Multimedia Tools and Applications (2022) 81:36375–36396
https://doi.org/10.1007/s11042-021-11480-0

1213: COMPUTATIONAL OPTIMIZATION AND APPLICATIONS FOR HETEROGENEOUS MULTIMEDIA DATA

YOLO with adaptive frame control for real-time object detection applications

Jeonghun Lee[1] · **Kwang-il Hwang**[1]

Received: 30 December 2020 / Revised: 1 July 2021 / Accepted: 19 August 2021 /
Published online: 18 September 2021

Abstract
You only look once (YOLO) is being used as the most popular object detection software in many intelligent video applications due to its ease of use and high object detection precision. In addition, in recent years, various intelligent vision systems based on high-performance embedded systems are being developed. Nevertheless, the YOLO still requires high-end hardware for successful real-time object detection. In this paper, we first discuss real-time object detection service of the YOLO on AI embedded systems with resource constraints. In particular, we point out the problems related to real-time processing in YOLO object detection associated with network cameras, and then propose a novel YOLO architecture with adaptive frame control (AFC) that can efficiently cope with these problems. Through various experiments, we show that the proposed AFC can maintain the high precision and convenience of YOLO, and provide real-time object detection service by minimizing total service delay, which remains a limitation of the pure YOLO.

Keywords Embedded systems · Frame control · Object detection · Real-time · YOLO

1 Introduction

Object detection is an important technology that enables computers to have object detection ability such as human vision by recognizing each object in an image. Traditional object detection methods (HARR [6], SIFT [20], and HOG [24]) were considered as representative techniques until the early 2000s, but it was difficult for them to be widely used in various applications due to the limited performance. Deep learning technology, which has rapidly developed since the mid-2000s, has shown the greatest achievement, especially in the field of image object detection. Various object detection techniques based on deep learning (R-CNN [9], Faster R-CNN [30], and YOLO [28]) overcome the limitations of the performance of existing technologies, and their object detection capabilities are similar to

✉ Kwang-il Hwang
hkwangil@inu.ac.kr

[1] Dept. of Embedded Systems Engineering, Incheon National University, Incheon 22012, Korea

those of humans or sometimes exceed human abilities. In particular, YOLO is a technology most widely used in various applications based on object detection in recent years by providing convenience of use through continuous version upgrades (YOLOv2 [26], YOLOv3 [27], and YOLOv4 [3]).

Thus, in recent years, various applications [4, 8, 12, 18, 22, 29, 32, 35] using YOLO object detection have been developed. In addition, interest in edge video processing techniques [2, 37] to improve intelligence at the edge of a network is increased. Many applications using YOLO perform object detection from images being captured in real time using IP (network) cameras. However, in the case of YOLO, the performance varies greatly depending on the hardware running the YOLO core for object detection service. As a result, YOLO's real-time processing is highly dependent on hardware specifications. Thus, interest in improving the real-time processing capability of YOLO has been greatly increased in recent years, and various studies are being conducted [5, 7, 11, 17, 19, 21, 23, 33] to solve the problem of YOLO's real-time object detection. Most studies on the improvement of YOLO's real-time processing have proposed various methods to reduce the size and complexity of the neural network inside YOLO. However, using these methods is not desirable in terms of user convenience of YOLO because all internal models of YOLO should be modified, and additional verification of the object detection performance itself is required to apply a new technique in an application.

Therefore, we propose an Adaptive Frame Control technique to enhance real-time object detection processing while maintaining the object detection performance of the existing YOLO. The proposed model enables consistent object detection service for various input sources of YOLO, and can solve real-time processing problems that may occur in Real Time Streaming Protocol (RTSP) used with YOLO object detection applications, when the video input source of the applications is IP cameras. The proposed AFC is designed and added on the existing YOLO core, and enables real-time object detection with deadlines required by various applications, by optimizing the frame input from various sources and providing adaptively controlled frames to YOLO core.

The organization of this paper is as follows. In Section 2 we present the requirements for YOLO real-time object detection applications, and then point out problems that may occur in a real-time object detection application in Section 3. Next, in Section 4 existing studies related to real-time object detection are introduced, and limitations of existing technologies are presented. In Section 5, we propose a new YOLO model with Adaptive Frame Control for real-time object detection applications. In Section 6 the details of the system implementation proposed in this paper are presented and the performance of AFC and YOLO in various aspects is compared and evaluated through experiments. Finally, this paper concludes in Section 7.

2 Requirements for real-time applications based on object detection

With the development of deep learning-based object detection technologies, the accuracy and reliability of image object detection has already risen to a very high level. Thus, many intelligent video applications are trying to actively utilize these object detection technologies, and most of them need to be able to detect objects in real time with respect to images captured from cameras.

More specifically, in real-time object detection applications based on a network camera, the following requirements should be satisfied.

– Highly reliable object detection
– Guarantee of object detection service time
– Guarantee of received frame rate
– Guaranteed object detection service in deadline (real-time capability)

2.1 Performance of object detection

The performance of object detection is determined by the structure of the neural network used, the training data, and the optimal weight selected in the learning process. If an application uses YOLO as an object detection model and wants to recognize general objects, it has the advantage of being able to use the weights already learned and provided in YOLO as it is. Precision and recall are used as important indicators to evaluate the actual object detection performance, and the objective performance of the object detection model is evaluated through mean Average Precision (mAP) and loss measurements in the learning process. Also, even in the case of the same model, the actual performance varies greatly depending on the image input size. Figure 1 is the result of measuring mAP of YOLO object detection for various input sizes.

2.2 Object detection service time

The object detection service time largely depends on the hardware running the software. In particular, the execution time of the object detection model varies greatly depending on the performance of the GPU. Figure 2 shows the comparison of the actual detection service time of YOLO executed on various hardware platforms [Jetson Nano [14], Jetson TX2 [15], Jetson Xavier NX [16], and Jetson AGX Xavier [13],

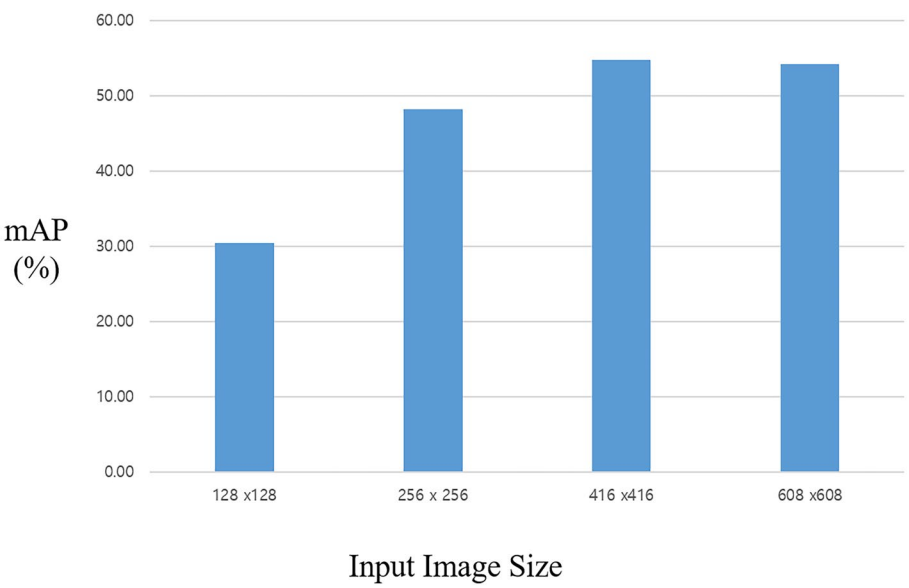

Fig. 1 YOLO object detection precision according to various input sizes (mAP)

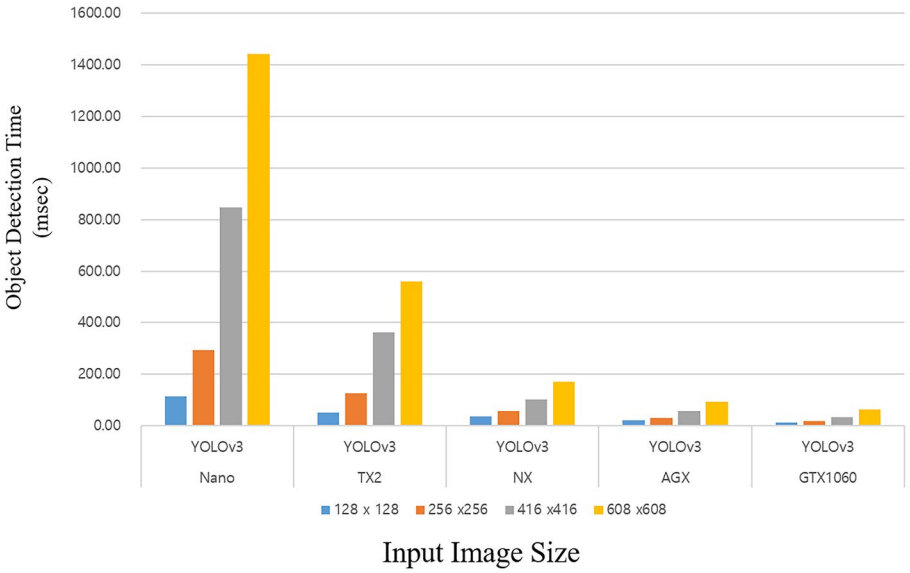

Fig. 2 YOLO detection service time on various hardware with respect to different input image sizes

GTX 1060 (Laptop)], which are being widely used for an AI embedded platform. Also, as shown in the figure, even if YOLO is executed in the same system, the performance also varies greatly depending on the image input size of the object detection module.

Nevertheless, it is desirable to use appropriate hardware in consideration of reasonable execution time in terms of system cost effectiveness of the corresponding application.

2.3 Frame service rate

A typical video frame rate is 15–30 FPS. However, such rates are the speed of a normal video that can be checked by human eyes, and the FPS required for real-time object detection may vary depending on the characteristics of the applications. For example, in order to detect very fast objects (such as traffic monitoring), the application requires a high speed of 10 FPS or higher, but for applications detecting objects moving at a slow speed such as counting the number of people passing through a specific area, it is sufficient to detect the objects at 2 ~ 3 FPS. In other words, it is better to select an appropriate speed that suits the properties of the applications being used rather than a faster object detection speed. The frame rate of object detection process is also highly dependent on the performance of the hardware platform being executed, and Fig. 3 shows the frame rate of video object detection processed on various hardware platforms.

Since different hardware eventually leads to a difference in the cost of the application system, it is desirable for the application developer to use a system with a performance that can be sustained by the application.

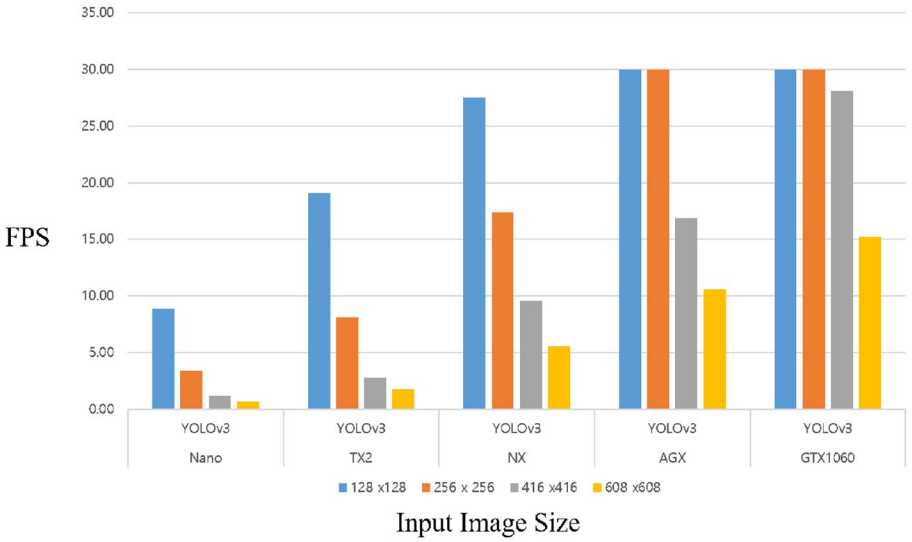

Fig. 3 Comparison of FPS performance on different hardware platforms

2.4 Real-time capability

Most network cameras are based on the RTSP [31], which can guarantee the frame rate of real-time video transmitted over the network. For processing real-time video input from a camera, Darknet YOLO supports the use of IP camera with RTSP protocol as an input source through option setting at execution time.

Figure 4 is the general architecture of RTSP-based YOLO object detection. RTSP has an RTSP queue for buffering frames transmitted over the network. Here we assume that the network delay transmitted by RTSP is negligible. If object detection service rate (T_s) of YOLO is faster than the frame interval (T_a) coming from RTSP, each frame can be processed at the maximum object detection service rate that the system can support. However, as shown in Fig. 3, this processing speed varies greatly depending on the hardware to be

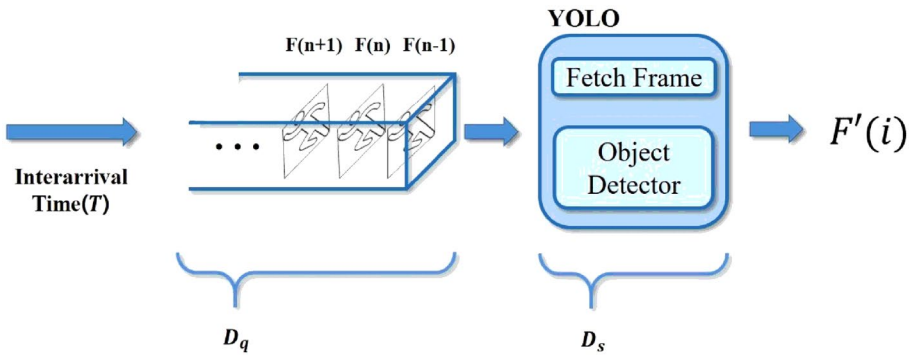

Fig. 4 RSTP-based YOLO object detection service architecture

used. In general, to satisfy the real-time detection service rate which is the same as an input rate, a system equipped with a high-spec GPU of AGX Xavier or higher is required.

3 Problems in real-time object detection

The frame processing speed for real-time object detection required by an application is about several hundred milliseconds. That is, real-time object detection speed of about 3–5 FPS or 10 FPS are enough depending on the characteristics of the application. However, in the case of the existing YOLO, if the object detection service rate is slower than the frame rate transmitted from the camera, it may cause a serious problem for real-time processing.

As shown in Fig. 5, real-time frames received through network cameras come in at a nearly constant rate (T_a). However, if the object detection service time (T_s) is greater than T_a, incoming frames are stored in the RTSP queue during object detection service, and each time the object detection ends, the frames are sequentially taken out of the queue and processed next. The service time of object detection depends on the capabilities of the system hardware in which YOLO is running, and even with the same hardware, the execution time may vary each time by other processes used at the same time.

If the object detection service time (T_s) is longer than the input period (T_a), the waiting time in the queue for subsequent incoming frames becomes longer as shown in Fig. 5, and the total service delay of each input frame (D(n)) can be equated as follows.

$$D(n) = \begin{cases} D_q^{(n-1)} + D_s^{(n-1)} + T_a + D_s^n, (D_q^{(n-1)} + D_s^{(n-1)} - T_a > 0) \\ D_s^n, \left(D_q^{(n-1)} + D_s^{(n-1)} - T_a \le 0\right) \end{cases} \quad (1)$$

where D_q^n is the waiting time in the queue of the nth frame, T_a is the interarrival time for the input frame, and D_s^n is the service time required in object detection of YOLO.

The total service completion time for each frame becomes longer as the delay accumulates over time, and eventually exceeds the deadline required for the application. This cumulative delay increases as time goes by, and as a result, the object detection

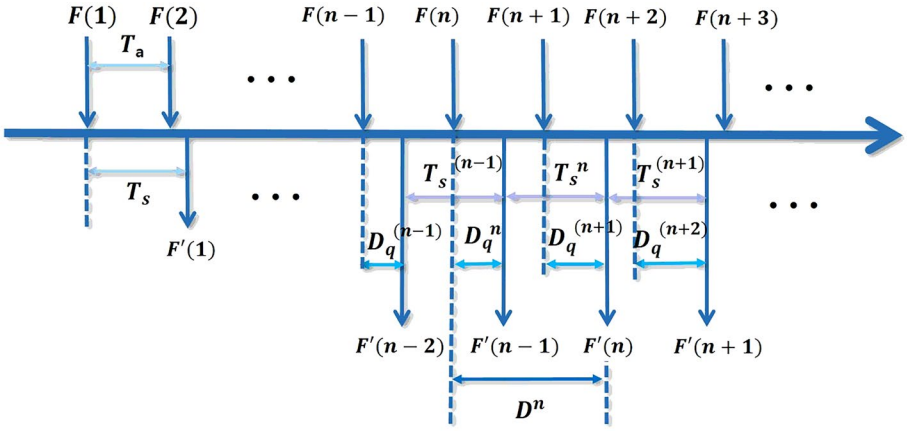

Fig. 5 YOLO real-time processing problem

application will process frames from earlier times (t–n), not in real-time (t). In the case of conventional YOLO, real-time processing from a network camera depends largely on the performance of the actual hardware it is running on. In particular, if object detection processing is delayed, real-time processing may become impossible by continuously affecting subsequent frames. In the case of YOLO, since its main goal was to maximize the processing FPS by input files, an additional element should be included to ensure real-time processing despite excellent recognition performance.

4 Related work

Although YOLO is being used in various applications due to its superior object detection performance and ease of use, problems related to real-time video object detection have been continuously raised, and various studies on these problems have been conducted.

Lu et al. proposed a real-time object detection algorithm [21] for video. The proposed algorithm eliminates the influence of the image background by image preprocessing, and then the Fast YOLO model for object detection to obtain the object information is trained. Kim et al. proposed spiking neural network for real-time object detection [19]. The Spiking-YOLO provides near-lossless information transmission in a shorter period of time for deep spiking neural network (SNN). Feng et al. also proposed Tinier-YOLO [7] for constrained environments. The Tinier-YOLO, based on tiny-YOLO [27] shrinks the model size while achieving improved detection accuracy and real-time performance. Oltean et al. [23] implemented real-time vehicle counting using yolo-tiny and fast motion estimation. He et al. proposed Tiny Fast You Only Look Once (TF-YOLO) [11] to be implemented in an embedded system. Jin et al. also proposed embedded real-time pedestrian detection system [17]. The proposed algorithm modified the YOLO with MobileNet and they performed the algorithm on Jetson TX2 system. Chen et al. proposed an improved YOLO network [5]. The proposed algorithm tailored the network layer structure of YOLOv3-tiny, and quantified the network parameters in the network to reduce the complexity of computing in embedded devices. Silva Pincay in his BS thesis [33] implemented not tiny-YOLO but YOLOv3 on the Raspberry Pi board for real-time object detection.

In addition, some studies have been conducted to perform YOLO's object detection function in hardware with more limited resources. Md. Bahar Ullah proposed CPU Based YOLO [34], a real time object detection model to run on Non-GPU computers. Gour et al. presented an optimized YOLO algorithm [10] for CPU in order to detect road traffic accident and alter systems. Barry, et al. also proposed xYOLO [1] for real-time object detection in humanoid soccer on low-end hardware. Wang et al. proposed Sparse-YOLO [36] which is based on hardware/software co-design of FPGA accelerator for YOLOv2.

In order to improve the real-time processing capability of YOLO, several studies have tried to reduce the execution time of object detection by changing YOLO's neural network itself. In addition, there have been various approaches to improve YOLO's real-time capabilities on hardware with more limited resources. However, most of these studies have been modified to be specific to the application instead of reducing the generalized object detection capability of the original YOLO. In particular, the problem of YOLO object detection for real-time video from a network camera still could not come up with a solution.

5 Adaptive frame control for real-time object detection applications

In Section 3, we already raised the problem of YOLO's real-time object detection on systems with different hardware specifications. In this Section, we propose a new YOLO architecture, YOLO with adaptive frame control (AFC) that can cope well with real-time object detection.

First, the design of a real-time object detection technique with a strict deadline should consider the following requirements:

- Consistent processing of various inputs (network cameras (RTSP), video files, USB cameras)
- Guaranteed maximum frame rate that can be supported by system hardware
- Guaranteed real-time processing of each input frame (no cumulative delay)

5.1 Overview of the proposed real-time YOLO architecture

Figure 6 shows the overall architecture of the proposed YOLO with AFC. As shown in the figure, the proposed AFC does not make changes to the existing YOLO network core. Instead, in order to ensure the real-time performance of YOLO, AFC performs adaptive frame control for real-time input in the preceding stage of YOLO frame fetch. AFC is largely composed of multi-source prefetcher (MSP) and admission control (AC) blocks. In addition, AFC enables consistent real-time processing from various input sources by using an integrated queue accessed by MSP and AC. Through this modular structure, various input sources can be used more conveniently, and since the object detection precision of the existing YOLO can be used as it is, it can be used in various object detection-based applications.

5.2 Multi-source prefetcher

In the case of Darknet YOLO, various input sources (RTSP, Video file, or USB Camera) can be used by parameter setting when executing YOLO. However, since frames coming

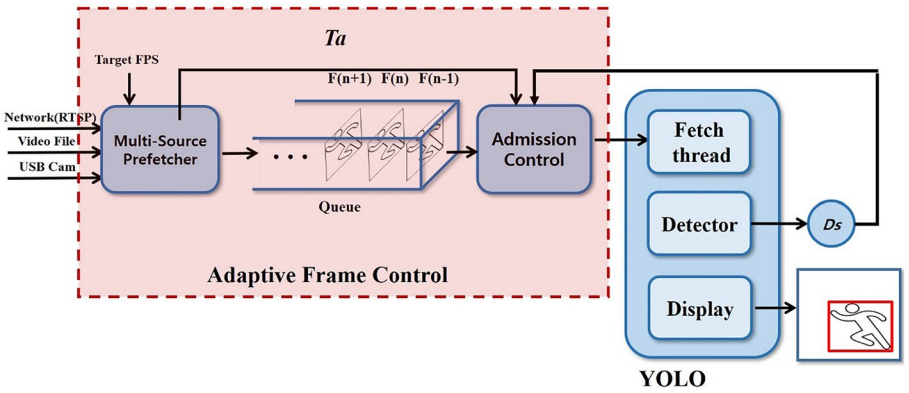

Fig. 6 Overview of real-time YOLO architecture

from each input source are processed at a rate that can be processed by YOLO core, they cannot be processed at a consistent rate. For example, when object detection is performed on a stored CCTV video file, the object detection service rate is faster than 30 FPS in a situation where the hardware running YOLO is high specification (Detection FPS is 30FPS or more). Therefore, it is difficult to grasp the exact time when some objects were detected from the video.

Multi-source prefetcher (MSP) performs a function that enables consistent frame input from various sources to the target FPS required by the application unlike the existing YOLO. MSP is operated as a separate thread, and it continuously reads frames for various inputs at high speed. Figure 7 shows the internal logic of the MSP. The synchronizer inside the MSP compares the reading time of frames with the target FPS, and synchronizes with the target FPS when the input rate is greater than the target FPS. Each frame passing through the synchronizer goes directly to the input of the queue, and at this time, the interval (T_a) between each input frames is continuously measured. In the proposed system, by using an integrated queue, it is possible to process data consistently with various inputs, and it is easy to measure the input interval of each frame. Since the queue in Fig. 4 is built into the RTSP module for real-time buffering over the network, using the queue may cause the RTSP function to be lost. Thus, our algorithm allows for more flexible functions by using an additional queue.

The measured T_a is used for frame admission control at the AC connected to the output terminal of the queue along with the detection service time (D_s) measured at the YOLO core.

5.3 Admission controller

Admission controller (AC) enables real-time object detection service in YOLO core through simple yet powerful frame control. As shown in Fig. 8, AC extracts frames from the integrated queue and delivers them to the fetch thread of YOLO core. Admission of each frame in AC is determined by T_a measured by MSP and D_s measured by YOLO detection thread. AC is triggered at the moment the YOLO core finishes processing one frame and fetches the next new frame. The conditions of the frame transmitted to YOLO core whenever AC is executed are as follows. First, when T_a is greater than or equal to D_s, this is generally a situation that operates in a high-performance hardware system, and is a situation where the object detection service time is faster than the input rate, or an initial state ($T_a = D_s = 0$). That is, there is only a frame entered as input while executing from YOLO core in the queue. Therefore, in this condition, AC outputs from the queue immediately

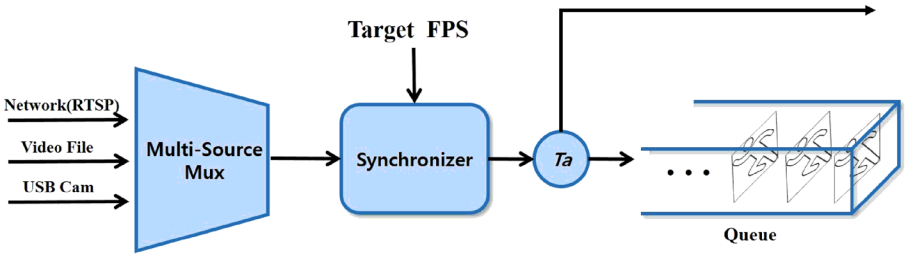

Fig. 7 MSP internal logic

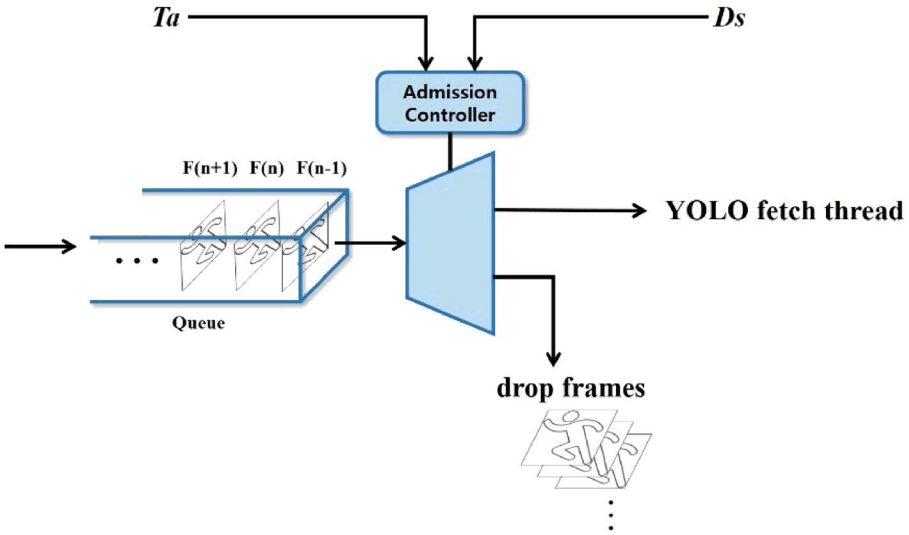

Fig. 8 AC block diagram

without any additional action and delivers it to the YOLO fetch thread. On the other hand, when T_a is less than D_s, this is the case that YOLO is running on relatively low-performance hardware, or the service time of YOLO increases due to insufficient GPU or memory by other processes at an instant or at a specific time. In this case, due to the fast input rate, a number of next frames are stored in the queue while YOLO is serving one frame. We have already seen in Section 3 that the problem of real-time frame processing occurs due to the frames delayed in the queue. Thus, AC drops all remaining frames in the current queue except the frame at the tail of the queue to deliver only the freshest frame to the YOLO core. Here, the number of dropped frames varies depending on the difference between the current T_a and D_s, and in terms of applications, real-time frames can be processed at a rate that converges with the YOLO object detection service time. For example, if the input rate is 30 FPS and the YOLO service rate is 15 FPS, only the latest 15 frames are serviced per second by YOLO, and the remaining 15 frames are dropped. In terms of time, since one frame is entered every 33 ms as input (@30 FPS), the object detection service is executed every 66 ms as a result of the dropped frame by AFC. During this time period, it is almost impossible for an object to appear or disappear on the screen. In the end, the sacrifice of these dropped frames results in real-time guarantees, and it enables cost-efficient system design by enabling selection of appropriate hardware suitable for the characteristics of the application.

5.4 Real-time capability of YOLO with AFC

As already mentioned in Section 3, the existing YOLO has a problem in real-time processing when the arrival rate of input frames is faster than the YOLO service rate. In this section, we investigate how AFC can guarantee real-time processing of each frame. Figure 9 shows an example of real-time frame processing in AFC. As shown in the figure, AFC does not perform YOLO detection service for all input frames, but processes only the latest frame. Therefore, the total service delay for frames actually processed is as follows.

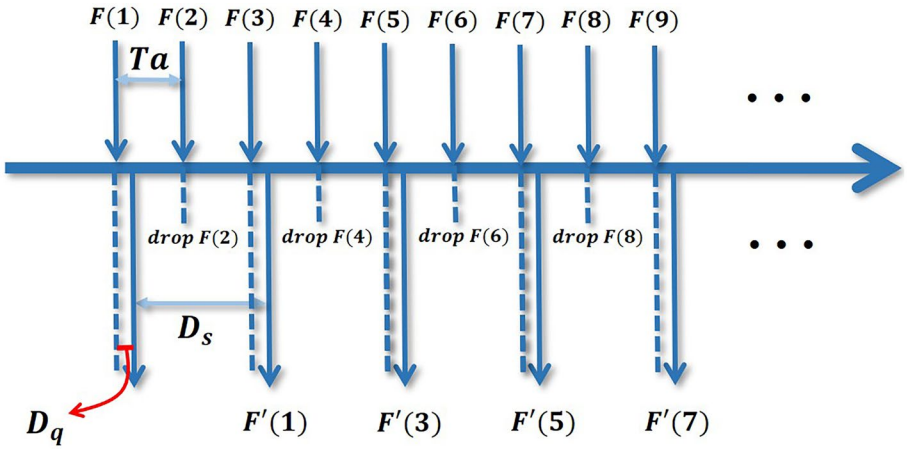

Fig. 9 AFC real-time processing example

$$\mathrm{D(n)} = D_q^n + D_s^n \tag{2}$$

In fact, since AFC processes the latest frame that arrives during the current service execution, it is a relatively small value as $0 \leq D_q^n \leq \frac{T_a}{D_s}(T_a < D_s)$. Therefore, the service time for each frame can be consistent with D_s^n.

Figure 10 shows the comparison of the total service delay in the conventional YOLO and YOLO with AFC when T_a is 30 FPS, YOLO service rate is 15 FPS, and application deadline is 0.3 s. As shown in the figure, in the case of the existing YOLO, the waiting time for the frames stored in the queue is accumulated as the time elapses. However, since the size of the queue is actually finite, the delay converging to the maximum queue size continues to occur. On the other hand, since AFC only serves the latest frame, this cumulative delay does not occur, and it shows that each frame can be processed within the deadline.

6 Experimental results

In this Section, rate of almost 450 ms are presented and various performances of the proposed system on various platforms are verified. Since AFC is mainly proposed to improve the real-time object detection performance in the edge environment, we experimented with the embedded hardware platforms that are the most used in the edge environment, not the high-performance hardware. Also, to compare the performance, we experimented together in a low-end GPU (GTX1060) environment used in a PC.

6.1 Experimental environments

Table 1 shows AFC experimental environment for comparative experiments with YOLO.

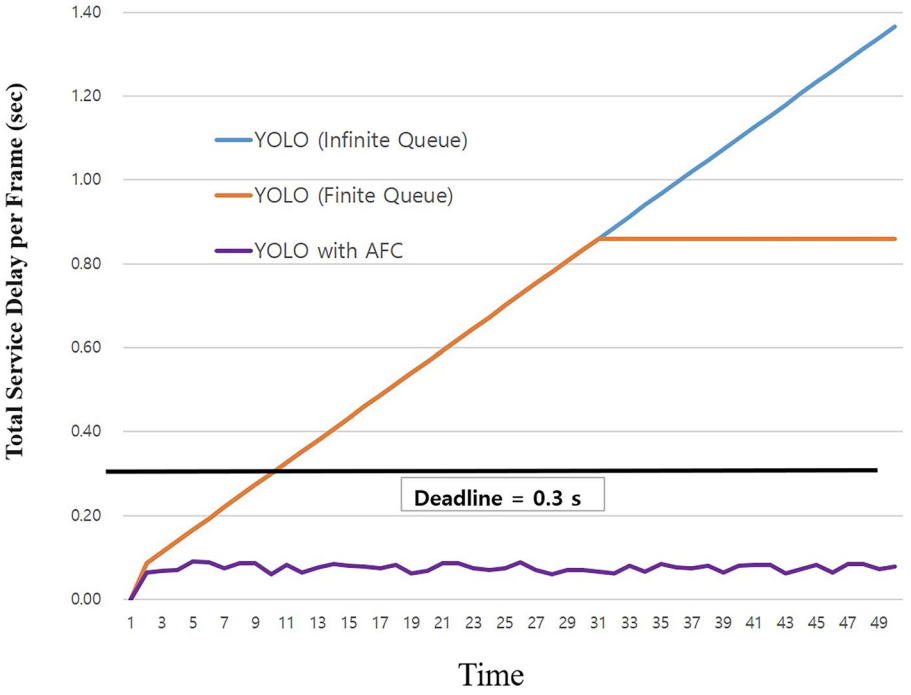

Fig. 10 Total service delay per frame (Numerical analysis)

Table 1 Experimental environment

OS	Linux ubuntu 18.04
Programming environment	C++, OpenCV (4.1.1)
CUDA	Version 10.2
CuDNN	Version 8.0.0
Hardware platform	Jetson Nano, Jetson TX2, Jetson Xavier NX, Jetson AGX Xavier, Laptop with CPU(i7-8750H), Memory (16 GB) and GPU(GTX1060),
Object detector	YOLOv3, Tiny YOLO
Input source	IP Camera (RTSP) (640×480, 30FPS) USB cam (H.264, 640×480, 30FPS) Video files (H.264, 640×480, 30FPS)

6.2 Frame interarrival time

Since the frame interarrival time (T_a) plays an important role together with the detection service time in AFC operation, we first measured the interval of frames coming from individual input sources (RTSP, Video files, and USB Cam). Table 2 shows the measurement results of T_a coming from each of the three input sources on various hardware platforms. Also, to confirm the dependency between YOLO core and T_a, we also observed the change in T_a when changing the input size of YOLO. YOLO's T_a was measured as the frame fetch

Table 2 Average frame interarrival time

Input size	Video source	NANO		TX2		NX		AGX		GTX 1060	
		YOLO	AFC	YOLO	AFC	YOLO	AFC	YOLO	AFC	YOLO	AFC
128×128	RTSP(30fps)	113.19	33.33	53.02	33.33	36.27	33.33	33.33	33.33	33.31	33.33
	Video file	115.72	32.96	53.92	32.95	36.35	32.75	19.84	33.06	16.63	33.76
	USB CAM	122.18	33.39	52.90	33.33	36.37	33.33	33.33	33.34	33.27	33.37
256×256	RTSP(30fps)	294.72	33.33	124.95	33.33	57.39	33.34	30.84	33.33	33.34	33.32
	Video file	295.48	33.10	125.96	32.92	57.44	32.70	31.00	33.06	20.75	33.58
	USB CAM	300.14	33.33	124.82	33.33	57.99	33.33	33.31	33.33	33.31	33.37
416×416	RTSP(30fps)	852.89	33.33	362.07	33.33	103.38	33.33	59.11	33.33	35.63	33.33
	Video file	853.44	33.04	363.19	32.90	105.17	32.85	59.11	32.99	36.52	33.50
	USB CAM	856.83	33.33	363.15	33.33	106.92	33.33	58.92	33.33	38.83	33.36
608×608	RTSP(30fps)	1458.80	33.20	559.84	33.33	176.74	32.33	94.91	33.33	65.99	33.34
	Video file	1443.87	33.08	559.44	32.89	177.60	32.93	93.04	32.98	64.88	33.00
	USB CAM	1453.35	33.33	560.80	33.33	178.95	33.33	92.46	33.33	65.62	33.34

interval from each input source. In this experimental result, it is remarkable that in the case of YOLO, the value of T_a varies greatly depending on the kind of input source and the hardware on which YOLO is running. From the results, in the case of the input size of 608×608 in Jetson Nano, the T_a value from RTSP input source shows a very slow rate of almost 1450 ms., whereas in the GTX1060, the T_a value from the same input source at the same input size is 65.99 ms. This difference is because YOLO's fetch thread is processed as an atomic operation along with the detection thread, so fetch of the next frame waits until detection processing of one frame is completed. Therefore, in YOLO, since there is no function of actual internal input frame control, T_a is dependent on detection service time. In particular, in the case of the RTSP input source, frames transmitted in real time according to the preset FPS (generally 30 FPS) are stored in the receive queue inside the RTSP client. However, if the rate at which the frame is fetched from YOLO is less than 30 FPS, frames that are continuously being received from RTSP are stored in RTSP queue, and eventually each frame is delayed by the time waiting in the queue, resulting in a serious problem.

On the other hand, the AFC has a dedicated MSP that continuously fetches frames from each input source and stores them in the AFC queue regardless of the detection service status, so it can fetch frames independent of the detection service. In addition, when the fetch speed of the input source is faster than the target FPS, a consistent T_a suitable for the target FPS can be maintained by the synchronizer inside the MSP. As a result, as shown in the experimental results, AFC can maintain a consistent T_a that fits the target FPS (30 FPS) from all input sources regardless of the platform on which YOLO is running.

6.3 Performance of detection service

To evaluate the performance of the detection service, we first measured the detection service time (D_s) of YOLO running on various hardware. D_s is the time it takes to complete detection of all available objects in one frame. Table 3 shows the D_s measurement results in each system for various input sizes. For the comparative experiment, we observed the difference in D_s of each of YOLOv3, AFC, and Tiny YOLO. Tiny YOLO, which has been lightened by reducing the complexity of the neural network in YOLO, shows the fastest service speed in all experiments.

As shown in Table 4, we also measured the average FPS of the serviced frames. As in the D_s result, it was confirmed that the FPS of Tiny YOLO shows a high processing speed of 24.5 FPS for 128×128 input size even in the Jetson Nano, which has the lowest performance. The two experimental results also show that there is little difference in the performance of the average detection service time and average FPS of YOLO and AFC. This is because AFC is designed to utilize the detection performance of the existing YOLO as it is, by minimizing the modification of the YOLO core.

According to the two experiments described above, Tiny YOLO shows the best performance in terms of D_s and FPS. However, the more important consideration in an actual object detection application is the accuracy of object detection. Thus, we performed comparative experiments on the mean average precision (mAP) of these three techniques. We used over 40,000 images from the MS COCO 2014 (Valid) dataset to measure the average mAP of each technique for different image sizes at the intersection of union (IoU) of 0.5. The experimental results of mAP are shown in Table 5. Since AFC uses YOLO core as it is, the actual mAP measurement result is the same as YOLO. Also, through experiments, it was confirmed that even if YOLO is running on different hardware, mAP shows the same results. As shown in the experimental results, it can be seen that the performance of mAP

Table 3 Average detection service time

Size	Nano			TX2			NX			AGX			GTX1060		
	YOLOv3	AFC	Tiny	YOLOv3	AFC	Tiny	YOLOv3	AFC	Tiny	YOLOv3	AFC	Tiny	YOLOv3	AFC	Tiny
128×128	112.79	112.66	42.06	52.36	51.93	17.65	35.69	35.62	6.11	20.05	20.04	3.75	12.51	12.49	2.61
256×256	295.05	295.15	64.10	125.23	124.98	29.69	57.60	57.83	8.92	30.69	30.73	4.74	17.07	16.99	3.11
416×416	847.74	847.14	175.87	362.28	362.02	79.25	102.30	102.15	17.53	58.16	57.92	8.39	33.84	33.58	5.37
608×608	1441.40	1442.16	349.87	558.77	558.32	145.64	170.77	169.98	30.14	92.14	92.72	14.87	61.82	61.24	9.93

Table 4 Average FPS

Size	Nano			TX2			NX			AGX			GTX1060		
	YOLOv3	AFC	Tiny	YOLOv3	AFC	Tiny	YOLOv3	AFC	Tiny	YOLOv3	AFC	Tiny	YOLOv3	AFC	Tiny
128×128	8.90	8.90	24.50	19.10	19.10	30.00	27.50	27.50	30.00	30.00	30.00	30.00	30.00	30.00	30.00
256×256	3.40	3.40	15.50	8.10	8.10	30.00	17.40	17.10	30.00	30.00	30.00	30.00	30.00	30.00	30.00
416×416	1.20	1.20	5.70	2.80	2.80	12.50	9.60	9.50	30.00	16.90	16.90	30.00	28.10	27.80	30.00
608×608	0.70	0.70	2.90	1.80	1.80	6.90	5.60	5.60	30.00	10.60	10.60	30.00	15.20	15.10	30.00

Table 5 Mean average precision comparison (mAP@IoU = 0.50)

H/W	NANO		TX2		NX		AGX		GTX1060	
Size	YOLO	Tiny	YOLO	Tiny	YOLO	Tiny	YOLO	Tiny	YOLO	Tiny
128×128	30.47	3.30	30.47	3.30	30.47	3.30	30.47	3.30	30.47	3.30
256×256	48.23	11.89	48.23	11.89	48.23	11.89	48.23	11.89	48.23	11.89
416×416	54.81	16.06	54.81	16.06	54.81	16.06	54.81	16.06	54.81	16.06
608×608	54.25	16.27	54.25	16.27	54.25	16.27	54.25	16.27	54.25	16.27

varies depending on the input image size. In other words, this shows that more precise object detection is possible through higher resolution images. Above all, the most notable point in this experiment is that the mAP performance of tiny YOLO, which showed the best performance in the previous two experiments (Average detection service time and Average FPS), was considerably lower than that of YOLO or AFC. It can be seen that even Tiny YOLO's mAP for 608×608 input size shows almost half the performance of the 128×128 size YOLO. We have confirmed that tiny YOLO has many false and undetected objects from the results of checking in the actual video. On the other hand, YOLO and AFC, which shows a very high mAP value, confirmed that object detection in a frame is very well performed even in a relatively small (128 X 128) input image.

In fact, most intelligent video applications should perform object detection autonomously without human intervention. Thus, although high-speed object detection service is possible even in low-spec systems, it is not desirable for these applications to use Tiny YOLO with low mAP. In addition, in this experiment, it was shown that AFC almost maintains the same performance as the existing YOLO.

6.4 Real-time ability

Through the experiments in Sections 6.2 and 6.3, it has been confirmed that AFC can consistently fetch frames from various sources, while still maintaining YOLO's high object detection performance. Thus, in this experiment, the performance of real-time capability for object detection, which is the most important design goal of AFC, is analyzed and compared with YOLO. As already confirmed by the previous experimental results, Tiny YOLO enables fast FPS even on low-end hardware, but due to its low precision, it is not considered as an object detection module in real-world applications. Therefore, Tiny YOLO was excluded in this experiment.

In this experiment, an IP camera was installed in our laboratory for the same environment as a real object detection application, and the video transmitted through RTSP was received in real time by each system and object detection was performed. In order to check the real-time processing of each frame, we continuously observed the time when object detection for each frame was completed, and displayed the processed frames including current timestamp and frame sequence. YOLOv3 and AFC were respectively executed and compared on various hardware platforms, and the effect of each technique on real-time processing at different input sizes was examined.

Figure 11a–e shows the total service delay of each frame received by RTSP on different hardware. First of all, the GTX 1060 is the system that shows the best performance among the hardware we tested. In Fig. 11a, it can be seen that 'YOLO 128' and

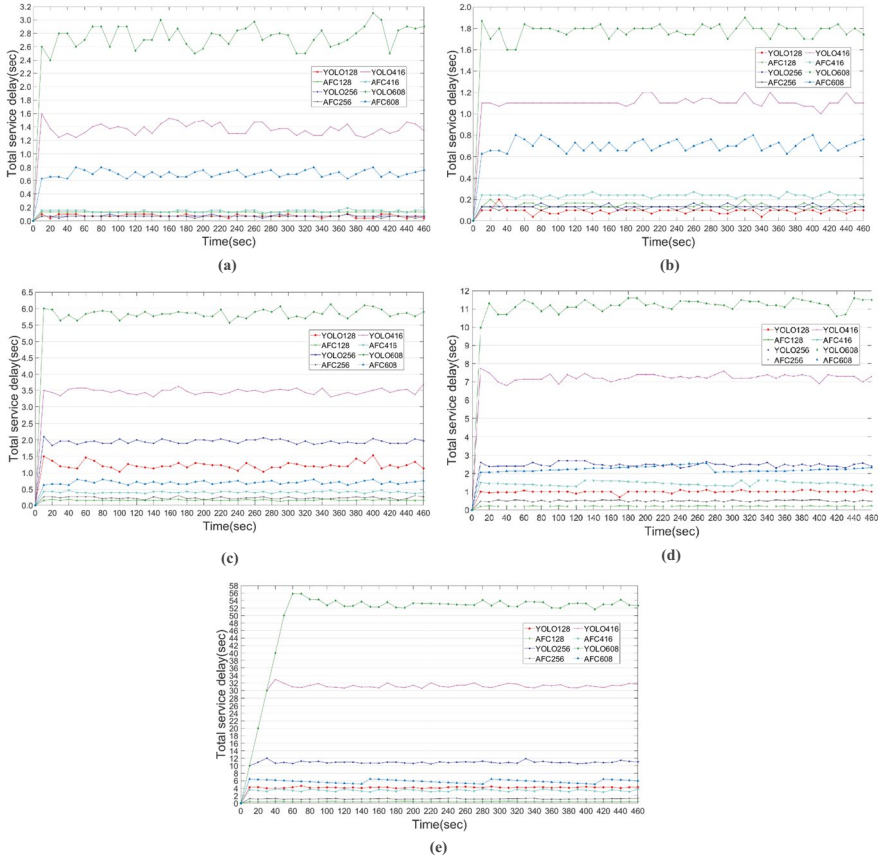

Fig. 11 Total service delay per frame. **a** GTX 1060. **b** Jetson AGX Xavier. **c** Jetson Xavier NX. **d** Jetson TX2. **e** Jetson Nano

'YOLO 256' are processed with a delay of less than 0.1 s. As already seen in Table 4, YOLO running on the GTX 1060 shows 30 FPS performance at both 128×128 and 256×256 image sizes. Since the input frame coming from the RTSP source has a rate of 30 FPS, it shows that all the input frames are processed in real time. However, when the same hardware handles inputs of larger image sizes (416×416 or 608×608), the FPS performance decreases to 28.10 and 15.20, respectively. Nevertheless, if we only saw the object detection result video without direct comparison with the original image, we would not be able to confirm the exact difference with our eyes. However, we confirmed this difference through direct comparison with the original video. In Fig. 11a, the total service delay of 'YOLO 416' and 'YOLO 608' is about 1.4 and 2.8 s, respectively. This result shows that the event or object currently viewed by the actual camera is detected after 1.5 to 2.8 s on this system. That is, this result proves that real-time processing of events from RTSP input sources cannot be guaranteed in the case of FPS lower than the object detection speed of 30 FPS. However, in the case of AFC, it can be seen that the service is performed with a delay of 0.3 s or less in 'AFC 608'. That is, it proved that real-time processing of AFC can be guaranteed despite the same hardware platform, the

same input conditions, and the same FPS as YOLO. It also can be seen that the problem of such service delay becomes more serious as it goes down to the low-end hardware in Fig. 11b–e. Of course, as the hardware specification decreases, the AFC delay increases accordingly. Nevertheless, the following interesting results can be obtained in this experiment. For example, if an application with an object detection deadline of 0.5 s performs object detection with an image size of 415 X 415 using YOLO, a laptop with a GTX 1060 GPU cannot meet the requirements. Accordingly, another system with better performance should be considered. However, when AFC is used, the application requirements can be satisfied even with the Jetson NX platform under the same conditions as shown in Fig. 11c. In this case, comparing the prices of Jetson NX and GTX 1060 or better-performing systems, the application can compose a system with NX at a lower price of at least 1/6. In addition, in an application with a looser deadline (for example, deadline = about 1 s), if the performance for an image size of 416 is required, AFC can even select a TX2 board as shown in Fig. 11d. On the other hand, in the case of the existing YOLO under the same conditions, a system beyond AGX as shown in Fig. 11b should be considered. In addition, if the requirements for the image size become lower (128×128 or 256×256), as shown in Fig. 11e, even the Jetson Nano can be considered with AFC. On the other hand, in the case of the existing YOLO, it can be seen that the delay occurs quite seriously. The reason for this difference in real-time performance is because the existing YOLO overlooked the problem of the difference between the input rate and the service rate that may occur in a system with hardware limitations. However, AFC does not depend on the buffering of RTSP by consistently controlling the input rate through MSP and using a separate integrated queue. In addition, since frames are adaptively controlled according to the input rate and service rate through AC, real-time processing capability is greatly improved. As such, this experimental result demonstrates that AFC's real-time object detection service capability for RTSP input sources in a system with hardware limitations is significantly superior to that of conventional YOLO.

Figure 12 shows the comparative result of the actual RTSP video processing of YOLO and AFC in GTX1060 (416×416), NX (416×416), and Nano (256×256). The images at the top of the figure are Ground Truth (GT), which is the original video, and real-time images displayed by each system and algorithm were captured and compared at intervals of 1 min and 30 s based on the GT. As shown in the figure, YOLO and AFC in each system perform object recognition well and display the corresponding bounding box. In the case of GTX1060, even at the same image size, AFC shows only 170 ms of delay and processes almost the same time frame as GT, while YOLO displays about 1.5 s delayed video. As a result, the person who appeared in the original video, like the photo at the moment of 4` 30``, has not yet appeared on the YOLO display at that moment. In the case of AFC, Jetson NX also shows that it is possible to perform near real-time processing similar to GT, while in YOLO, it is seen that scenes completely different from ones in the original video appear due to the delay of almost 3 min and 50 s. As shown in Fig. 11e, even if AFC is applied to NANO, it is difficult to apply it to real-time object detection applications because a delay of 3 s or more occurs at a size of 416 or more. Nevertheless, if the image size 256 is applied to the NX, which is the lowest specification of the hardware used in the experiment, the object detection service time of NANO AFC is closer to real time than when general YOLO is applied to GTX1060 and NX. Through this experiment, it was verified that when a network camera as an input is interfaced with YOLO, significant processing delay occurs depending on the performance of the hardware, and AFC has proven that it has improved YOLO's real-time capability by efficiently solving this problem.

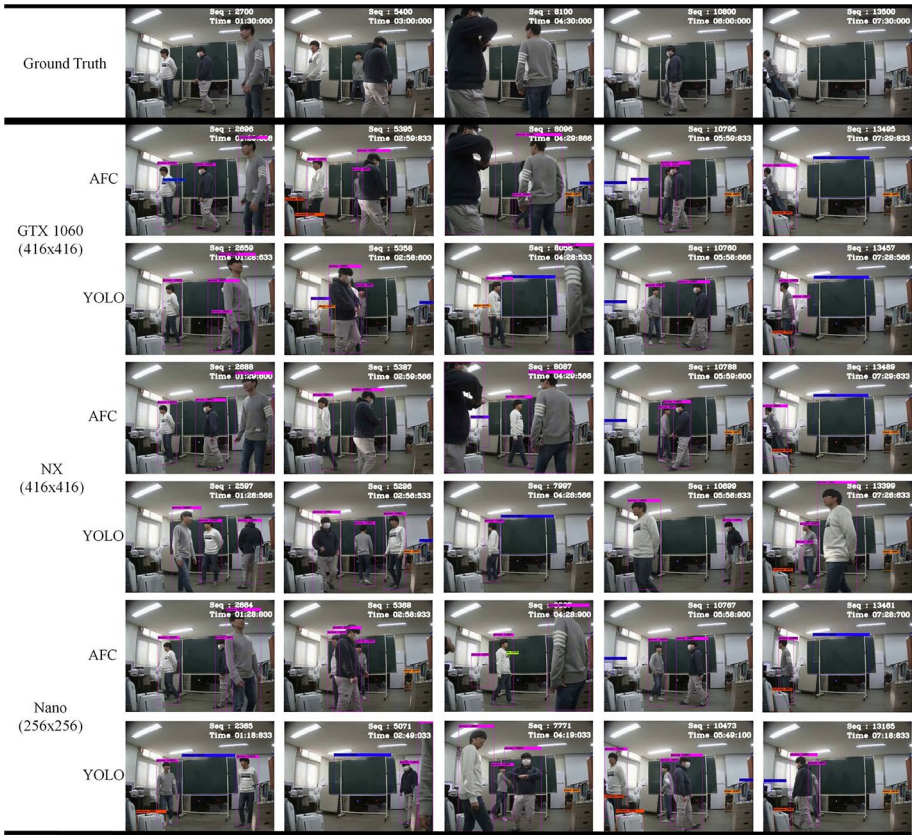

Fig. 12 Photos captured during object detection processing of video input from RTSP network camera source

7 Conclusion

In this paper, we raised the problem of real-time processing of YOLO. In particular, we pointed out problems that may occur in network camera-based object detection applications, and observed and analyzed various performances of YOLO on various hardware platforms that are widely used in artificial intelligence applications. To solve this problem, we proposed an AFC that can improve real-time capability while maintaining the convenience and object detection performance of the existing YOLO. Through various experiments, we proved that the proposed AFC shows superior performance by effectively coping with the problem of real-time delay occurring in the existing YOLO. Above all, this paper derives practical requirements for object detection-based application design, unlike previous studies that focused on changing the structure of the YOLO network itself, and proposed practical solutions to solve these problems. Therefore, the major contribution of this study is to provide efficient guidelines for appropriate hardware platform selection according to requirements of YOLO-based intelligent video application.

Recently, YOLOv4 and YOLOv5 have been released, but all of them are mainly focused on improving the precision performance of object detection. Compared to YOLOv3, the

new versions of YOLO are slightly improved in terms of mAP, but the problem of cumulative delay arising from RTSP input in resource-constrained platforms is still not considered in any version. In addition, the proposed AFC runs as an independent unit at the front end of the YOLO object detection core, so it is compatible with all versions of YOLO. That is, even though we experimented AFC with YOLOv3 which is the most popular and stable version, it can be also applied to YOLOv4 and YOLOv5. Finally, AFC is expected to be used as an efficient add-on package for real-time processing not only in YOLO but also in various object detection algorithms to be proposed in the future.

Acknowledgements This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1F1A1054896)

Funding This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) (No. NRF-2019R1F1A1054896).

Declarations

Conflict of interest The Authors declares that there is no conflict of interest.

References

1. Barry D et al (2019) xYOLO: a model for real-time object detection in humanoid soccer on low-end hardware. In: 2019 International Conference on Image and Vision Computing New Zealand (IVCNZ). IEEE
2. Barthélemy J et al (2019) Edge-computing video analytics for real-time traffic monitoring in a smart city. Sensors 19(9):2048
3. Bochkovskiy A, Wang CY, Liao HYM (2020) YOLOv4: optimal speed and accuracy of object detection. Accessed https://arxiv.org/abs/2004.10934
4. Cao Z et al (2020) Detecting the shuttlecock for a badminton robot: a YOLO based approach. Expert Syst Appl 164:113833
5. Chen S, Wei L (2019) Embedded system real-time vehicle detection based on improved YOLO network. In: 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). IEEE
6. Dalal N, Triggs B (2005) Histograms of oriented gradients for human detection. In: 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05) (Vol. 1, pp 886–893). IEEE
7. Fang W, Lin W, Peiming R (2019) Tinier-YOLO: a real-time object detection method for constrained environments. IEEE Access 8:1935–1944
8. Fikri RM, Byungwook K, Mintae H (2020) Waiting time estimation of hydrogen-fuel vehicles with YOLO real-time object detection. Information science and applications. Springer, Singapore, pp 229–237
9. Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR'14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition. pp. 580–587

10. Gour D, Amit K (2019) Optimised YOLO: algorithm for CPU to detect road traffic accident and alert system. Int J Eng Res Tech 8:160–163
11. He W et al (2019) TF-YOLO: an improved incremental network for real-time object detection. Appl Sci 9(16):3225
12. Jamtsho Y, Panomkhawn R, Rattapoom W (2020) Real-time Bhutanese license plate localization using YOLO. ICT Express 6(2):121–124
13. Jetson AGX Xavier. Accessed https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems/jetson-agx-xavier/
14. Jetson Nano. Accessed https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems/jetson-nano/
15. Jetson TX2. Accessed https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems/jetson-tx2/
16. Jetson Xavier NX. Accessed https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems/jetson-xavier-nx/
17. Jin Y, Yixun W, Jingting L (2020) Embedded real-time pedestrian detection system using YOLO optimized by LNN. In: 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE). IEEE
18. Kalhagen ES, Ørjan LO (2020) Hierarchical fish species detection in real-time video using YOLO. MS Thesis. University of Agder
19. Kim S, et al. (2019) Spiking-yolo: spiking neural network for real-time object detection. Accessed https://arxiv.org/abs/1903.06530
20. Lowe G (1999) Object detection from local scale-invariant features. In: The proceedings of the seventh IEEE international conference on Computer vision, 1999. Vol. 2, pp. 1150–1157. IEEE
21. Lu S et al (2019) A real-time object detection algorithm for video. Comput Electr Eng 77:398–408
22. Mohd P, Nurul PA (2020) A real-time traffic sign recognition system for autonomous vehicle using Yolo. Diss. Universiti Teknologi MARA, Cawangan Melaka
23. Oltean G et al (2019) Towards real time vehicle counting using yolo-tiny and fast motion estimation. In: 2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME). IEEE
24. Papageorgiou CP, Oren M, Poggio T (1998) A general framework for object detection. In: Sixth international conference on Computer vision, 1998. IEEE. pp. 555–562
25. Raspberry PI 4. Accessed https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
26. Redmon J, Farhadi A (2017) YOLO9000: better, faster, stronger. In: Computer Vision and Pattern Recognition (CVPR). Vol. 1, pp. 6517–6525
27. Redmon J, Farhadi A (2018) YOLOv3: an incremental improvement. Technical Report. Accessed https://arxiv.org/abs/1804.02767
28. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: Unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Vol.1, pp. 779–788
29. Ren P et al (2020) A novel squeeze YOLO-based real-time people counting approach. Int J Bio Inspir Comput 16(2):94–101
30. Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. Adv Neural Inf Process Syst 39:1137–1149
31. Schulzrinne H, et al. (1998) RFC 2326, Real time streaming protocol (RTSP). Accessed https://tools.ietf.org/html/rfc2326
32. Shi R, Tianxing L, Yasushi Y (2020) An attribution-based pruning method for real-time mango detection with YOLO network. Comput Electron Agric 169:105214
33. Silva P, Paul A (2019) Implementing YOLO algorithm for real time object detection on embedded system. BS Thesis. Universidad de Investigación de Tecnología Experimental Yachay
34. Ullah MB (2020) CPU based YOLO: a real time object detection algorithm. In: 2020 IEEE Region 10 Symposium (TENSYMP). IEEE
35. Wang J et al (2020) Real-time behavior detection and judgment of egg breeders based on YOLO v3. Neural Comput Appl 32(10):5471–5481
36. Wang Z et al (2020) Sparse-YOLO: hardware/software co-design of an FPGA accelerator for YOLOv2. IEEE Access 8:116569–116585
37. Zhou Z et al (2019) Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proc IEEE 107(8):1738–1762