 Faster R-CNN model was extremely low, meaning that the model cannot be applied in practical applications, such as for live stream object detection; compared with the FPS of MobileNet version 2 SSD, that of YOLOv3 was slightly greater when training 320 × 320 images but slightly lower when training 512×512 images. This suggests that compared with the other models, the YOLOv3 model's detection speed was unsatisfactory when detecting relatively large images. Regarding the results obtained after incorporating actual farmland images, except for the mAP of the MobileNet version 2 SSD model, that of the other models decreased.

The final object detection network model ultimately needed to be selected from the models that performed under 12 different conditions. A simple procedure was devised, as shown in Fig. 27. The FPS and mAP values were visualized using a pie chart, and the percentage was used to select the final model. First, a high detection speed was desired; therefore, the models with an FPS exceeding 10% of the pie chart were selected, as shown in the first step of Fig. 27. Next, from these models, the one with the highest mAP in the pie chart was selected, as shown in the second step of Fig. 27. Ultimately, this study selected the MobileNet version 2 SSD model as the model to use.

As illustrated in Fig. 28–30, after the object detection neural network models were trained, 89 images of different sizes that had not been used in training were input into the model. The green and blue frames respectively indicated the tomato and immature classifications. Overall, the objects were clear and were largely not covered or otherwise occupied a large proportion of the image; thus, they could easily be detected regardless of their classification. Detection errors occurred under poor lighting conditions and when the object was excessively eclipsed, was too small in the image, or had a color similar to that of the background.

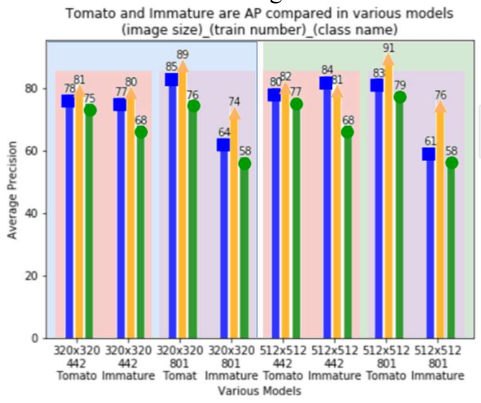


Fig. 25. Comparison of the Classification Accuracy of Various Models

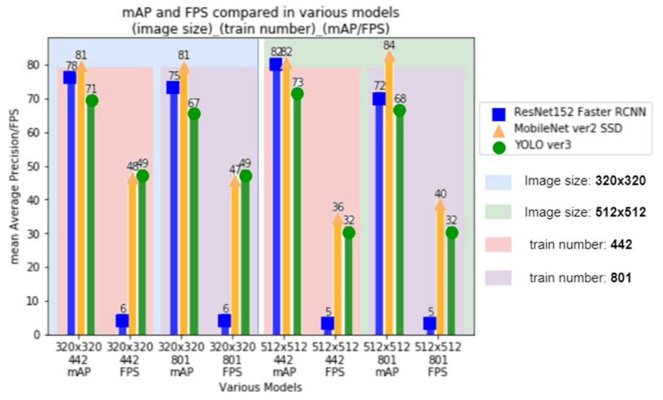


Fig. 26. Comparison of the mAP and Frames per Second of Various Models

Image 14 in Fig. 28 shows a detection error that may have occurred when the detection features reassembled the tomato classification after image compression. Image 72 in Fig. 30 shows a detected object that almost resembled the original object; however, the color of the defects and skin were highly similar, thus resulting in a detection error. The model failed to detect any object in Images 65 and 78. This could have been because the objects were too small in the images or poor lighting conditions. The training data included few pictures depicting dimly lit environments; thus, the model was not trained for detection under this condition. The error may have also occurred because the object did not present the defined tags, such as bearing fruit or fruit buds.

Image 14 in Fig. 28 shows a detection error that may have occurred when the detection features reassembled the tomato classification after image compression. Image 72 in Fig. 30 shows a detected object that almost resembled the original object; however, the color of the defects and skin were highly similar, thus resulting in a detection error. The model failed to detect any object in Images 65 and 78. This could have been because the objects were too small in the images or poor lighting conditions. The training data included few pictures depicting dimly lit environments; thus, the model was not trained for detection under this condition. The error may have also occurred because the object did not present the defined tags, such as bearing fruit or fruit buds.

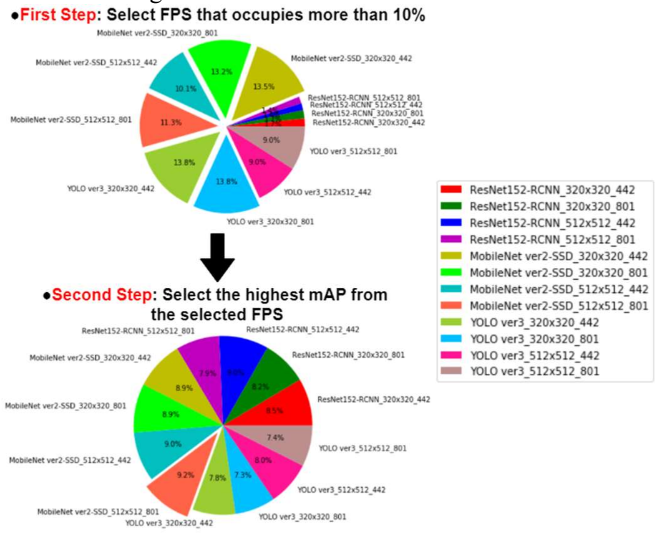


Fig. 27. Process Flow of Selecting the Final Model

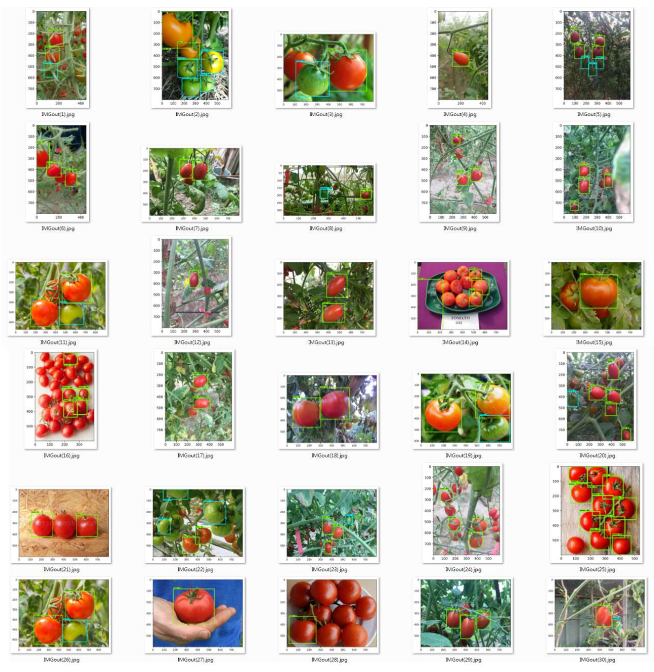


Fig. 28. Results of Testing Images (1–30)

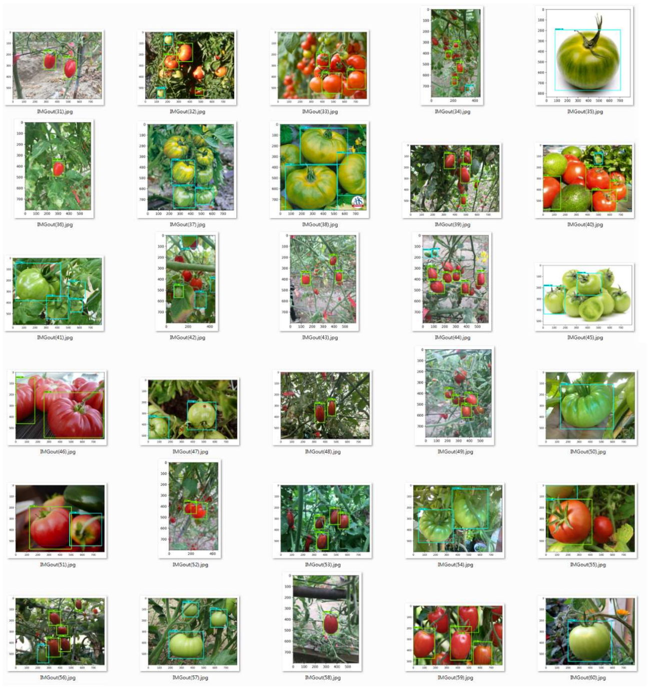


Fig. 29. Results of Testing Images (31–60)

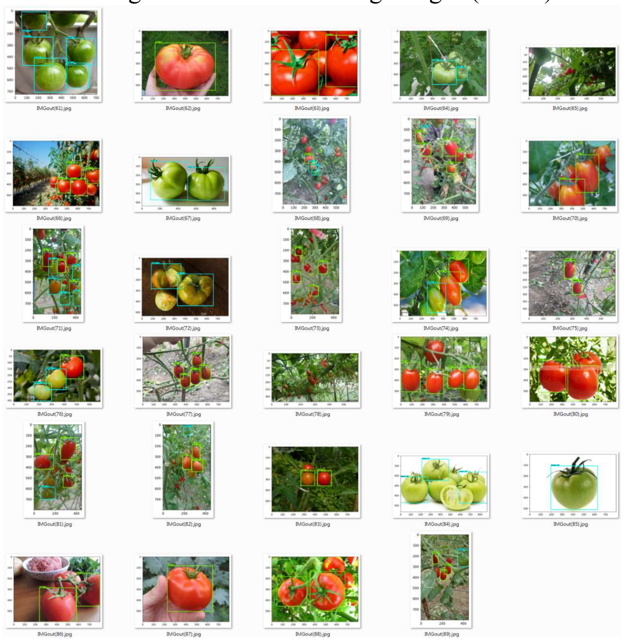


Fig. 30. Results of Testing Images (61–89)

B. *Training Results of the Arm Movement Prediction Model*

Because obtaining arm training data was cumbersome, this study followed the process shown in Fig. 16. First, we collected 100 data points and performed the first round of training. Next, the model was used for prediction. When a predicted angle was inaccurate, adjustments were made to the model and saved. To increase the amount of training data, the model settings were saved even when the predicted angle was correct. Finally, 230 data points were collected, and the weights of the first training model were used to adjust the model. The greatest advantage of this approach is that it saved a considerable amount of training time.

As presented in Fig. 31 and Fig. 32, two bar graphs depicted the AP of the ten models trained using k-fold cross-validation. The blue bar graph displays the results of using (2), which is the Keras custom model accuracy evaluation equation. The orange bar graph shows the results of using (3) to evaluate actual arm picking performance. The mean values of the blue equation were greater than those of the orange equation because the equations proposed in this study were rigorous. After first-round training, the blue bar graph of the ten models obtained through k-fold cross-validation showed an mAP value of 0.72, whereas the orange bar graph had an mAP value of 0.38. Marked increases were observed after inputting 230 data points and performing adjustments on the model; the mAP value of the blue bar graph became 0.78, whereas that of the orange bar graph became 0.49. Furthermore, the model with the highest blue bar graph did not perform the best in actual prediction (e.g., Iteration 10 in Fig. 31 and Iteration 5 in Fig. 32). Therefore, we used the proposed equation for computation and the model with the highest accuracy (e.g., Iteration 8 in Fig. 32 and Iteration 6 in Fig. 32).

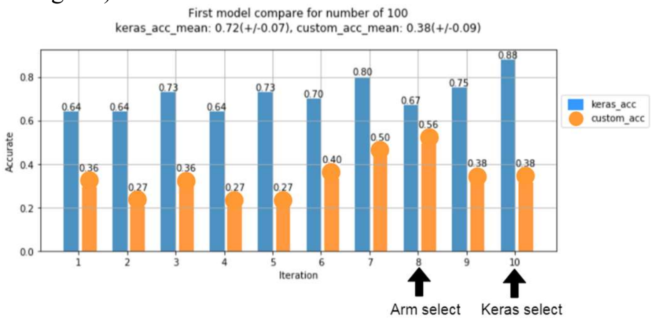


Fig. 31. AP values of Each Fold of 100 Data Points Obtained from the First-Round Training Model

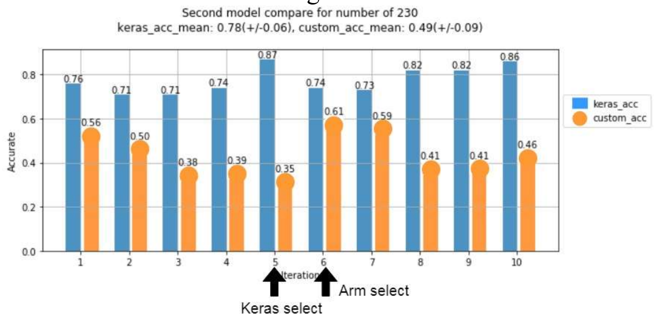


Fig. 32. AP values of Each Fold of 230 Data Points Obtained from the Adjusted Model

This study adopted k-fold cross-validation and used ten folds as an example for training. Hence, ten different models were obtained following training, and the model with the highest score for actual arm picking performance according to an evaluation using (3) was selected as the ultimate model for use. A total of 100 data points were used in the first round of training. Fig. 33 shows the eighth of ten rounds of training. The AP value obtained for the eighth round of training through the Keras custom model was approximately 0.67, whereas the highest AP obtained using the actual arm picking performance equation was approximately 0.56. The first round of training showed that the training loss values were greater than the validation loss values. This is because the training samples resembled the testing samples, and the insufficient rate of change yielded the present results. After 180 iterations, the training curve gradually normalized. However, because the model began to show signs of overfitting after 1000 iterations, its training was terminated; training accuracy and validation accuracy reached saturation at 1000 iterations.

A total of 230 data points were then used to adjust the previous model. Fig. 34 shows the results for the sixth of ten rounds of training. The AP value obtained for the sixth round of training through the Keras custom model was approximately 0.74, whereas the highest AP obtained through the actual arm picking performance equation was approximately 0.61. The results indicated a marked increase in AP of the adjusted model but also little change in training accuracy and validation accuracy values, which were maintained within a specific interval. Little change was also observed for training loss and validation loss. However, training was terminated beforehand because the model began to show signs of overfitting after 120 iterations.

The model accuracy did not change considerably after the second training (Figs. 33 and 34). This was due to two reasons. First, the data sets used in the first and second training processes were overly similar and insufficiently complex. To extend the durability of the robotic arm, we employed a data collection method that would not damage the hardware components. Consequently, the data sets were highly similar. Second, the training data were normalized to values between 0 and 1. Thus, the differences between the floating-point numbers were less noticeable. Nevertheless, the model accuracy was increased after the first and second training processes.

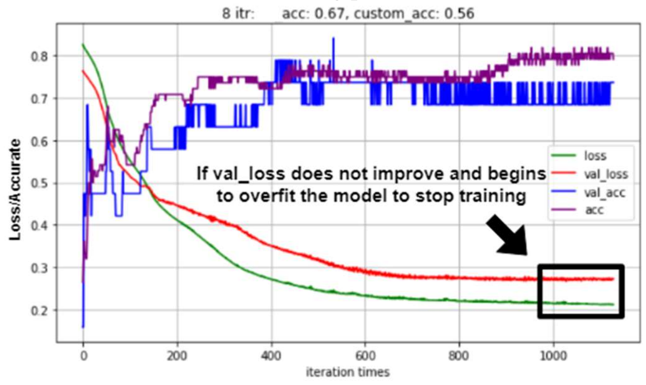


Fig. 33. First-Round Training Model Process of the Eighth Fold of the 100 Data Points

Fig. 35 and Fig. 36 present the schematics of the actual picking operation performed using the two trained models. The blue dots indicate objects that can be picked, and the red crosses indicate objects that cannot be picked. Fig. 35 shows that following the first round of training, the objects were mostly located on the upper right corner of the visible range, which suggested that the objects could not be picked. The figure also shows that the actual three angles did not differ greatly from the predicted angles; however, because angle B of the robotic arm was too large, the arm was too high to perform picking. The two locations on the lower left corner could not be picked because angle A exceeded the permissible range. The three locations on the lower right corner could not be picked because angles C and D were too small, rendering the objects too far for picking. An actual picking accuracy of 0.71 was obtained after testing 35 locations.

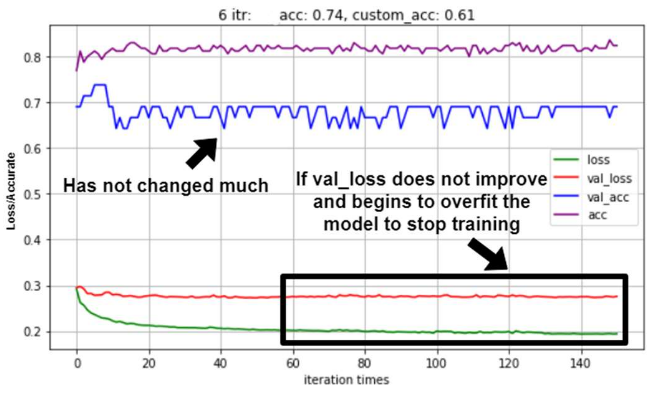


Fig. 34. Adjusted Training Model Process of the Sixth Fold of the 230 Data Points

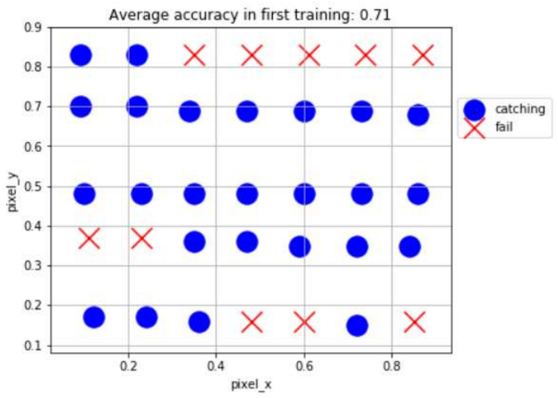


Fig. 35. Actual Picking Performed after the First-Round Training

Fig. 36 reveals marked improvement after the model was adjusted. Most of the objects in the upper right corner could be picked. No substantial change was observed for the locations on the lower left corner; angle A may have exceeded the permissible range because the data collected during the training stage may have been erroneous, leading to the current prediction errors. The two lowest locations may have been erroneous because angles C and D were too small; however, the one in the lower left corner was a new error and may have been the result of the new data influencing the weights of the previous model during model adjustment. This shows that care should be taken when collecting additional data; if the collection standards change, then the subsequent training results will be affected.

Although we used Keras to construct the arm movement prediction model, the model was finalized only after it had achieved acceptable accuracy levels following numerous adjustments. As shown in Fig. 37, all models underwent first-round training, as indicated by the blue bar graphs, and also model adjustment, as indicated by the orange bar graphs. At all stages, the mAPs of the ten models trained using k-fold cross-validation were used. Finally, the adjusted model was used to perform actual picking at 35 locations, obtaining the model's corresponding picking accuracy, as indicated by the green bar graph in Fig. 37.

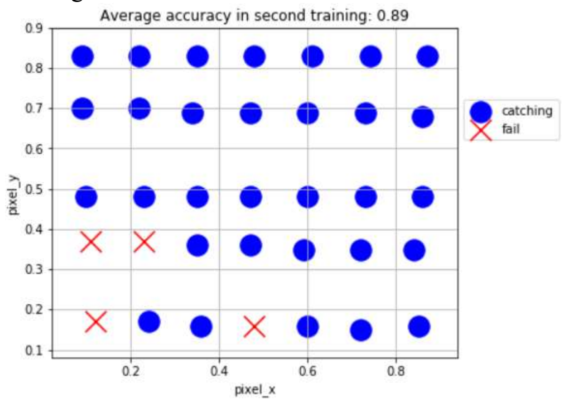


Fig. 36. Actual Picking Performed after Model Adjustment

A one-layered model with ten nodes was used; however, the training outcomes were not satisfactory. Even after adjustment, the model's mAP and picking accuracy were only 0.26 and 0.6, respectively. The mAP and picking accuracy markedly improved after increasing the number of nodes, suggesting that increasing the number of nodes improved model accuracy. We also tested the effects of increasing the number of hidden layers. Fig. 37 shows that although model accuracy improved, the rate of increase began to show signs of slowing when the number of hidden layers increased from two to three. Nonetheless, multilayer models had higher accuracy levels compared with single-layer models. Finally, a strategy of amplification followed by diminishing was adopted to construct the model. As shown in the right-hand side of Fig. 37, the highest values for mAP (0.49) and actual picking accuracy (0.89) were observed in the model with four hidden layers.

This study designed a webpage for user operations that displays live-streamed postdetection images. Users can then control robotic arms by referencing their schematics, which are displayed on the webpage, with a selection of buttons. The proposed system sends the detection results to the webpage, enabling users to ultimately make harvesting decisions. Users can also control the movement direction and speed of the mobile vehicle. To prevent the vehicle from being lost, the GPS module transmits the vehicle's geographical coordinates to the server every second after receiving satellite signals and performing positioning; the coordinates are then updated on a map.

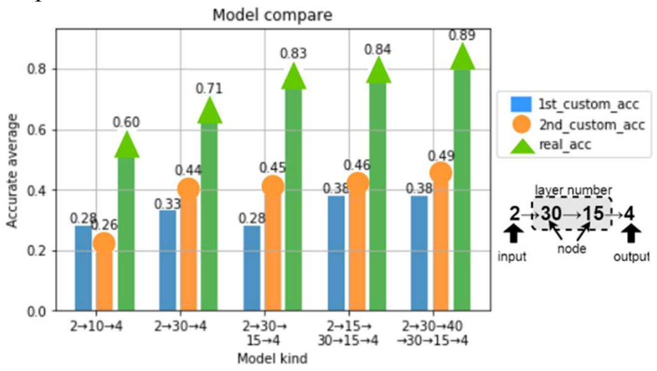


Fig. 37. Comparison of the mAP of the Various Models

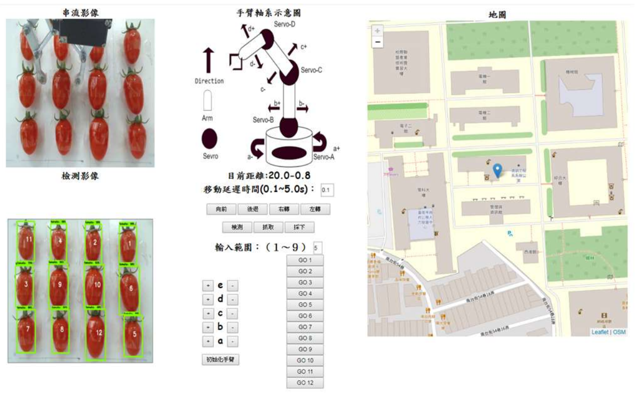


Fig. 38. Webpage Server and Object Detection Program Execution

V. Conclusions

This study used an object detection algorithm with IoT technology to develop a remote crop harvesting system. The proposed object detection model was a MobileNet SSD model; its mean accuracy of 84% was higher than that of other models. An arm movement prediction model that uses a four-hidden-layer perceptron model was also designed; its mean picking accuracy reached 89%.

This study makes the following four contributions. First, using this system, younger farmers can quickly acquire practical farming knowledge. Second, the results indicate that using deep learning on collected images to perform smart image recognition increases convenience and the number of potential applications. Third, machine learning was demonstrated to solve challenging problems encountered in multiaxial robotic arm movement design. Fourth, the developed system can be used by various populations and can provide different functions according to user needs. The simple design of the developed system also enables experienced older farmers to operate the system easily and provides useful and informed suggestions. For novice farmers, the system can enable them to quickly learn the trade and develop their career.

References

[1] Agricultural Statistics Visualized Query System, Council of Agriculture, Executive Yuan, Taiwan.

[2] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "ImageNet classification with deep convolutional neural networks", ACM Communications Magazine, Vol. 60, Issue 6, June 2017, pp. 84-90.

[3] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE, Vol. 86, Issue: 11, 1998.

[4] Matthew D., ZeilerRob Fergus, "Visualizing and Understanding Convolutional Networks", European Conference on Computer Vision (ECCV):Computer Vision, 2014, pp 818-833.

[5] Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, "Learning Local Feature Descriptors Using Convex Optimisation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 36, Issue:8, Aug. 2014, pp. 1573-1585.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, "Going deeper with convolutions", 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), DOI: 10.1109/CVPR.2015.7298594.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), DOI:10.1109/CVPR.2016.90.

[8] A. G. Howard et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Cornell University Library. 17 Apr 2017.

[9] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, DOI: 10.1109/CVPR.2018.00474.

[10] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 39, Issue: 6, June 2017, pp. 1137-1149.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", 2014 IEEE Conference on Computer Vision and Pattern Recognition.

[12] Ross Girshick, "Fast R-CNN", 2015 IEEE International Conference on Computer Vision (ICCV), pp. 1440-1448.

[13] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788.

[14] Joseph Redmon, Ali Farhadi, "YOLO9000: Better, Faster, Stronger", 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), DOI: 10.1109/CVPR.2017.690.

[15] J. Redmon, A. Farhadi. YOLOv3: An Incremental Improvement. Cornell University Library. 8 Apr 2018.

[16] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie, "Feature Pyramid Networks for Object Detection", 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), DOI:10.1109/CVPR.2017.106.

[17] Wei Liu, Dragomir, AnguelovDumitru, ErhanChristian, SzegedyScott, ReedCheng-Yang, FuAlexander C. Berg, "SSD: Single Shot MultiBox Detector", European Conference on Computer Vision (ECCV), Computer Vision 2016, pp. 21-37.

[18] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy, "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors", 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), DOI:10.1109/CVPR.2017.351.

[19] B. Kristyanto, B. B. Nugraha, A. K. Pamosoaji, K. A. Nugroho, "Analysis of human arm motions at assembly work as a basic of designing dual robot arm system", 2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), DOI:10.1109/IEEM.2017.8290106.

[20] Jiaxin Zhai, Gen Pan, Weixin Yan, Zhuang Fu, Yanzheng Zhao, "Dynamic analysis of a dual-arm humanoid cooking robot", 2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA), DOI:10.1109/ICIEA.2015.7334226.

[21] Pannawit Srisuk, Adna Sento, Yuttana Kitjaidure, "Inverse kinematics solution using neural networks from forward kinematics equations", 2017 9th International Conference on Knowledge and Smart Technology (KST), DOI:10.1109/KST.2017.7886084.

[22] Tzutalin. labelImg. GitHub. https://github.com/tzutalin/labelImg.

[23] Vitaly Bushaev. Adam-latest trends in deep learning optimization. Medium. Oct 22, 2018.https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

[24] jacksonliam. mjpg-streamer. GitHub. https://github.com/jacksonliam/mjpg-streamer.

[25] Diederik Paul Moeys, Chenghan Li, Julien N.P. Martel, Simeon Bamford, Luca Longinotti, Vasyl Motsnyi, David San Segundo Bello, Tobi Delbruck, "Color temporal contrast sensitivity in dynamic vision sensors", 2017 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 638.

[26] Wenqi Wu, Yingjie Yin, Xingang Wang, De Xu, "Face Detection With Different Scales Based on Faster R-CNN", IEEE Transactions on Cybernetics, Vol. 49, Issue: 11, 2019, pp. 4017-4028.

Gwo-Jiun Horng received an M.S. (2008) in electronic engineering from National Kaohsiung University of Applied Sciences, Kaohsiung, Taiwan. He is received the Ph.D. (2013) in Computer Science and Information Engineering at National Cheng Kung University, Taiwan. He is currently an associate professor in the Department of Computer Science and Information Engineering, Southern Taiwan University of Science and Technology, Tainan, Taiwan. His research interests include mobile service, internet of things, intelligent computing, cloud networks.

Min-Xiang Liu received the MS (2019) in Computer Science and Information Engineering, Southern Taiwan University of Science and Technology, Tainan, Taiwan. He has been working toward a PhD from the Institute of Manufacturing Information and Systems and the Department of Computer Science and Information Engineering, National Cheng-Kung University, Taiwan. His research interests include intelligent systems, internet of things, and artificial intelligence.

Chao-Chun Chen is an associate professor in the Institute of Manufacturing Information and Systems (IMIS) and the Department of Computer Science and Information Engineering (CSIE), National Cheng-Kung University, Taiwan. He received his PhD degree in the Department of Computer Science and Information Engineering at National Cheng-Kung University, Taiwan in 2004. His research interests include distributed data management, cloud manufacturing, artificial intelligence, system integration and optimization, and databases.