Novel Embedded Smart Gateway Framework for Fruit/Vegetable Quality Classification

Ming-Chih Chen, Yin-Ting Cheng,[*] and Chun-Yu Liu

Department of Electronic Engineering, National Kaohsiung University of Science and Technology (First Campus), No. 1, University Rd., Yanchao Dist., Kaohsiung City 82445, Taiwan

(Received May 12, 2021; accepted November 24, 2021)

Keywords: quality classification, artificial intelligence, edge computing, microprocessors

Recent advances in technology have increased the use of automation in agriculture. Commercially available equipment associated with fruit and vegetable quality classification or collection has a success rate of classification of around 50%. Even though a system can operate continuously, its low recognition rate causes misjudgments in fruit quality. In this work, we propose a new, embedded gateway structure with simple implementation and enhanced success rates for fruit and vegetable quality classification. We have combined an edge-computing embedded development platform with an artificial intelligence (AI) algorithm structure, where microprocessors are used to control the gateway switch of a conveyor, and this platform is used to build a small fruit/vegetable quality classification system, which has been implemented and tested. The system's hardware is controlled by different pulse widths. By combining AI algorithms of an image sensor for recognition, we have effectively enhanced the system's capability of fruit and vegetable quality recognition.

1. Introduction

According to a report estimating Taiwan's population from 2018 to 2065, the percentage of people over 65 years old will increase from 14.5% in 2018 to 19.9% in 2025. Moreover, by 2050, the percentage will increase to 38%. The rapid decrease in the young adult population is raising the average age of people employed in agriculture. The traditional manual fruit screening method relies heavily on human labor, has low efficiency, and places a strong physical demand on middle-aged workers.

Traditionally, fruit growers refer to the Fruit and Vegetable Quality Classification Standard and Packaging Specification Manual provided by the Agriculture and Food Agency of Taiwan for guidelines to manually grade fruit and vegetables before selling them at wholesale markets. In this study, we propose a novel embedded smart gateway framework that applies artificial intelligence (AI) and edge computing to fruit screening methods as an alternative to traditional manual screening. The proposed framework is expected to reduce the fruit error rate and quality disputes between fruit growers and consumers. It will also reduce the labor required for the fruit

[*]Corresponding author: e-mail: i107109118@nkust.edu.tw
https://doi.org/10.18494/SAM3552

harvesting process. The automatic screening system designed in this study uses an image sensor to recognize whether the fruit has serious pest infestation or sunburn and pick out damaged fruit to maintain the quality of fruit on the market.

2. Literature Review

Apte *et al.* applied the You Only Look Once (YOLO) algorithm in a mobile application in 2017.(1) It was able to process images acquired by a mobile device. They subsequently modified the YOLO feature extraction network model to enhance the inference speed. Iandola *et al.* applied a fire module layer to reduce the number of parameters in the YOLO algorithm and increase the image detection speed.(2) In 2018, Redmon and Farhadi proposed the YOLO-v3 algorithm, which was an improvement of the YOLO-v1 algorithm of Apte *et al.*(3) The YOLO-v3 algorithm was able to process 608 × 608 images on an NVIDIA Titan X graphics card at a speed of up to 20 frames per second.

Santad *et al.* proposed a system based on the YOLO algorithm to detect and analyze the relations between luggage and its owners.(4) The proposed system used camera images to enhance security surveillance.

In 2016, Liu *et al.* proposed a single-shot detector that combined object frame coordinates and classes in a single network architecture and used multiscale feature maps to obtain output predictions. The structure was implemented using a convolution neural network (CNN).(5) In the same year, Dubey discussed how to detect skin defects on fruit by creating a feature database of the external properties of fruit (color, size, shape, and texture).(6) The system first segmented the images and applied the speeded up robust features (SURF) algorithm to the segmented images. The features were then used to detect defects on the fruit.

Marimuthu and Roomi established a fuzzy model in 2017 to grade bananas into unripe, ripe, and over-ripe. The decision was determined by a framework with eight fuzzy rules created from a decision tree.(7) In the same year, Kamran and Pormah combined image processing and an artificial neural network to detect whether a cucumber's shape is ideal (cylindrical) or defective (curved or conical). A novel algorithm was implemented using MATLAB 2010a to preprocess and extract shape features from images.(8) However, these identification and classification systems are unsuitable for fruit classification since they can only perform simple classifications for the same type of fruit.(9,10)

In 2018, Khaing *et al.* proposed a CNN based on the back-propagation algorithm and applied it to fruit classification, as well as a vision-based automatic classification system.(11,12) In the same year, Hossain *et al.* proposed a vision-based fruit classification framework using deep learning. The framework also used a CNN, where a finely tuned VGG-16 model showed outstanding accuracy.(13) Lu *et al.* used a six-layer CNN composed of convolutional, pooling, and fully connected layers for fruit classification. The system obtained an accuracy of 91.44%, which was higher than that of algorithms such as the voting-based support vector machine, wavelet entropy, and genetic algorithm. However, increasing the number of types of fruit decreased the accuracy.(14)

Bochinski *et al.* proposed using the intersection over union (IOU) tracking algorithm for high-speed tracking.(15) The tracking method performed IOU calculations at the target position detected in the original image and the target position in the next frame. If the value of the IOU is higher than the established threshold value, the regions are considered as related and determined as the same object. This method can effectively track the same object. Chen *et al.* proposed the use of an embedded imaging system and AI to detect fruit quality. The accuracy rate was as high as 88% in experiments, thus demonstrating effective detection.(16)

3. Structure of System

3.1 System overview

Our newly proposed system with the embedded smart gateway framework was developed to satisfy the following research and development objectives:

A. Its automatic gateway control system is capable of classifying the fruit quality.
B. The neural network model of the system is embedded in a development board to perform model inference.
C. The system has a graphical interface to display the delivery of the fruit in real-time.
D. The system is capable of using the graphical interface to collect data and evaluate the neural network model.
E. The system has a low fabrication cost.

The model combines AI and gateways for intelligent decision-making and classification. Fruit and vegetable image detection is performed through a camera sensor module. The AI model uses the detection data to determine the gateways' actions. The Keras model framework is used for the classification and evaluated the computation time of building the model. Keras is an open-source software library that provides a Python interface for artificial neural networks. Data collection was performed to obtain the data required for the neural network model, and an experiment was conducted using the collected data. Moreover, the graphical user interface of the system integrates the functions used for training the model. In summary, an effective procedure is proposed in this work to develop the fruit classification system.

The model is trained to search for functions that fit the features in the data. After evaluating the completed model, the model is integrated into the system architecture. The YOLO algorithm is implemented on a Jetson TX2 embedded platform to perform object detection, circle the fruit image, and perform image recognition using the CNN algorithm. On the basis of the recognition results, different pulse width modulation (PWM) signals are sent by an STM32 microcontroller development board to control the gateway automatically.

3.2 Hardware architecture

The system includes three pieces of essential hardware: a Jetson TX2 embedded platform development board, an STM32 microcontroller development board, and an MG996R servomotor.

Figure 1 shows the hardware architecture of the system. First, the operator turns on the front gateway control switch. Then the front gateway delivers the fruit to the conveyor platform. Next, a camera on the Jetson TX2 development board is used to capture images to perform tracking and recognize fruit images. Finally, the recognition results are sent by automatically detecting whether the fruit is ready to leave the conveyor platform. Since there is no PWM or transistor–transistor logic (TTL) for the RS232 UART output module on the Jetson TX2 board, the UART signals are sent to the STM32 microprocessor module through an FT232 Bluetooth module (USB to TTL). Different PWM signals are sent to control the direction of the MG996R servomotor and to sort the fruit on the conveyor platform according to the received string.

The computer is used to train the neural network model in advance, and the model is embedded in the Jetson TX2 development board. After transferring the images from the camera on the development board to the neural network model, the FT232 module sends the inference results to the STM32 microcontroller development board and the PWM signals to the MG996R servomotor.

3.3 System of operation process

Figure 2 shows a flow chart of the system operation. Object detection is used to detect whether or not the fruit is delivered to the platform. The tracking and recognition function continues to work before the fruit leaves the conveyor platform during the delivery process and saves all the tracking results. When the fruit leaves the conveyor platform, all the recognition results are saved as the final answer to determine the quality of the fruit, and the gateway channel control is activated.

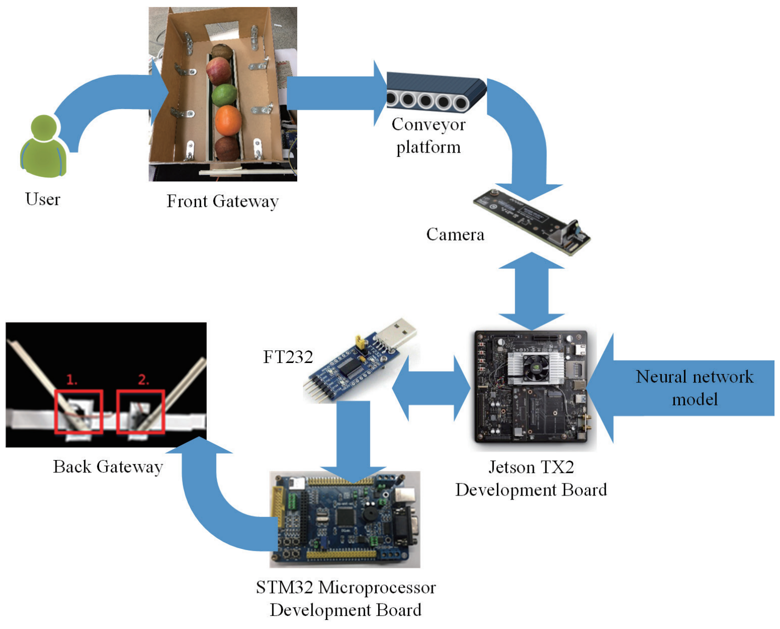


Fig. 1. (Color online) System hardware architecture.

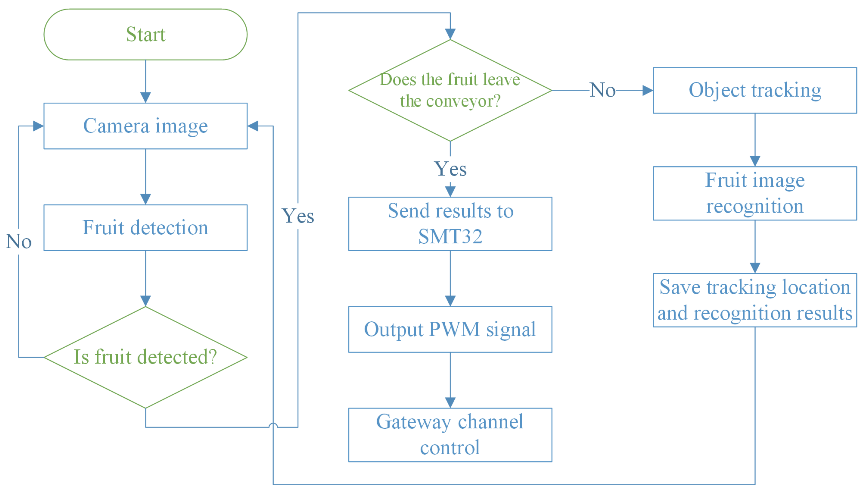


Fig. 2. (Color online) System operation process.

4. Systematic Procedures

4.1 Front gateway

Figure 3 shows a flow chart of the control of the front gateway. Two servomotors take turns to perform lifting and lowering movements. When servomotor 2 for the gateway is lifted and released to pass the first fruit to the conveyor platform, servomotor 1 for the gateway is lowered so that the second fruit rolls over to the front row to the release position. At this moment, servomotor 2 for the gateway is lowered and servomotor 1 for the gateway is lifted to make a buffer for the momentum of the subsequent fruit. By continuing this action, fruits are consecutively rolled onto the conveyor platform.

4.2 Tiny-YOLO

Because the Tiny-YOLO algorithm is a network architecture for feature extraction provided by the YOLO algorithm, the Tiny-YOLO algorithm has a lighter neural network architecture design than YOLO. The tiny-YOLO's architecture is composed of a CNN and max-pooling. A batch normalization (BN) layer is added after each convolution layer to normalize the parameters, help the model learn, and accelerate training. A leaky rectified linear unit (ReLU) is used as the activation function. Figure 4 shows the Tiny-YOLO architecture, from which we can see that the sizes of the filters are increased as powers of 2, which allows the input dimension to be reduced using the max-pooling layer. In addition, a 1×1 size convolution kernel is used at the end of the architecture for training. Since the Tiny-YOLO algorithm uses a two-layer feature pyramid network (FPN), the dimensions of the output layer are configured as output tensors with sizes of 13×13 and 26×26 to perform object detection.

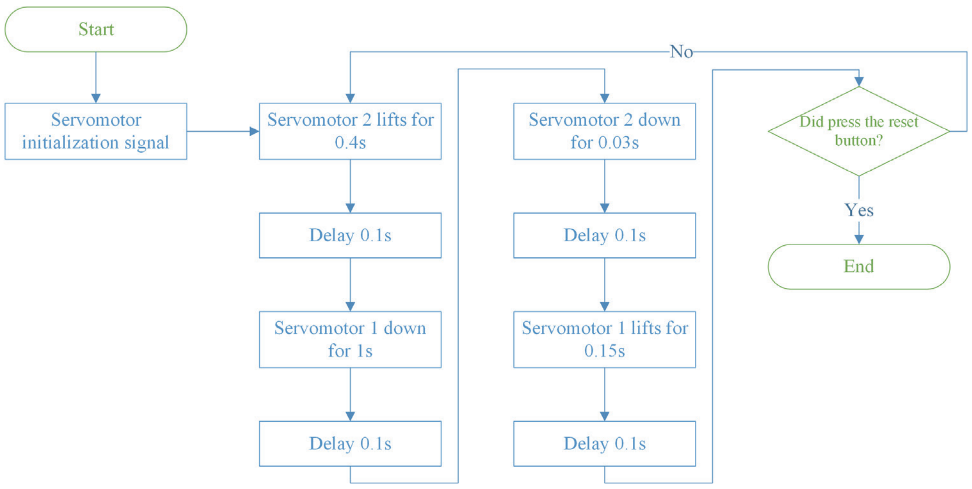
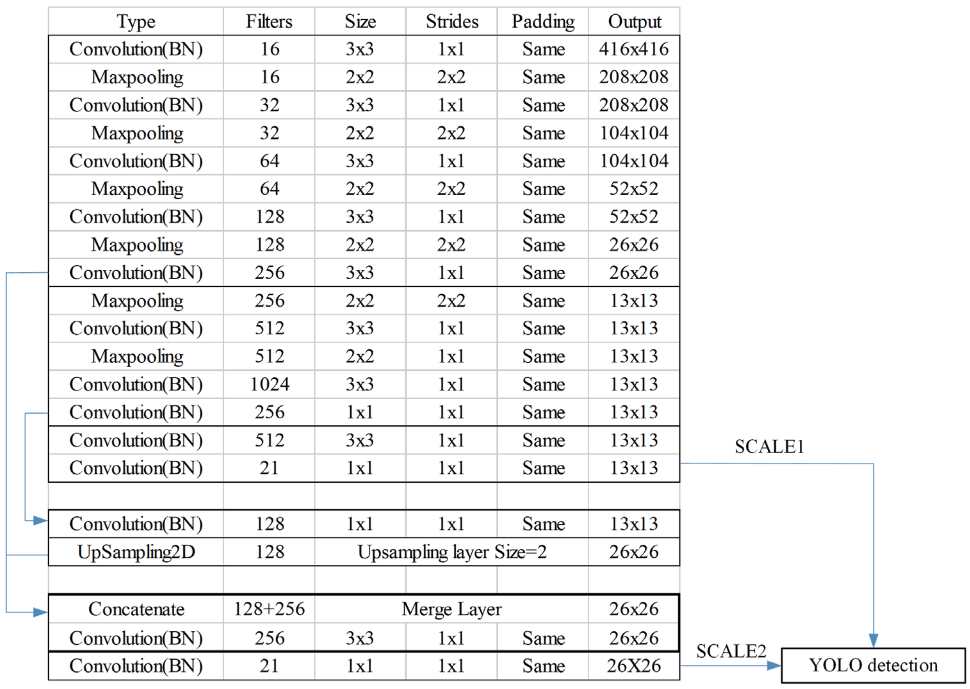

Start
Servomotor initialization signal
Servomotor 2 lifts for 0.4s
Delay 0.1s
Servomotor 1 down for 1s
Delay 0.1s
Servomotor 2 down for 0.03s
Delay 0.1s
Servomotor 1 lifts for 0.15s
Delay 0.1s
No
Did press the reset button?
Yes
End

Fig. 3. (Color online) Front gateway control process.



Type	Filters	Size	Strides	Padding	Output
Convolution(BN)	16	3x3	1x1	Same	416x416
Maxpooling	16	2x2	2x2	Same	208x208
Convolution(BN)	32	3x3	1x1	Same	208x208
Maxpooling	32	2x2	2x2	Same	104x104
Convolution(BN)	64	3x3	1x1	Same	104x104
Maxpooling	64	2x2	2x2	Same	52x52
Convolution(BN)	128	3x3	1x1	Same	52x52
Maxpooling	128	2x2	2x2	Same	26x26
Convolution(BN)	256	3x3	1x1	Same	26x26
Maxpooling	256	2x2	2x2	Same	13x13
Convolution(BN)	512	3x3	1x1	Same	13x13
Maxpooling	512	2x2	1x1	Same	13x13
Convolution(BN)	1024	3x3	1x1	Same	13x13
Convolution(BN)	256	1x1	1x1	Same	13x13
Convolution(BN)	512	3x3	1x1	Same	13x13
Convolution(BN)	21	1x1	1x1	Same	13x13
Convolution(BN)	128	1x1	1x1	Same	13x13
UpSampling2D	128	Upsampling layer Size=2			26x26
Concatenate	128+256	Merge Layer			26x26
Convolution(BN)	256	3x3	1x1	Same	26x26
Convolution(BN)	21	1x1	1x1	Same	26X26

Fig. 4. (Color online) Tiny-YOLO architecture.

4.3 Uff model

The fruit classification recognition model (Filename Extension.h5) built by Keras is converted to a model file (Filename Extension.uff) available in TensorRT through the software developed by NVIDIA TensorRT. Furthermore, model inference is performed on the converted model file using C++.

As shown in Table 1, the original Keras model is written in Python, whereas the TensorRT model is written in C++. It can be seen that the use of C++ and the uff model enhances the

Table 1
Comparison of model speeds.

	Keras model	TensorRT model	Enhancement of speed
Programming language	Python	C++	
Speed of inference model	4.6–8.2 ms	1.8–3 ms	2.5–2.7

inference speed by 2.5–2.7 times. The enhancement will be even greater if a deeper network architecture is used.

4.4 Back gateway

Figure 5 shows the control of the gate that waits to receive the final fruit identification results. Since the system is divided into three channels, it accepts three different codes and sends out three types of PWM signals. After the gate is turned, it is set to the center position before the next motor turns.

4.5 Object detection model

The architecture of the feature extraction network is modified for the YOLO algorithm. Therefore, the architectures in Table 2 were designed for comparison in the experiment.

Note that the system uses the FPN framework to output coarse and fine features for the regression prediction of object locations. Therefore, the output dimension must be set to the output tensor of the corresponding category.

5. Experimental Results

We use the YOLO algorithm to experimentally evaluate the model, and two parameters are set for the YOLO object detection: (1) non-maximum suppression (NMS)–IOU is set to 0.45 and (2) object confidence is set to 0.3 to make the system work more smoothly and achieve better accuracy. Two indices are used to perform the evaluation, as discussed in this section. An example is first used to explain the definition of the index used for evaluation, which is followed by the displayed experiment data.

5.1 Precision–recall (PR) curve

The IOU must be calculated before drawing a PR curve. In Eq. (1) and Fig. 6, A is the real annotated box and B is the predicted object box. The IOU represents the rate of overlap between the real object box and the predicted object box. Larger IOU values indicate better object detection capabilities.

$$IOU = \left(area\left(A \cap B\right)\right) / \left(area\left(A \cup B\right)\right) \tag{1}$$

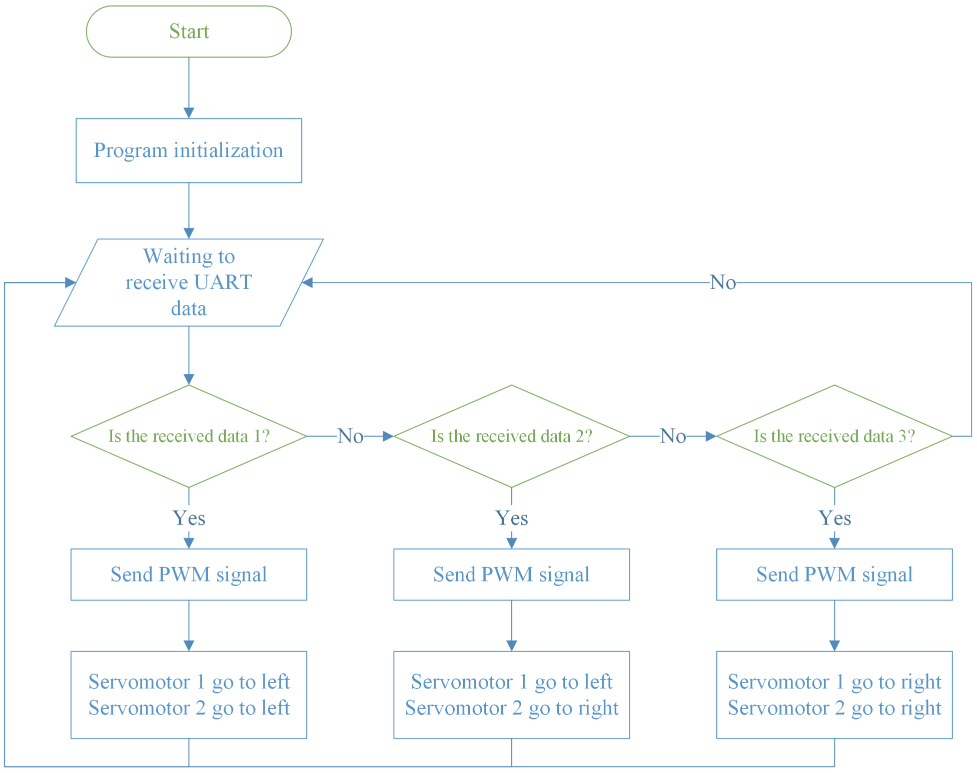


Fig. 5. (Color online) Back gateway control process.

Table 2
Architectures used for comparison.

Architectures	Small-Mobilenet-v1
	Small-Mobilenet-v2
	Tiny-Batch-Dropout
	Tiny-Nobatch-Have-Leaky
	Tiny-SqueezeNet
	Tiny-Octave

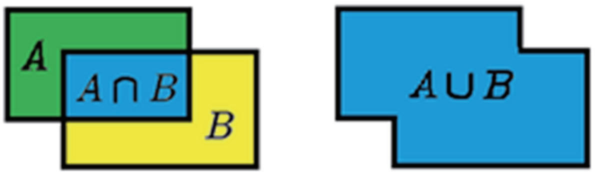


Fig. 6. (Color online) Difference and intersection.

We refer to the model proposed by Padilla *et al.* for object prediction.(17) A graph is plotted after ranking the object's confidence levels (confidences) according to the model's predictions of the object type. The PR curve is plotted by calculating the precision and recall values of true positive (TP) and accumulated TP (Acc TP) against the values of false positive (FP) and accumulated FP (Acc FP) by calculating the accumulated TP or FP.

Figures 7–13 show the PR curves for the neural model architectures used for comparison and the proposed architecture. In the figures, FRUIT AP represents the PR curve of good fruit and B_ FRUIT AP represents the PR curve of bad fruit. In Figs. 7–9, the precision value decreases gradually as the recall value increases. The recall value for the Tiny-Batch-Dropout architecture reaches a very high level, indicating that all boxes containing the objects are circled, and it correctly determines the quality of fruit for the circled object. The recall values for the Small-

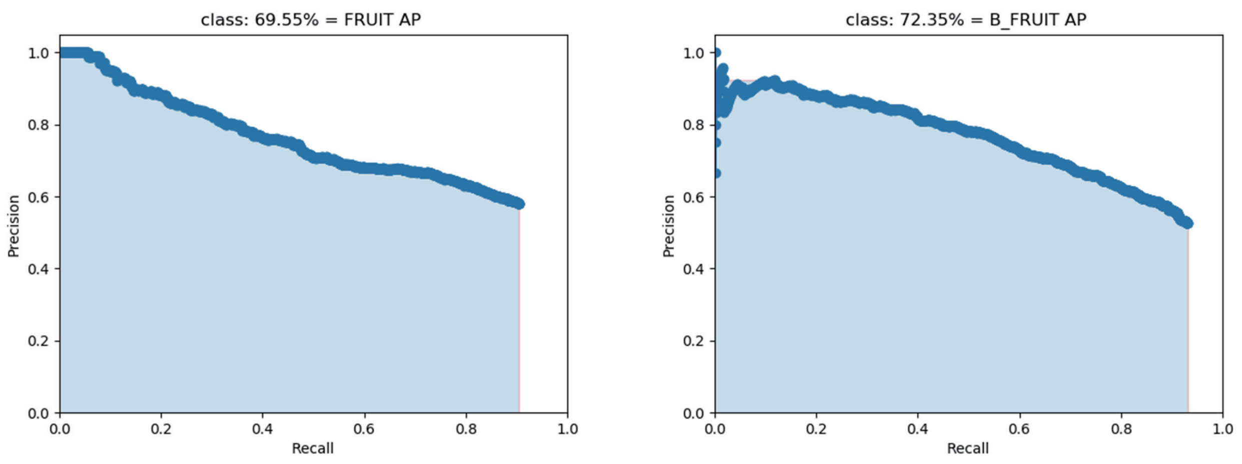


Fig. 7. (Color online) Small-Mobilenet-v1 PR curve (single test).

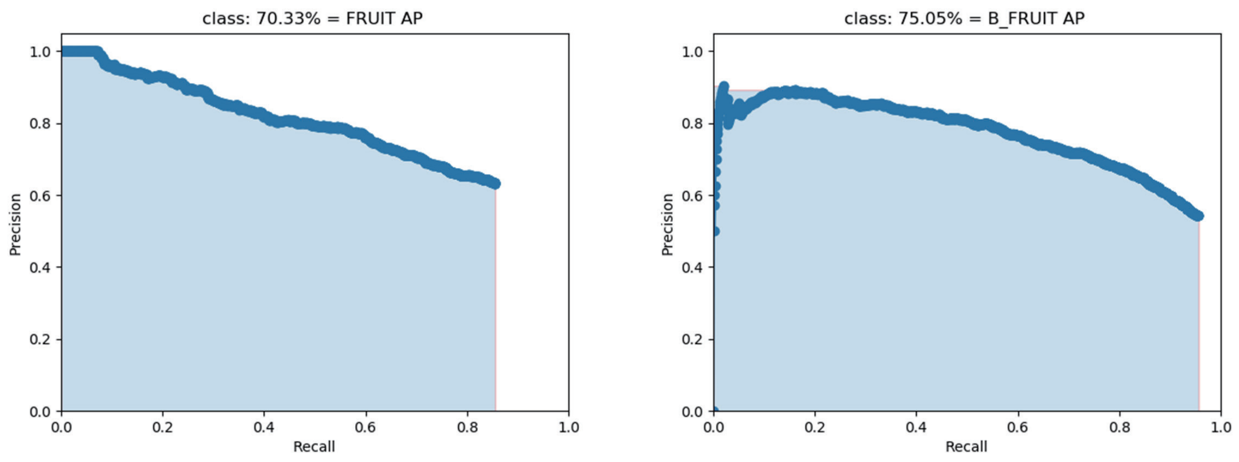


Fig. 8. (Color online) Small-Mobilenet-v2 PR curve (single test).

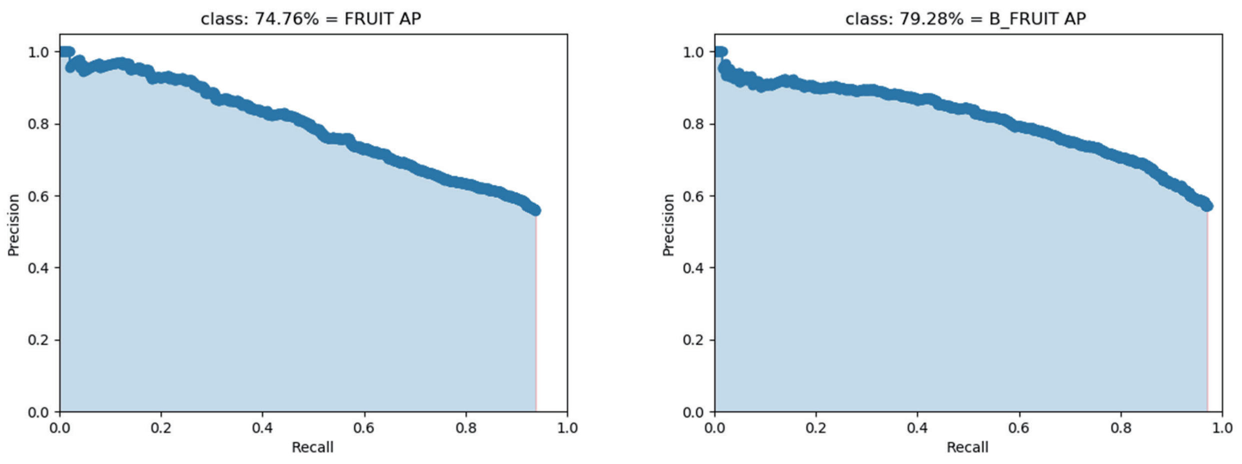


Fig. 9. (Color online) Tiny-Batch-Dropout PR curve (single test).

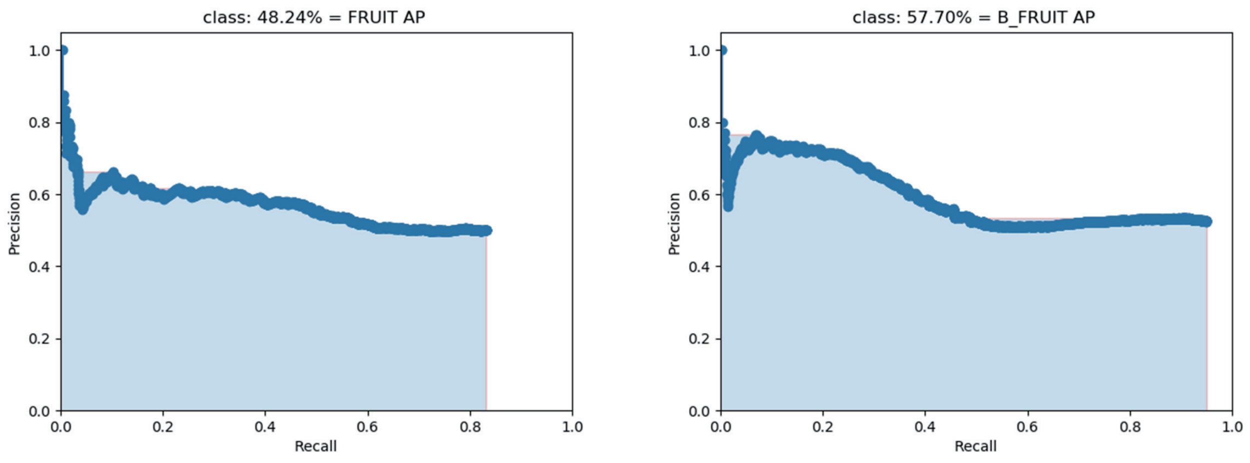


Fig. 10. (Color online) Tiny-Nobatch-Have-Leaky PR curve (single test).

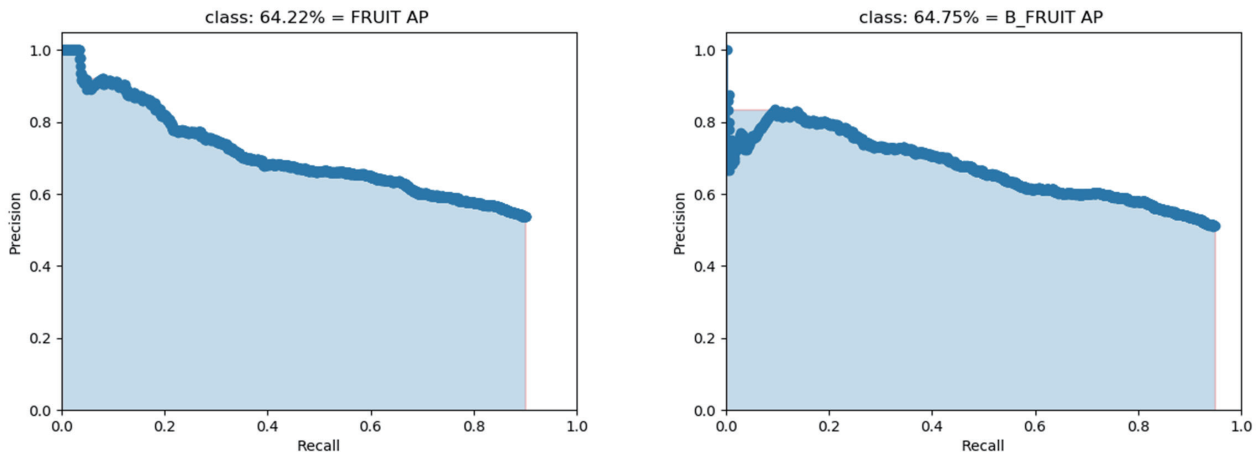


Fig. 11. (Color online) Tiny-Octave PR curve (single test).

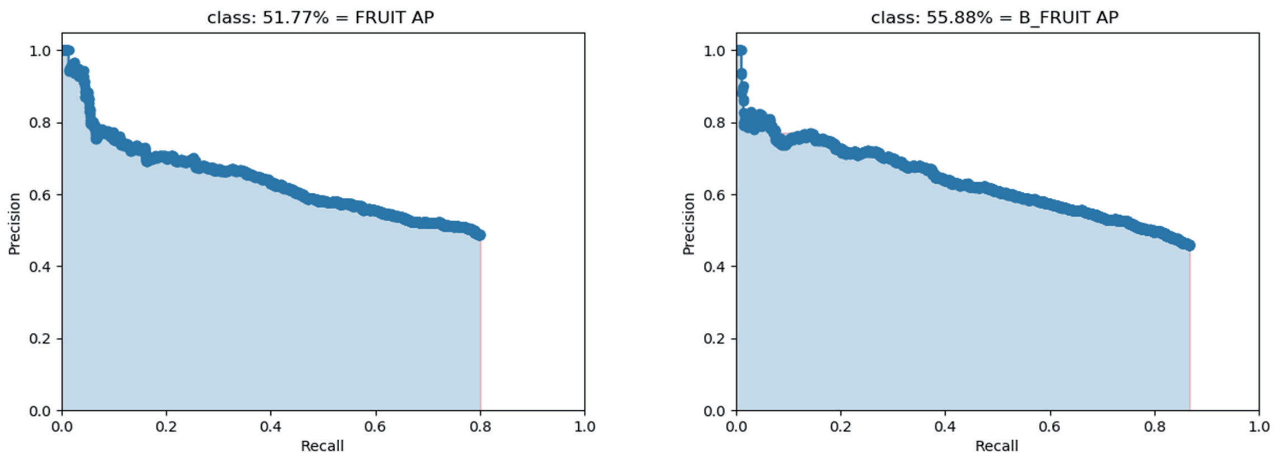


Fig. 12. (Color online) Tiny-SqueezeNet PR curve (single test).

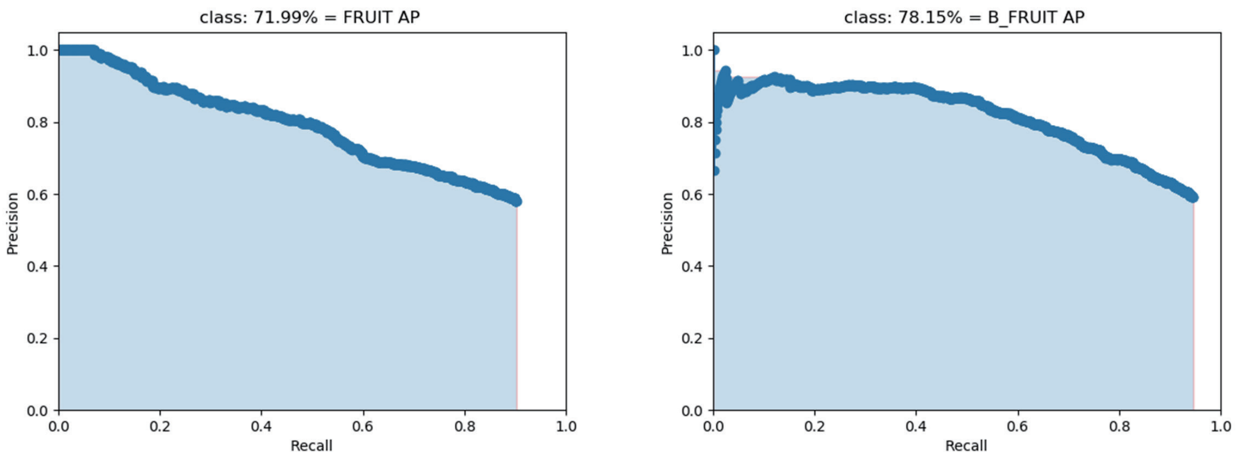


Fig. 13. (Color online) Tiny-YOLO PR curve (single test).

Mobilenet-v1 and Small-Mobilenet-v2 architectures are not significantly different and these architectures have average performance.

Figure 10 shows the PR curves for the Tiny-Nobatch-Have-Leaky architecture without the BN layer. The results of the judgment box are incorrect, and the performance is very poor even though the recall values can be improved.

In Fig. 11, the PR curve for the Tiny-Octave architecture is based on the accuracy of detecting fruit quality. Since the architecture uses fewer parameters than the Small-Mobilenet-v2 and Small-Mobilenet-v1 architectures, the PR curves are rough with a downward slope. As a result, the performance of the model is poor and the accuracy is reduced by 10% compared with the two architectures.

Figure 12 shows the PR curve for the Tiny-SqueezeNet architecture. Regardless of the quality of the fruit, there is a gradual decrease in precision as the recall increases, and the area under the curve (AUC) is quite small, indicating the poor performance of the model.

Figure 13 shows the PR curve for the proposed Tiny-YOLO architecture. The curve for the good fruit is relatively smooth, which means that the model performs satisfactorily, and the overall accuracy is relatively high compared with that of the Tiny-SqueezeNet architecture. When the recall increases to 90%, the precision is still around 60%.

5.2 Mean average precision (mAP)

The mAP is calculated by summing the average precision (AP) for each class and dividing by the number of classes. In this work, the AP values are calculated by calculating the maximum precision corresponding to each value of the recall, similar to the AUC for each type of precision–recall curve. The precision–recall curve is plotted by calculating the precision and recall values of the accumulated TP or FP detections.

Table 3 shows the mAP for various architectures of neural networks. An evaluation model was used with the same training dataset and test set to compare the mAP of training and testing

Table 3
Comparison of mAP for different architectures.

	Result 1	Result 2	Result 3	Result 4	Result 5	Result 6
Small-Mobilenet-v1	49.26	80.83	18.73	35.52	44.30	70.95
Small-Mobilenet-v2	51.96	83.32	18.92	34.09	46.69	72.69
Tiny-Batch-Dropout	54.47	84.27	20.49	32.82	48.17	77.02
Tiny-Nobatch-Have-Leaky	32.34	51.91	10.20	20.84	29.96	52.97
Tiny-SqueezeNet	33.48	57.21	10.29	22.35	32.26	53.83
Tiny-Octave	37.41	65.46	15.21	31.07	37.67	64.48
Tiny-YOLO	61.53	85.59	22.94	36.36	52.88	75.07

for a single subject with the mAP of multiple subjects. For the Small-Mobilenet-v2 and Small-Mobilenet-v1 architectures, both training sets have a good mAP in general. The Small-Mobilenet-v2 architecture has an accuracy improvement of 1.71% for a single fruit. However, there is no noticeable enhancement compared with the Tiny-YOLO architecture.

The results of testing performed on the Jetson TX2 Developer Edition in terms of the number of parameters required for different architectures and the detection speed are given in Table 4. Most of the models have close to 50 layers. However, the Small-Mobilenet-v2 architecture has as many as 70 layers since it uses the Add layer twice. Both the Small-Mobilenet-v1 and Small-Mobilenet-v2 architectures have a reduced number of parameters owing to the use of a 1×1 filter compared with the use of no filter. The number of parameters for the Tiny-SqueezeNet architecture is 19.2 times less than that for the Tiny-YOLO architecture, and the computation time is reduced by over 0.02 s. Lastly, the performance of the Tiny-Octave architecture is typical. It has a reasonable number of parameters and an average operating speed.

5.3 Model training

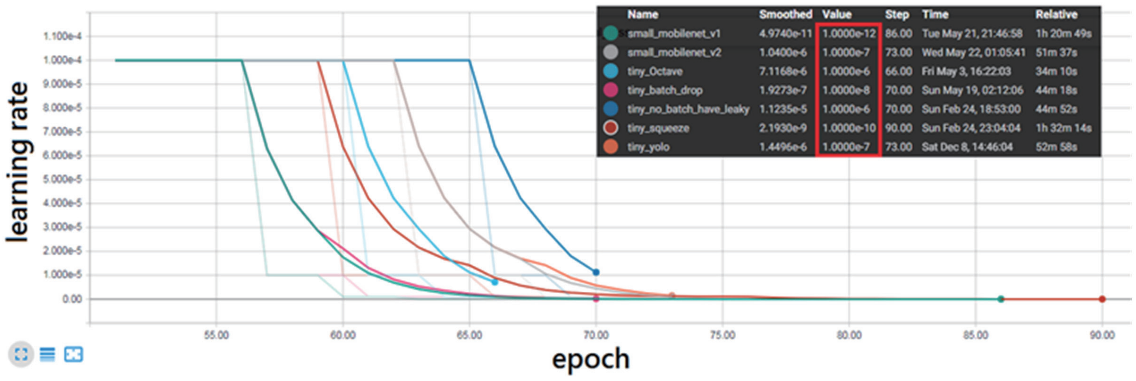
The NVIDIA GTX1070 Ti training model is used for comparison of the architectures with the batch size set to 16. Also, the number of epochs is set to 100, where the first 50 epochs are set with a fixed learning rate and the first 20 network layers of the model are set to fixed weights. For the subsequent 50 epochs, all network layers are set to be trainable, and the learning rate is varied dynamically to gradually adjust to a lower learning rate. The initial loss value is set at a rate of 10^{-3} units, and training is stopped if the new loss value is not lower than the original loss value.

Figure 14 shows the changes in the learning rates of the models during training. The learning rates are adjusted in the subsequent 50 training sessions according to changes in the value of the loss function. The benefit of adjusting the learning rates is that the best local parameter is still searched for when there is no escape from the local optimization parameters. The Small-Mobilenet-v1 architecture uses a lower learning rate at the beginning and extends the number of training epochs to 86. On the other hand, the Tiny-Octave architecture is ineffective in adjusting the learning rate to reduce the loss function value.

Figure 15 shows the loss function graphs for the training set of each model. The graphs of all models slowly flatten out without showing significant fluctuations in the learning loss function.

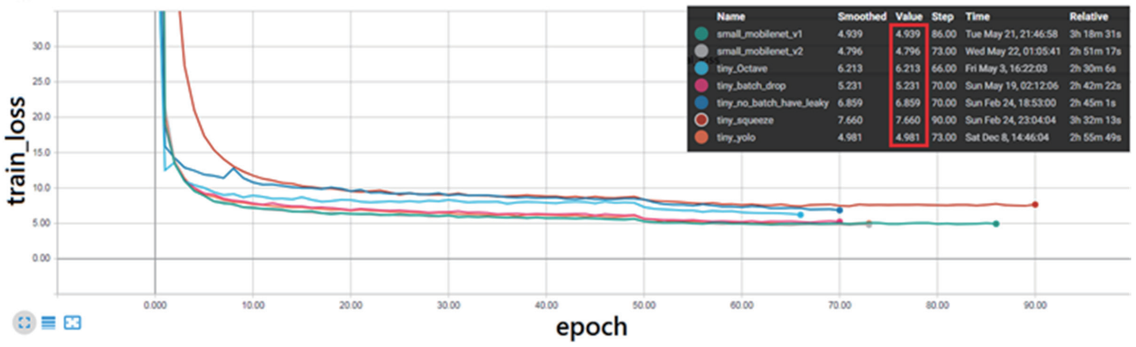
Table 4
Comparison of parameters under different architectures.

	Parameter (MB)	Network layers	Training time (min)	Detection speed (s)
Small-Mobilenet v1	3.58	58	198	0.0784
Small-Mobilenet v2	4.47	70	171	0.1058
Tiny-Batch-Dropout	8.67	55	162	0.0916
Tiny-Nobatch-Have-Leaky	8.66	35	165	0.0784
Tiny-Octave	1.78	51	150	0.0785
Tiny-SqueezeNet	0.45	50	212	0.0578
Tiny-YOLO	8.67	44	175	0.0860



	Name	Smoothed	Value	Step	Time	Relative
●	small_mobilenet_v1	4.9740e-11	1.0000e-12	86.00	Tue May 21, 21:46:58	1h 20m 49s
●	small_mobilenet_v2	1.0400e-6	1.0000e-7	73.00	Wed May 22, 01:05:41	51m 37s
●	tiny_Octave	7.1168e-6	1.0000e-6	66.00	Fri May 3, 16:22:03	34m 10s
●	tiny_batch_drop	1.9273e-7	1.0000e-8	70.00	Sun May 19, 02:12:06	44m 18s
●	tiny_no_batch_have_leaky	1.1235e-5	1.0000e-6	70.00	Sun Feb 24, 18:53:00	44m 52s
●	tiny_squeeze	2.1930e-9	1.0000e-10	90.00	Sun Feb 24, 23:04:04	1h 32m 14s
●	tiny_yolo	1.4496e-6	1.0000e-7	73.00	Sat Dec 8, 14:46:04	52m 58s

Fig. 14. (Color online) Adjustment of the learning rate.

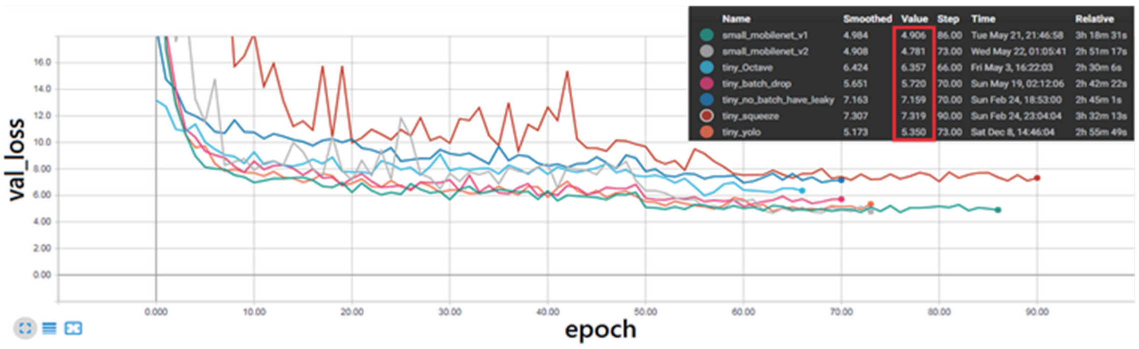


	Name	Smoothed	Value	Step	Time	Relative
●	small_mobilenet_v1	4.939	4.939	86.00	Tue May 21, 21:46:58	3h 18m 31s
●	small_mobilenet_v2	4.796	4.796	73.00	Wed May 22, 01:05:41	2h 51m 17s
●	tiny_Octave	6.213	6.213	66.00	Fri May 3, 16:22:03	2h 30m 6s
●	tiny_batch_drop	5.231	5.231	70.00	Sun May 19, 02:12:06	2h 42m 22s
●	tiny_no_batch_have_leaky	6.859	6.859	70.00	Sun Feb 24, 18:53:00	2h 45m 1s
●	tiny_squeeze	7.660	7.660	90.00	Sun Feb 24, 23:04:04	3h 32m 13s
●	tiny_yolo	4.981	4.981	73.00	Sat Dec 8, 14:46:04	2h 55m 49s

Fig. 15. (Color online) Model loss function graphs (training).

However, the loss function of the Tiny-SqueezeNet architecture is the highest and its average accuracy is relatively poor. Similarly, the Small-Mobilenet-v1 architecture also requires a long training time. On the other hand, both the Small-Mobilenet-v2 and Tiny-YOLO architectures have relatively low loss function values.

Figure 16 shows the loss function graphs for the validation set of each model. The loss values of all models tend to converge. Nevertheless, fluctuations occur for the Tiny-Nobatch-Have-Leaky architecture. However, the model training becomes unstable without using the BN layer and the loss function is relatively high. The loss functions of the Small-Mobilenet-v1, Small-



	Name	Smoothed	Value	Step	Time	Relative
●	small_mobilenet_v1	4.984	4.906	86.00	Tue May 21, 21:46:58	3h 18m 31s
●	small_mobilenet_v2	4.908	4.781	73.00	Wed May 22, 01:05:41	2h 51m 17s
●	tiny_Octave	6.424	6.357	66.00	Fri May 3, 16:22:03	2h 30m 6s
●	tiny_batch_drop	5.651	5.720	70.00	Sun May 19, 02:12:06	2h 42m 22s
●	tiny_no_batch_have_leaky	7.163	7.159	70.00	Sun Feb 24, 18:53:00	2h 45m 1s
●	tiny_squeeze	7.307	7.319	90.00	Sun Feb 24, 23:04:04	3h 32m 13s
●	tiny_yolo	5.173	5.350	73.00	Sat Dec 8, 14:46:04	2h 55m 49s

Fig. 16. (Color online) Model loss function graphs (validation).

Mobilenet-v2, and Tiny-YOLO architectures have relatively similar mAP values at the end of the validation.

6. Conclusion

In this work, we proposed a novel embedded smart gateway framework that combines the YOLO-v3 algorithm and the Tiny-YOLO neural network model. We performed object detection and compared the proposed framework with several other models in terms of classification performances. Furthermore, we built a model architecture to develop the proposed system using TensorFlow and the Keras framework. We also acquired all images used in this work. The proposed Tiny-YOLO architecture achieves an mAP of 75% for object detection.

Overall, we implemented a vision-based application on an embedded system to realize an automatic screening system for fruit on a conveyor platform. In the future, the proposed model can be adapted for a field-programmable gate array board by loading only a single classification model for a single type of fruit.

References

1 M. Apte, S. Mangat, and P. Sekhar: Stanford University CS class CS231n: Convolutional Neural Networks for Visual Recognition (2017). http://cs231n.stanford.edu/reports/2017/pdfs/135.pdf
2 F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer: Comput. Sci. (2016). https://arxiv.org/abs/1602.07360
3 J. Redmon and A. Farhadi: Comput. Sci. (2018). https://arxiv.org/abs/1804.02767
4 T. Santad, P. Silapasupphakornwong, W. Choensawat, and K. Sookhanaphibarn: Proc. IEEE Global Conf. Consumer Electronics (IEEE, 2018) 157. https://doi.org/10.1109/GCCE.2018.8574819
5 W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg: Proc. European Conf. Computer Vision (Springer, 2016) 21. http://doi.org/10.1007/978-3-319-46448-0_2
6 Yogesh and A. K. Dubey: Proc. Int. Conf. Reliability, Infocom Technologies and Optimization (IEEE, 2016) 590. http://doi.org/10.1109/ICRITO.2016.7785023
7 S. Marimuthu and S. M. M. Roomi: IEEE Sens. J. **17** (2017) 4903. https://doi.org/10.1109/JSEN.2017.2715222
8 K. Kamran and A. Pormah: J. Food Process Eng. **40** (2017) e12558. https://doi.org/10.1111/jfpe.12558
9 Nandi, C. Sekhar, B. Tudu, and C. Koley: IEEE Trans. Instrum. Meas. **63** (2014) 1722. https://doi.org/10.1109/TIM.2014.2299527
10 M. Ghulam: Eng. Appl. Artif. Intell. **37** (2015) 361. https://doi.org/10.1016/j.engappai.2014.10.001
11 Z. M. Khaing, Y. Naung, and P. H. Htut: Proc. IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conf. (IEEE, 2018) 1805. http://doi.org/10.1109/EIConRus.2018.8317456

12 S. Haykin and B. Kosko: Intelligent Signal Processing (Wiley-IEEE Press, 2001) Gradient Based Learning Applied to Document Recognition.
13 M. Hossain, M. Shamim, Al-Hammadi, and G. Muhammad: IEEE Trans. Ind. Inf. **15** (2018) 1027. https://doi.org/10.1109/TII.2018.2875149
14 S. Lu, Z. Lu, S. Aok, and L. Graham: Proc. Int. Conf. Digital Signal Processing (IEEE, 2018) 1. https://doi.org/10.1109/ICDSP.2018.8631562
15 E. Bochinski, V. Eiselein, and T. Sikora: Proc. IEEE Conf. Advanced Video and Signal Based Surveillance (IEEE, 2017) 1. https://doi.org/10.1109/AVSS.2017.8078516
16 M.-C. Chen, Y.-T. Cheng, and C.-Y. Liu: Sens. Mater. **34** (2022) 151. https://doi.org/10.18494/SAM3553
17 R. Padilla, S. L. Netto, and E. A. B. da Silva: Proc. Int. Conf. Systems, Signals and Image Processing (IEEE, 2020) 237. https://doi.org/10.1109/IWSSIP48289.2020.9145130