Mitigating Overflow of Object Detection Tasks Based on Masking Semantic Difference Region of Vision Snapshot for High Efficiency

Heuijee Yun[1] and Daejin Park[1*]
[1]School of Electronics Engineering, Kyungpook National University, Daegu, Republic of Korea
*Correspondence to: Daejin Park (boltanut@knu.ac.kr)

***Abstract*—Object recognition functions are essential to properly perform safety and autonomous driving functions. However, sophisticated object recognition work requires extensive computation. It is difficult to handle a large amount of computation on the lightweight embedded boards currently used in vehicles. In this paper, we propose a method using machine learning and deep learning for lightweight object recognition algorithm in lightweight embedded boards. We created an algorithm suitable for lightweight embedded boards by appropriately using deep neural network architecture that requires small computational volumes but provides low accuracy, as well as deep-learning algorithms that require large computational volumes but provide high accuracy. After determining the area using a deep neural network architecture algorithm with a relatively small amount of computation, we improved the accuracy by using a more accurate deep learning algorithm. We used OpenCV to process input images in Python, and we processed image by using efficient neural network (ENet) and You Only Look Once (YOLO). By executing this algorithm, we can realize more accurate and lightweighted object recognition.**

***Index Terms*—Autonomous driving, object detection, OpenCV, ENet, YOLO, deep learning**

I. Introduction

Currently, with the development of artificial intelligence technology, its application is expanding. For most functions, image processing is essential. However, to process images in real time, the amount of computation must be small. Although current image processing technologies are developing, they have been unable to effectively process the required amount of computation with required accuracy. In addition, to run on an embedded board with relatively little memory, the computation volume must be very small. In this study, we used a combination of deep neural network architecture and deep-learning techniques to efficiently increase the amount of computation and accuracy. We used semantic segmentation using deep neural network architecture and YOLO using deep learning to highlight the advantages of computational amount and accuracy when each is independently executed.

This study was supported by the BK21 FOUR project funded by the Ministry of Education, Korea (4199990113966, 10%), and the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF2019R1A2C2005099, 10%), and Ministry of Education (NRF2018R1A6A1A03025109, 10%), and Institute of Information & communication Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (no. 2021-0-00944, Metamorphic approach of unstructured validation/verification for analyzing binary code, 70%), and the EDA tool was supported by the IC Design Education Center (IDEC),Korea.

II. Proposed Method

A. Overall Structure

Fig 1 (a) illustrates the system's structure. After receiving the image input to the webcam in units of frames, semantic segmentation is executed using ENet [1]. After inserting an image masked only with ROI as an input of YOLO [2], the result of YOLO is written on the image. It is a structure that combines these frames to output results in real time. Therefore, by setting the ROI (region of interest) using semantic segmentation, we improved the accuracy in YOLO and made it possible in real time using Python.

B. Image Detection Structure

We used ENet program for semantic segmentation. 1 (b) illustrates the ENet segmentation program's structure. First, the image frame from the webcam is set to be processed on the ENet. Because the trained model processes with a size of 256, it is set accordingly. It then loads through the path of the trained model and processes the image. When the image is processed, the resulting image represents the label suitable for each object and its color in the form of a matrix. To filter for only human images in the results, all information except for humans is binarized as 0, and information about humans is binarized as 1. Semantic segmentation is completed only when the mask, which is a binarized matrix, and the original image are combined. In Python, you can use NumPY to represent an image as a matrix. With the used of this term, the binarized matrix and the original image are composed into a matrix of the same format and then masked using the OpenCV function. The original image is depicted in Fig 1 (d) (1), the mask in Fig 1 (d) (2), and the result in Fig 1 (d) (3).

Fig 1 (c) illustrates the structure of the YOLO object detection program. Among the programs that use YOLO, we used darkflow [3], which can be used with Tensorflow. It is convenient to use the functions of Tensorflow which reduces the amount of code. Earlier in the process, the masked image using the ENet is received as an input. Afterward, the YOLO model is loaded into the appropriate path. Because the ROI is set by semantic segmentation, the confidence threshold of YOLO is set high to reduce the amount of computation in YOLO. When the image is processed using the model loaded with the set threshold, each object's label, confidence and the position are displayed as a matrix. With the OpenCV function in Python, the detected object's position is drawn as a box, and the object label and confidence are written on the box. [4]

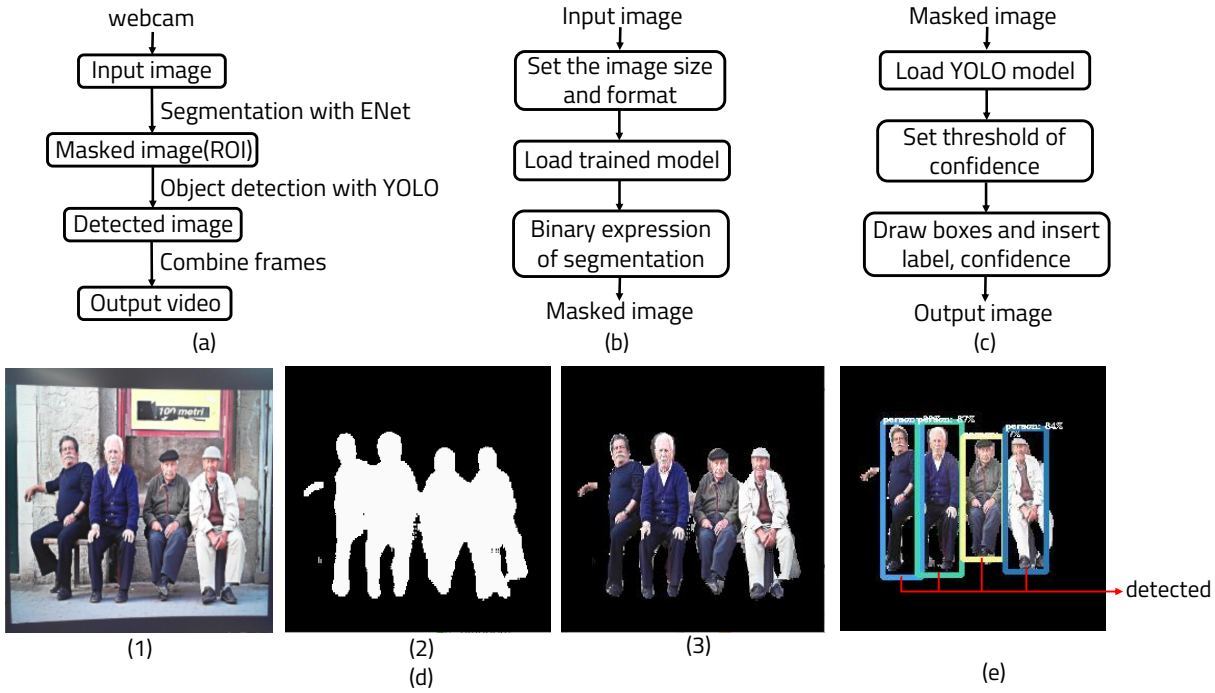

Fig. 1. Structure of program and result (a) Overall structure (b) Semantic segmentation structure (c) YOLO structure (d) LS1028a board (e) Semantic segmentation result (d) YOLO result

C. Measurement

Fig 2 (a) illustrates YOLO time measurement and Fig 2 (b) demonstrates time of ENet and YOLO program. Since it is difficult to photograph a place where the number of people changes in real time, it was measured while continuously showing pictures with different numbers of people on the webcam. In the program using YOLO alone, processing time and FPS measurements are clearly unstable, as they depend on the number of people being counted. However, the program used by integrating ENet and YOLO is clearly stable, even when the number of people changes in FPS and processing time.

We measured memory use as each program ran. "Working set" represents the amount of physical memory the program is currently using. "Private bytes" indicates the amount of memory allocated to the program which includes physical memory and swapped memory. "Page Faults/sec" shows the use of virtual memory. Fig 2 (c) illustrates YOLO's memory measurement and Fig 2 (d) shows YOLO's and ENet's memory usage. In both programs, the value of Page Faults/sec was used small, so interrupts occurred less frequently. The average amount of physical memory used to process YOLO alone was 1.558 GB, and the average working set required to use YOLO and ENet together was 2.223 GB. When ENet and YOLO are used together, the models of the two programs need to be loaded separately, so memory use is of course higher than when YOLO is used alone. However, this method is valid because the increased number of memory bytes greatly improves accuracy, and it can be used on an embedded board.

Fig 3 (a) illustrates the programs' average time and fps measurement. With much fewer convolution layers than YOLO and a fast and compact encoder decoder structure, the lowest average processing time was 0.156 seconds per frame, and the average fps was highest at 6.41. When YOLO was used alone, the average processing time per frame was 0.281 seconds, and the average fps was 3.645. When processing the results of ENet with YOLO, the average time spent in YOLO was 0.269 seconds and the average fps was 3.701. We confirmed that the method of setting the ROI using ENet and processing it as YOLO's input can reduce the burden on YOLO's execution. This result is an advantage of noticeably lowering the threshold of YOLO.

Fig 3 (b) demonstrates the programs' accuracy and average errors. Several methods are available to measure a model's accuracy in deep learning, among which it was measured using a confusion matrix [5]. We can divide into 4 states as TP when the model corrects the correct answer, TN when the model incorrectly predicts the correct answer, FP when the model incorrectly predicts an incorrect answer as a correct answer, and FN when the model incorrectly predicts the correct answer as an incorrect answer. Accuracy can be calculated using these states by this equation $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$. To measure the object recognition program's accuracy, we divided the total number of recognition frames by the number of frames that accurately matched the number of people. The accuracy is clearly better when YOLO and ENet are used together. However, since we recognized the part where the number of people was accurately matched with TP and TN, we calculated the error by comparing the number of

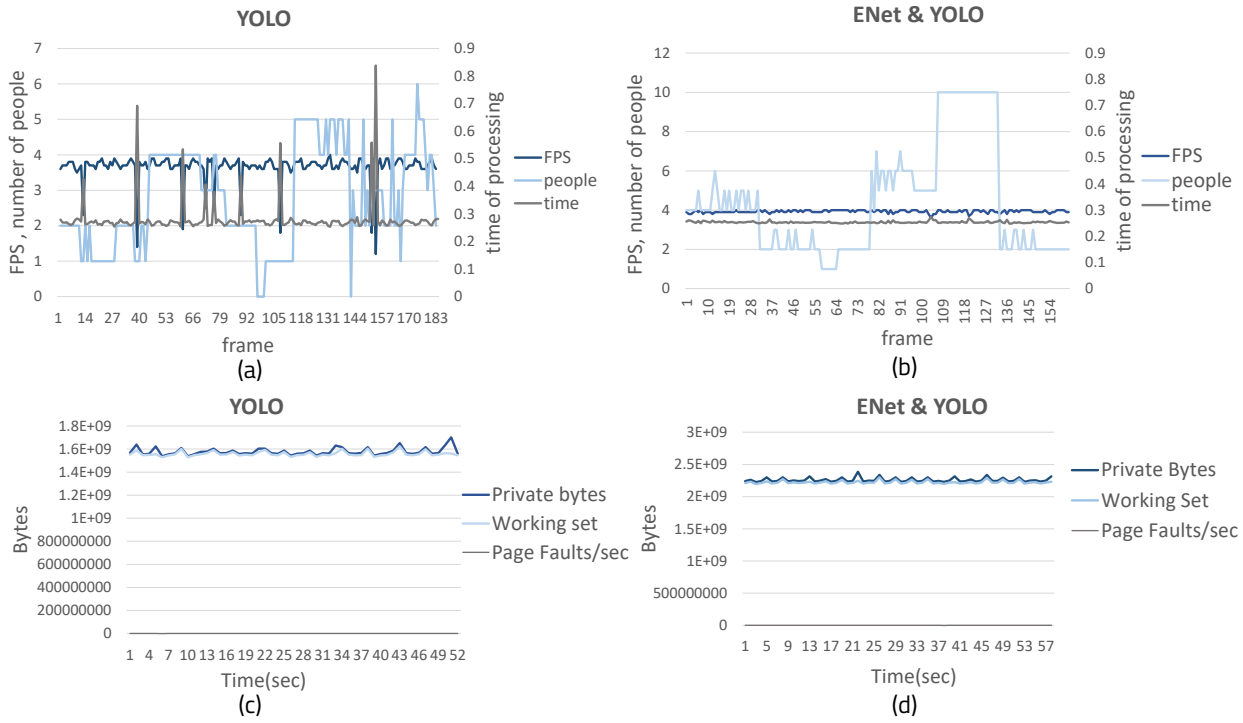

Fig. 2. Time and memory usage measurement (a) Time measurement of YOLO only (b) Time measurement of YOLO and ENet (c) Memory usage measurement of YOLO only (d) Memory usage measurement of ENet and YOLO

recognized people with the number of real people to calculate with more appropriate accuracy. When we calculated accuracy, our method did not improve significantly. However, regarding the average recognition error, when YOLO was used alone, it was 7.097, and when ENet and YOLO were used together, it was 2.913, a clear difference. As the number of people in the photo increases, signs emerge that the errors increase when YOLO is used alone. However, if ENet and YOLO are used together, errors can be reduced and accurate recognition can be achieved.

Program	Average time	Average fps
ENet	0.156s	6.41
YOLO	0.281s	3.645
ENet + YOLO	0.269s	3.701

(a)

Program	Accuracy	Average error of counting
ENet + YOLO	0.574	2.913
YOLO	0.549	7.097

(b)

Fig. 3. Average time and accuracy measurement of programs (a) Average time and fps measurement of ENet, YOLO and ENet and YOLO (b) Accuracy of YOLO, ENet and YOLO

III. Conclusion

In this paper, we proposes a structure of realtime object detection using semantic segmentation and YOLO for advanced accuracy and small amount of computation. We proposed this structure because it is difficult to implement high accuracy in real time due to computational volume and memory limitations. When a webcam image is input, the ROI is set by semantic segmentation using the trained model of ENet. Then we can run YOLO to mark the object's position, label and confidence. As a result of measuring processing time, FPS, accuracy and error, respectively, our program clearly recognizes objects with much fewer errors and greater accuracy. With this structure, we can optimize real-time object detection with great accuracy and less computation.

References

[1] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," 2016.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[3] T. H. Trieu, "Darkflow," *GitHub Repository.Available online:https://github.com/thtrieu/darkflow(accessed on 14 February 2019)*, 2018.

[4] P. Ren, W. Fang, and S. Djahel, "A novel yolo-based real-time people counting approach," in *2017 International Smart Cities Conference (ISC2)*, 2017, pp. 1–2.

[5] J. T. Townsend, "Theoretical analysis of an alphabetic confusion matrix," *Perception & Psychophysics*, vol. 9, no. 1, pp. 40–50, 1971.