Sensors and Materials, Vol. 3#, No. # (202#) 1–##
MYU Tokyo

S & M ####

Chip Contour Detection Based on Real-time Image Sensing and Recognition

Bao-Rong Chang,[1] Hsiu-Fen Tsai,[2*] Chia-Wei Hsieh,[1**] and Mo-Lan Chen[3]

[1]Department of Computer Science and Information Engineering, National University of Kaohsiung, 700, Kaohsiung University Rd., Nanzih District, Kaohsiung 811, Taiwan
[2]Department of Fragrance and Cosmetic Science, Kaohsiung Medical University, 100, Shih-Chuan 1st Road, Kaohsiung 80708, Taiwan
[3]NXP Semiconductors, Taiwan, Ltd., 10, Jing 5th Rd., Nanzih District, Kaohsiung 811, Taiwan

(Received March 28, 2021; accepted May 26, 2021; online published June 15, 2021)

Keywords: deep learning, Jetson Nano, object tracking, real-time image sensing and recognition, GSEH-YOLOv5, attention mechanism, SENet, GhostNet

In this study, the GSEH-YOLOv5 (GhostNet and SENet included in Head-YOLOv5) algorithm is used to realize real-time object tracking and image sensing and recognition on the Jetson Nano embedded platform. The purpose is to instantly detect the appearance contour of the chip inside the chip slot. As soon as our system detects the damaged chip, a warning is generated, and the correct location of the damaged chip in the chip slot is labeled. After that, the operator immediately removes the damaged chip to prevent the next chip from being damaged. Finally, we also analyze and compare the performance between the improved GSEH-YOLOv5 algorithm and the traditional YOLOv5 algorithm to verify that the proposed method has the better performance.

1. Introduction

With the promotion of Industry 4.0, innovative technology has brought significant reforms to factories, introducing big data, cloud technology, automation and simulation, and other technologies into factories, significantly increasing factory production capacity. However, unexpected emergencies occur when the plant runs on a production line due to problems with old equipment. For example, in automated chip transportation in a production line, if a chip is accidentally crushed, the next batch of chips will also be affected. This will impact productivity and yield, so how to minimize such losses is the primary goal of this research. Generally speaking, object detection is composed of two modes. Mode 1 (single-stage) combines the two procedures of identifying an object's position and classification for processing. Mode 2 (two-stage) performs these two procedures separately. Among the past object detection algorithms, the accuracy of two-stage algorithms for object detection is better than that of single-stage algorithms. Common two-stage algorithms are RCNN,[(1)] Fast-RCNN,[(2)] and Faster-RCNN,[(3)] but the most significant disadvantage of two-stage algorithms is the calculation time. When there

*Corresponding author: e-mail: sftsai@kmu.edu.tw
**Corresponding author: e-mail: david9106432@gmail.com

ISSN 0914-4935

are a large number of frame-selected targets in an image, the subsequent object classification must classify a large number of frame-selected targets, which is time-consuming. In particular, many everyday applications often require real-time object detection. Typical applications include vehicle tracking, street view analysis, mask-wearing testing, operator clothing testing, and product inspection of factory production lines. Object detection has recently become a mature technology, but the importance of the single-stage algorithms has increased because of their higher speed while maintaining reasonable accuracy of object detection.[4] Therefore, many applications require a single-stage algorithm for object detection in real-time. Moreover, current single-stage algorithms have benefited from improved hardware and the development of technology. Indeed, the accuracy of single-stage algorithms can be indistinguishable and even exceed that of two-stage algorithms. In this study, we mainly use a single-stage YOLOv5 algorithm[5–7] released in 2020, which is expected to achieve a higher speed and higher accuracy in image sensing and recognition.

2. Related Work

According to Ref. 8, the core feature of the original YOLO is to treat an input image as many grids with the same width and height and predict the objects in each grid. However, at that time, YOLO used two bounding boxes in each grid to predict objects, and a grid had only one class. Redmon and Farhadi[9] reported that YOLOv2 had a few newly improved methods to improve the speed and accuracy of the model, such as batch normalization and an anchor box.

Redmon and Farhadi newly added a residual network to YOLOv3, making the network structure deeper.[10] Compared with YOLOv2, YOLOv3 had greatly improved accuracy while maintaining comparable speed to the previous version.[11] The anchor box skills were retained in the later YOLO versions. Moreover, in Ref. 12, the main goal of the authors was to design a fast operating system using new functions. The proposed fast operation system was not only to reduce the computation load dramatically but also to speedup target detector in the production system and optimize parallel computing significally. After that, some of these new functions were combined to achieve state-of-the-art results.

Marco *et al.*[13] found that although compression algorithms can usually successfully reduce the inference time, this is at the cost of reduced accuracy. They proposed a new alternative method to execute a deep neural network (DNN) on embedded devices efficiently by dynamically determining which DNN to use for a given input by considering the required accuracy and inference time. Moreover, Sun *et al.*[14] proposed a target detection network for embedded systems. The M-YOLO (Mobile-YOLO) model presented in their study combined residual blocks[11] and depthwise separable convolution[15] of the feature selection layer to reduce the computational complexity of the network.

Howard *et al.*[16] reported that MobileNet mainly uses depthwise separable convolution[15] to construct a lightweight[17] DNN. By performing a traditional convolution operation to generate similar feature maps with lower computational costs, Han *et al.*[18] demonstrated that not all feature maps need be generated.

The dominant sequence transduction model is based on complex recursive or convolutional neural networks, including encoders and decoders.[19] The model with the best performance also connects the encoder and decoder through the attention mechanism. Moreover, the innovative feature of the SENet network is to pay attention to the relationship between feature vectors, so that the model can actively learn the importance of features between different feature vectors.[20]

3. Method

In the following steps, we will use Anaconda3 to build an executable environment for YOLOv5 in Windows 10 and collect the data used. The traditional model of YOLOv5 is modified to make it more suitable for real-time object detection of a Jetson Nano embedded platform, which is a GPU-driven platform designed by NVIDIA with an executable environment. Then, the improved model deploys Jetson Nano. Finally, we perform a test on Jetson Nano to compare the performance of the traditional version of YOLOv5 with our improved version.

3.1 Architecture for on-site detection of chip contour

Figure 1 shows the architecture of the YOLOv5 on-site chip detection system.[21,22] The YOLOv5 model is trained with supervised learning. It is necessary to collect data and manually label the collected data as the input of the training model. We have also improved the traditional YOLOv5 model to make it more prominent on the Jetson Nano embedded platform. Finally, we also made a warning system with chip detection as the main novelty of this study, which can immediately provide helpful location information for users to view.

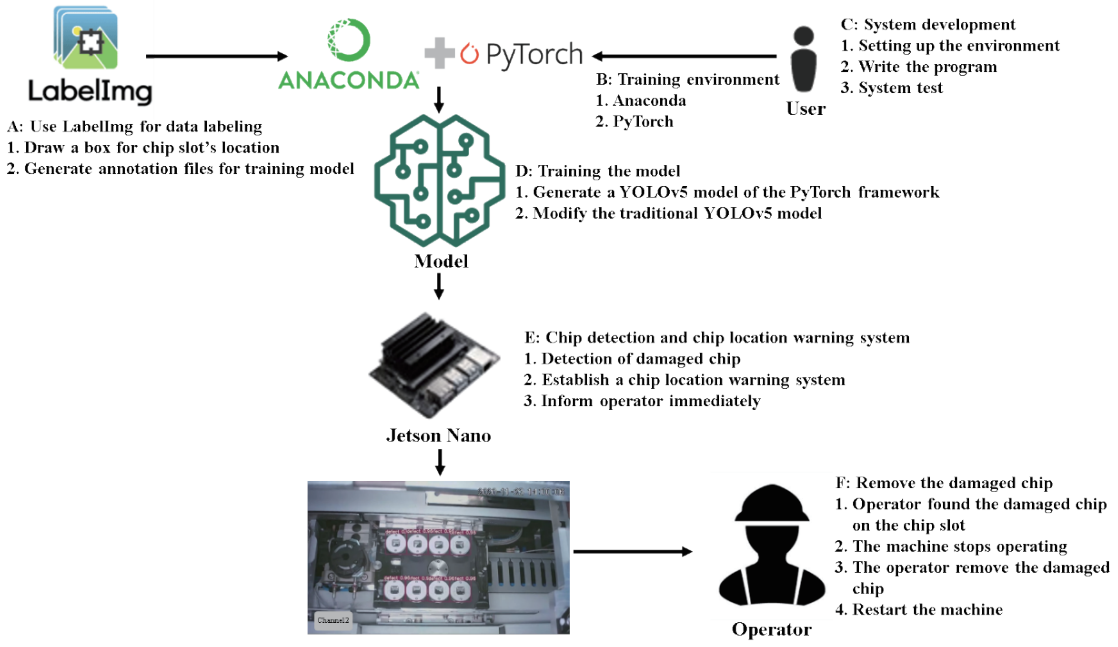


Fig. 1. (Color online) Architecture of YOLOv5 chip detection system.

3.2 Data preparation and labelling

A total of 553 training data and 89 verification data in the data set[23] provided by Taiwan NXP Semiconductors Co., Ltd., are labeled through LabelImg, as shown in Part A of Fig. 1. Firstly, the recorded video is outputted as a piece of an image with part of the video in each frame. Then the part of the image with the detection target is labeled manually. Each image has eight targets, which are labeled with their respective conditions, empty, occupied, or defective, as shown in Fig. 2.

3.3 Model building and package installation

Anaconda3 has commonly used packages with the primary executable environment of YOLOv5 pre-installed, and thus users do not have to install them again in Windows 10. However, YOLOv5 operates under the PyTorch framework, so it is still necessary to install other PyTorch-related packages required by YOLOv5 on Anaconda3, as shown in Fig. 3.

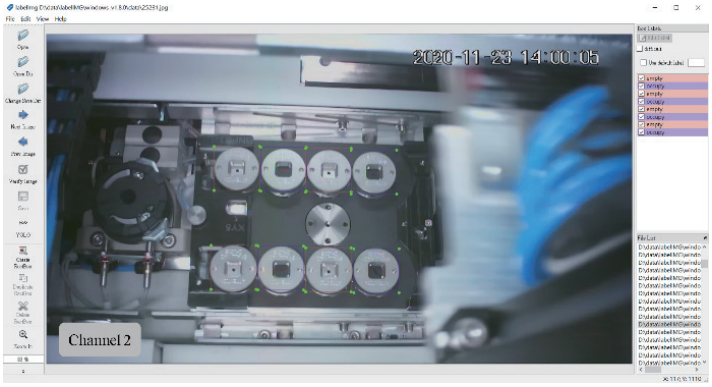


Fig. 2. (Color online) Image labeling with LabelImg.

PyTorch Build	Stable (1.8.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 10.2	CUDA 11.1	ROCm 4.0 (beta)	None

Fig. 3. (Color online) PyTorch tool kit installed in a workstation.

3.4 Model training and performance evaluation

In this study, we use PyTorch as the training framework and use the collected and labeled data as the training data set to train a model suitable for the data. A screenshot of the actual training process record is shown in Fig. 4. After the model is trained, the self-trained model is evaluated. If the accuracy rating does not reach the expected level, it is necessary to adjust the parameters or check whether the data set is labeled incorrectly or not carefully labeled. After completing the adjustment, training is performed again to confirm that the accuracy rating reaches the expected level. The mean average precision (mAP) is commonly used to judge the quality of a model. The closer mAP is to 1, the better the performance of the model, as shown in Fig. 5.

Epoch	gpu_mem	box	obj	cls	total	targets	img_size	
1/1999	6.24G	0.1059	0.2519	0.03734	0.3951	972	416:	11%
1/1999	6.24G	0.1066	0.2513	0.03752	0.3954	1009	416:	11%
1/1999	6.24G	0.1066	0.2513	0.03752	0.3954	1009	416:	22%
1/1999	6.24G	0.1057	0.2469	0.03742	0.39	894	416:	22%
1/1999	6.24G	0.1057	0.2469	0.03742	0.39	894	416:	33%
1/1999	6.24G	0.1057	0.2474	0.03729	0.3904	1011	416:	33%
1/1999	6.24G	0.1057	0.2474	0.03729	0.3904	1011	416:	44%
1/1999	6.24G	0.1055	0.2492	0.0373	0.392	1004	416:	44%
1/1999	6.24G	0.1055	0.2492	0.0373	0.392	1004	416:	56%
1/1999	6.24G	0.1051	0.2506	0.03728	0.393	974	416:	56%
1/1999	6.24G	0.1051	0.2506	0.03728	0.393	974	416:	67%
1/1999	6.24G	0.1049	0.251	0.03717	0.393	993	416:	67%
1/1999	6.24G	0.1049	0.251	0.03717	0.393	993	416:	78%
1/1999	6.24G	0.1048	0.2524	0.03709	0.3943	1050	416:	78%
1/1999	6.24G	0.1048	0.2524	0.03709	0.3943	1050	416:	89%
1/1999	6.24G	0.1045	0.2527	0.03699	0.3942	620	416:	89%
1/1999	6.24G	0.1045	0.2527	0.03699	0.3942	620	416:	100%
1/1999	6.24G	0.1045	0.2527	0.03699	0.3942	620	416:	100%

Fig. 4. Screenshot of the actual training process record.

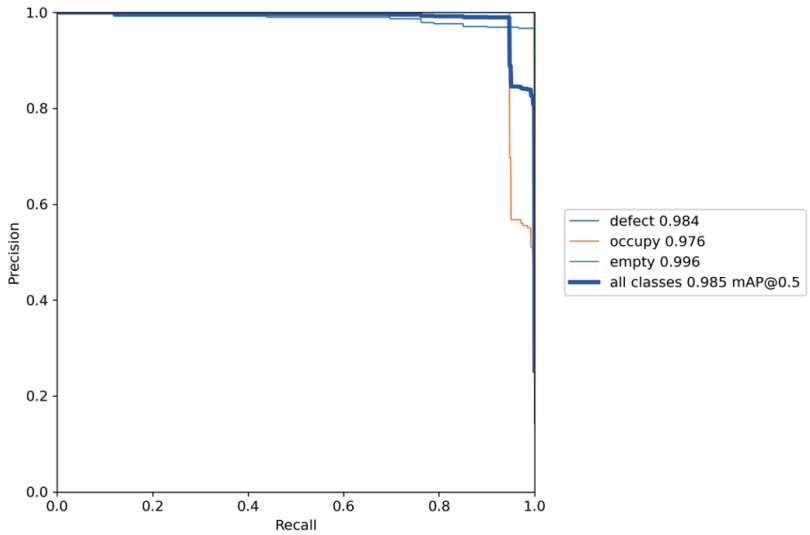
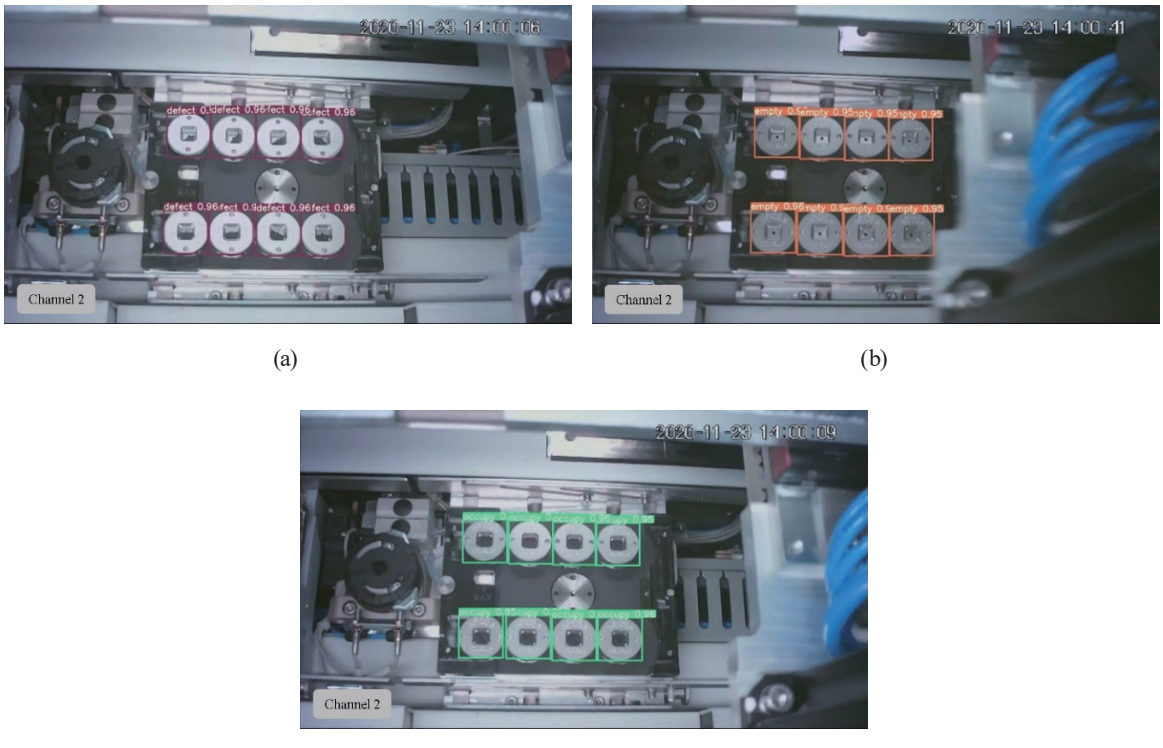


Fig. 5. (Color online) Line chart of mAP with traditional YOLOv5.

3.5 On-site image sensing and recognition

We use the best model in the Jetson Nano platform.[24] This is the smallest embedded platform in the NVIDIA Jetson series and is shown in Fig. 6. The detection results are output after the Jetson Nano platform is used for image sensing and recognition. In fact, the result of the on-site image sensing and recognition is acceptable according to the model performance as shown in Fig. 7.

Fig. 6. Jetson Nano embedded platform.



(a) (b)

(c)

Fig. 7. (Color online) Chip detection showing results for (a) defective condition, (b) empty condition, and (c) occupied condition.

3.6 On-site detection of chip contour

The data analyzed include the classification of the object, the classification accuracy, and the chip's exact location, as shown in Fig. 8. This is because under the preset conditions, when a damaged chip appears, the machine must stop operation immediately to avoid further damage. Therefore, as long as the chip detection system detects a damaged chip, it will automatically immediately send a message to the user informing them that a chip inside the chip slot on the machine is damaged, as shown in Fig. 9. The Notepad text editor contains detailed information of the chip, and its actual location is shown by the position of the red box.

3.7 Modification of the YOLOv5 network architecture

The YOLOv5 network architecture is modified to reduce the number of calculations required for feature extraction and the number of parameters used to generate valuable features. The main modules used include the Ghost bottleneck block of GhostNet[18] and the SE module of SENet[20] to replace the traditional CSP module, as shown in Fig. 10.

3.8 Average accuracy of GSE-YOLOv5 model

The identification performance of the improved GSE-YOLOv5 model is evaluated, and the overall average accuracy obtained by training with the improved model is shown in Fig. 11.

3.9 Average accuracy of GSEH-YOLOv5 model

Next, the identification performance of the improved GSEH-YOLOv5 model is evaluated, and the overall average accuracy obtained by training with the improved model is shown in Fig. 12.

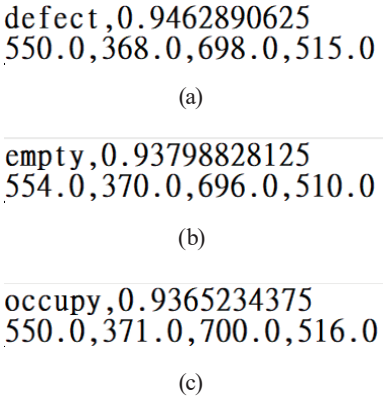


Fig. 8. Spatial location and accuracy of detected objects. (a) Defective case. (b) Empty case. (c) Occupied case.

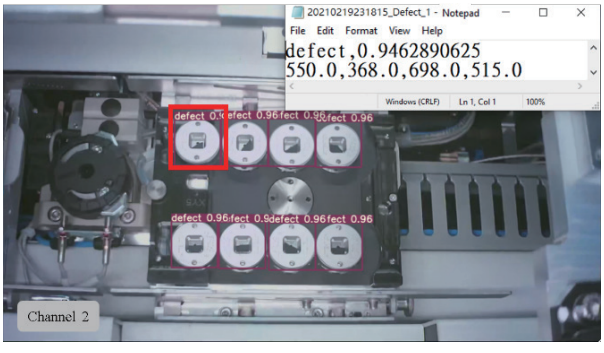
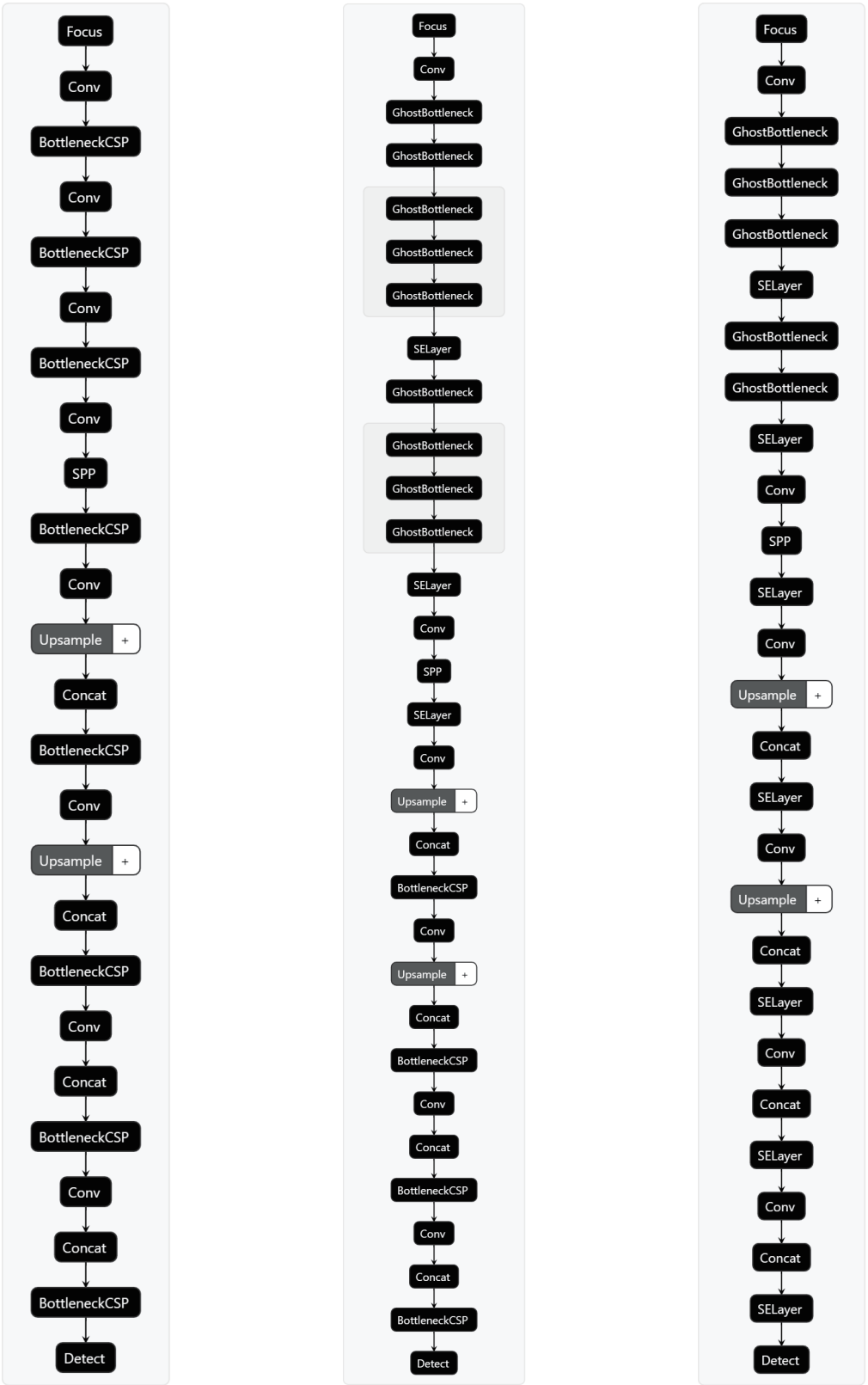


Fig. 9. (Color online) Display information about the exact location.



(a) (b) (c)

Fig. 10. Architectures of three YOLOv5-related models. (a) Traditional YOLOv5 model, (b) GSE-YOLOv5 model, and (c) GSEH-YOLOv5 model.

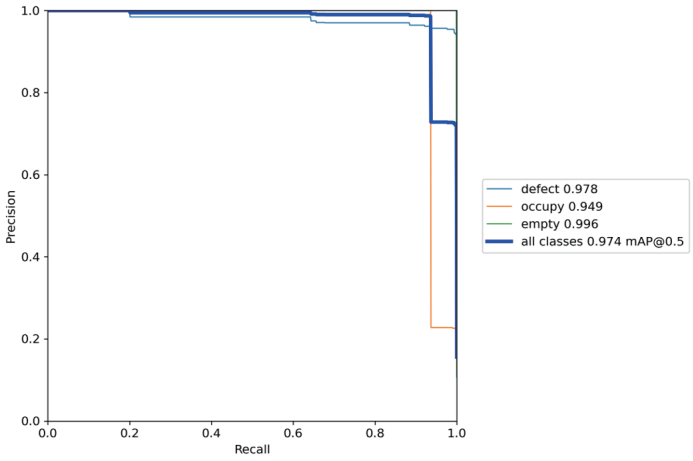


Fig. 11. (Color online) Line chart of mAP of GSE-YOLOv5.

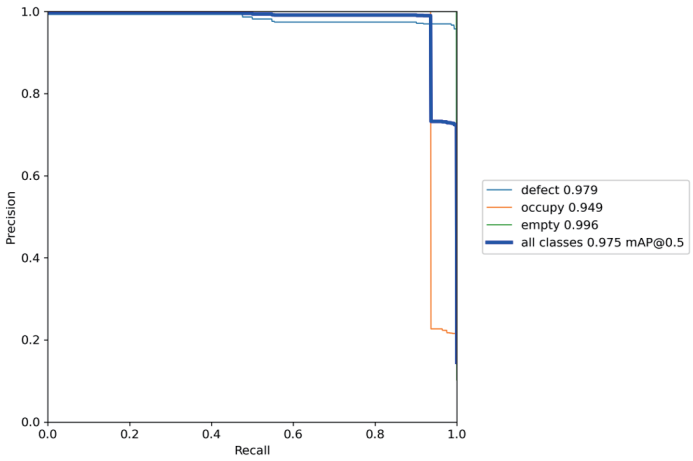


Fig. 12. (Color online) Line chart of mAP of GSEH-YOLOv5

4. Experimental Results and Discussion

4.1 Estimation of training time and video inference time

We train the same training data set for the three YOLOv5-related models on a workstation. The object detection performances of the three YOLOv5-related models are tested using Jetson Nano with 1805 frames of test video, and the inference time needed for each image frame is calculated. Equation (1) is used to calculate the average inference time AIT_{ijk} of the three YOLOv5-related models for each image frame, where VIT_{ijk} represents the total test video's inference time and FN is the total number of test video frames.

$$AIT_{ijk} = \frac{VIT_{ijk}}{FN}, \text{where } i = 1, 2, ..., l,\ j = 1, 2, ..., m,\ k = 1, 2, ..., n \tag{1}$$

The input image size is set to 416 × 416, the batch size is set to 64, and the number of iterations is set to 2000. The first row in Table 1 gives the training times of the three YOLOv5-related models based on the same parameters, the second column gives the time needed to infer 1805 frames in the same test image, and the third column gives the average inference time for each frame. Figure 13 shows the inference time for each frame of the test image.

4.2 Real-time detection speed and recognition accuracy

The performance of real-time object detection depends on the number of recognizable frames per second and the recognition accuracy. Equation (2) is used to calculate the number of frames per second with which three YOLOv5-related models can detect objects in real time, where

Table 1
Training and inference times (unit: s).

Method	YOLOv5	GSE-YOLOv5	GSEH-YOLOv5
Training	27784.8	27576	27475.2
Inference	540.701	490.012	410.458
Average	0.17591	0.13429	0.09010

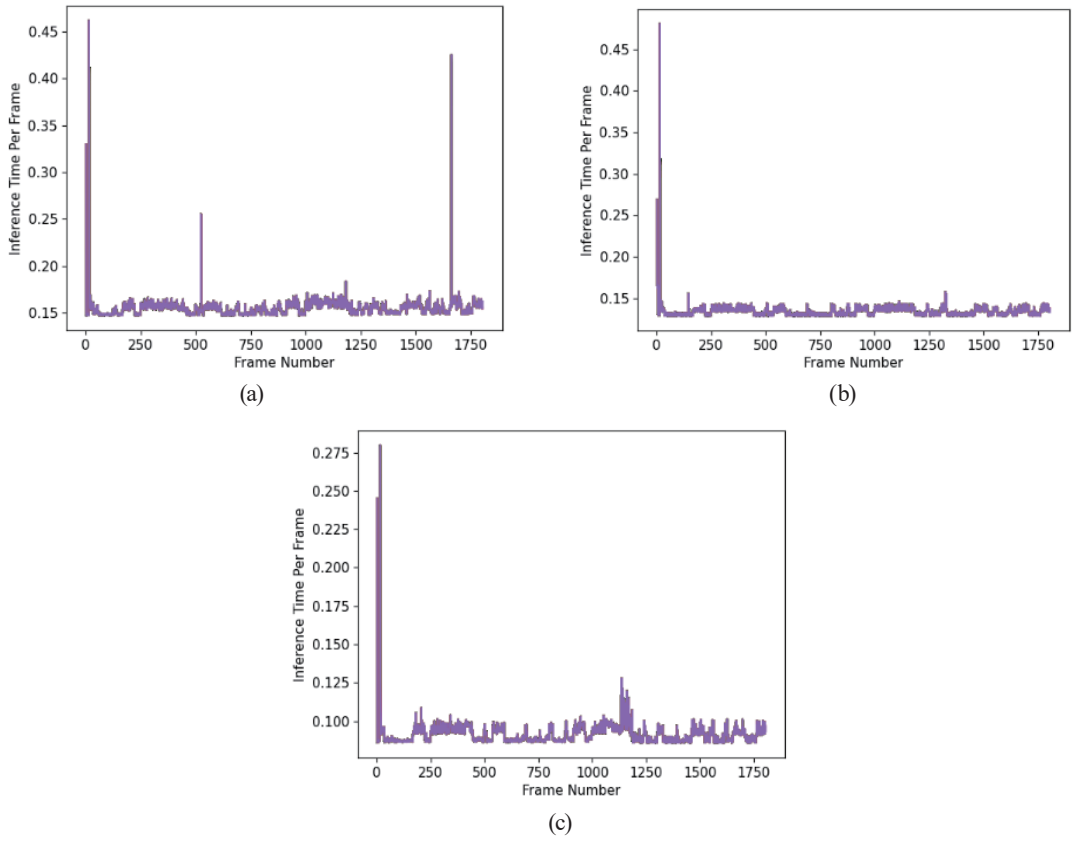


Fig. 13. (Color online) Average inference time for each frame for (a) traditional YOLOv5 model, (b) GSE-YOLOv5 model, and (c) GSEH-YOLOv5 model.

$RAIT_{ijk}$ is the time required for each image in the real-time image source from the camera with 480 × 640 resolution . Equation (3) is used to calculate the accuracy of the three YOLOv5-related models, where c_{ijk} represents the identified categories and APc_{ijk} represents the accuracy of each class.

$$FPS_{ijk} = \frac{1}{RAIT_{ijk}}, \text{where } i = 1, 2, ..., l,\ j = 1, 2, ..., m,\ k = 1, 2, ..., n \tag{2}$$

$$mAP_{ijk} = \frac{APc_{ijk}}{c_{ijk}}, \text{where } i = 1, 2, ..., l,\ j = 1, 2, ..., m,\ k = 1, 2, ..., n \tag{3}$$

Equation (2) is used to calculate the speed in the real-time object detection with the Jetson Nano embedded platform. After that, Eq. (3) is used to calculate the average accuracy of the three YOLOv5-related models after training with the same parameters, as shown in Table 2.

4.3 Operational cost

The number of parameters used and the number of calculations considerably vary among the three YOLOv5-related models, as shown in Table 3.

4.4 Performance indicator

We mainly focus on maintaining high accuracy and improving the frame rate when implemented on the embedded platform, with the frame rate obtained from the traditional YOLOv5 model used as the baseline. Equation (4) is used to calculate the frame rate difference between the three YOLOv5-related models for the Jetson Nano embedded platform. Here, FPS_{ijk} is calculated using Eq. (2), and O_i is the frame rate measured for the traditional YOLOv5 model.

$$PI_{ijk} = \frac{FPS_{ijk}}{O_i}, \text{where } i = 1, 2, ..., l,\ j = 1, 2, ..., m,\ k = 1, 2, ..., n \tag{4}$$

Table 2
Speed and accuracy of models.

Method	YOLOv5	GSE-YOLOv5	GSEH-YOLOv5
Speed (fps)	5.74713	6.09756	8.77193
Accuracy (%)	98.5	97.4	97.5

Table 3
Numbers of parameters and flops of models.

Method	YOLOv5	GSE-YOLOv5	GSEH-YOLOv5
Parameters (#)	7251912	5310840	4182136
Flops (Gflops)	16.8	10.8	6.9

Table 4
Performance indicators of models.

Method	YOLOv5	GSE-YOLOv5	GSEH-YOLOv5
PI	1	1.06097	1.53631

For the traditional YOLOv5 model, the performance indicator is calculated to be 1 using Eq. (4), and the performance indicators of the GSE-YOLOv5 and GSEH-YOLOv5 models are analyzed, as shown in Table 4.

4.5 Discussion

The experimental results show that the size of a single data file required for the traditional YOLOv5 model is 14.4 MB, and real-time object detection can be carried out on the embedded platform at a frame rate of 5.74713 fps with an accuracy of 98.5%. The size of a single data file required for the GSE-YOLOv5 model is 10.6 MB, which is 26.1% less than that of the traditional YOLOv5 model, and it can perform real-time object detection at a frame rate of 6.09756 fps on the embedded platform, which is 6.1% higher than that of the traditional YOLOv5 model. The accuracy is 97.4%, which is 1.1% less than that of the traditional YOLOv5 model. The size of a single data file required for the GSEH-YOLOv5 model is 8.3 MB, which is 42.4% less than that of the traditional YOLOv5 model, and it can perform real-time object detection on the embedded platform at a frame rate of 8.77193 fps, which is 52.6% higher than that of the traditional YOLOv5 model. The accuracy is 97.5%, which is 1% less than that of the traditional YOLOv5 model.

5. Conclusion

In this study, we used the object-tracking algorithm of YOLOv5 to perform real-time identification of a chip contour and detect whether there is damage. We evaluated the implementation efficiency and the accuracy of the proposed algorithm experimentally. For real-time object detection on an embedded platform, the results show that the performance of the improved GSEH-YOLOv5 model is better than that of the traditional model and the enhanced GSE-YOLOv5 model. As a result, the proposed approach achieves not only almost the same accuracy as the other two methods, but it also outperforms the others in terms of the object detection speed to significantly shorten the response time.

Acknowledgments

This research was funded by the Special Research Project of the Ministry of Science and Technology of the Republic of China (project numbers MOST 110-2622-E-390-001 and MOST 109-2622-E-390-002-CC3). We also sincerely thank Taiwan NXP Semiconductors Co., Ltd. for providing assistance in this research.

References

1 R. Girshick: 2015 IEEE Int. Conf. Computer Vision (ICCV) (IEEE, 2015) 1440–1448. https://doi.org/10.1109/ICCV.2015.169
2 S. Ren, K. He, R. Girshick, and J. Sun: Proc. 28th Int. Conf. Neural Information Processing Systems (NIPS, 2015) 91–99. https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf
3 K. He, G. Gkioxari, P. Dollár, and R. Girshick: 2017 IEEE Int. Conf. Computer Vision (ICCV) (IEEE, 2017) 2980–2988.
4 L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu: IEEE Access **7** (2019). https://doi.org/10.1109/ACCESS.2019.2939201
5 G. Jocher: YOLOv5, GitHub, https://zenodo.org/record/4418161#.X_iH_ugzaUk (accessed March 2021).
6 T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie: 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR) (IEEE, 2017) 936–944. https://doi.org/10.1109/CVPR.2017.106
7 S. Liu, L. Qi, H. Qin, J. Shi and J. Jia: 2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition (IEEE, 2018) 8759–8768. https://doi.org/10.1109/CVPR.2018.00913
8 J. Redmon, S. Divvala, R. Girshick, and A. Farhadi: Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR) (IEEE, 2016) 779–788. https://doi.org/10.1109/CVPR.2016.91
9 J. Redmon and A. Farhadi: Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR) (IEEE, 2017) 6517–6525. https://doi.org/10.1109/CVPR.2017.690
10 J. Redmon and A. Farhadi: CoRR (2018). http://arxiv.org/abs/1804.02767
11 K. He, X. Zhang, S. Ren, and J. Sun: Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR) (IEEE, 2016) 770–778. https://doi.org/10.1109/CVPR.2016.90
12 A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao: YOLOv4 (2020). http://arxiv.org/abs/2004.10934
13 V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib: CoRR (2019). http://arxiv.org/abs/1911.04946
14 Y. Sun, C. Wang, and L. Qu: 2019 IEEE Int. Conf. Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS) (IEEE, 2019) 506–512. https://doi.org/10.1109/IUCC/DSCI/SmartCNS.2019.00110
15 F. Chollet: 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR) (IEEE, 2017) 1800–1807. https://doi.org/10.1109/CVPR.2017.195
16 A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam: CoRR (2017). http://arxiv.org/abs/1704.04861
17 X. Chen, J. Chen, X. Han, C. Zhao, D. Zhang, K. Zhu, and Y. Su: IEEE Access **8** (2020). https://doi.org/10.1109/ACCESS.2020.2970461
18 K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu and C. Xu: Proc. 2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR) (IEEE, 2020) 1577–1586. https://doi.org/10.1109/CVPR42600.2020.00165
19 A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin: CoRR (2017). http://arxiv.org/abs/1706.03762
20 J. Hu, L. Shen, and G. Sun: 2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition (IEEE, 2018) 7132–7141. https://doi.org/10.1109/CVPR.2018.00745
21 N. Yu, Q. Xu, and H. Wang: IEEE Trans. Semicond. Manuf. **32** (2019) 4. https://doi.org/10.1109/TSM.2019.2937793
22 T. Nakazawa and D. V. Kulkarni: IEEE Trans. Semicond. Manuf. **31** (2018) 2. https://doi.org/10.1109/TSM.2018.2795466
23 M. Saqlain, Q. Abbas, and J. Y. Lee: IEEE Trans. Semicond. Manuf. **33** (2020) 3. https://doi.org/10.1109/TSM.2020.2994357
24 V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge: IEEE Access **8** (2020). https://doi.org/10.1109/ACCESS.2020.2964608