2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS) | 979-8-3503-7222-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICETSIS61505.2024.10459422

AFAR-YOLO: An Adaptive YOLO Object Detection Framework

[1]Ainal Irham, [1]Kurniadi, [1]Khoirinisa Yuliandari, [1]Farhan Mozart Aditya Fahreza, [1]Daffa Riyadi, [1]Ary Mazharuddin Shiddiqi
Informatics Department, [1]Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
6025231006@student.its.ac.id, 6025231002@student.its.ac.id, 6025231017@student.its.ac.id, 6025231052@student.its.ac.id, 6025231014@student.its.ac.id, ary.shiddiqi@its.ac.id

***Abstract*—This study focuses on developing an advanced early warning system utilizing YOLOv5 to detect objects indicative of potential fire hazards. This research is motivated by the fact that continuous monitoring is impractical, especially in high-risk and inaccessible areas. We introduce an innovative approach: adaptive YOLO for object detection to enhance early fire detection capabilities in these challenging environments. The main contribution of this research is the development of adaptive frames per second (FPS) resolution in YOLO object detection. We found that implementing adaptive FPS alone does not significantly impact the efficiency of CPU and RAM resources in the tested devices. However, when adaptive FPS is combined with adaptive resolution, resource usage is significantly reduced—specifically, a 33% decrease in CPU usage and a 0.5-1% (200-400 MB) reduction in RAM usage. These efficiency gains are important in enhancing safety in the industrial sector.**

***Index Terms*—Adaptive YOLO, Adaptive FPS, Adaptive Resolution, Adaptive Object Detection, Object Detection**

I. Introduction

You Only Look Once (YOLO) leverages deep learning to accurately identify and pinpoint visual objects from specific classes, as highlighted in [1] and [2]. Its unique approach involves utilizing a single neural network to simultaneously predict bounding boxes and class probabilities for each object in one pass, a technique detailed in [1] and [2]. Data from BNPB data shown the reports 462 land and forest fire cases in 2023 across four islands, with Sumatra experiencing the highest incidence at 417 cases. This number peaked in 2023, marking an alarming trend over the past three years. Those data underscores the urgency of this technology.

Continuous monitoring of high-risk buildings or limited access presents significant challenges. Therefore, early fire detection becomes crucial. High-accuracy fire object detectors play a vital role in minimizing fire-related losses. Image processing and computer vision techniques, including those based on YOLOv5s, are pivotal in this regard, as they offer early detection, high accuracy, and flexibility, as noted in [3]. However, Continuous object detection requires significant computational resources. Optimizing the balance between efficiency and accuracy by adapting FPS and resolution is crucial to enhancing object detection capabilities. This concept addresses resource constraints while maintaining performance in varying visual and computational environments.

Our approach introduces a novel adaptive FPS and resolution mechanism that intelligently adjusts the frames per second (FPS) and resolution parameters in response to the presence or absence of detectable objects. This method optimizes resource usage, balancing performance and computational load. When the system detects objects, it can increase the resolution and FPS to ensure accuracy and detail in object detection. Conversely, in the absence of detectable objects, it reduces these parameters to save on computational resources. This dynamic adjustment is central to the efficacy of our system, ensuring that it maintains high performance in object detection while being resource-efficient.

II. Related Works

The effectiveness of the YOLO object detection method is explored in various studies, each focusing on different aspects and applications. Research in [4] demonstrated that YOLO achieves an impressive average accuracy of 90% and an average loss value below 0.313. These results underscore YOLO's potential for accurate fire detection. Complementing this, the work in [5] indicated that YOLO could detect fire with a confidence level of up to 0.8, further validating its effectiveness in this domain.

Another study [6] investigated combining YOLO with various configurations of the moving image feature to recognize objects, such as vehicles. This approach, however, highlights the challenge of needing substantial computational resources and extended processing times. The researchers found that reducing resolution and frame rate affects CPU usage (averaging 95.51%) but can significantly decrease processing time by up to 70%.

Research by Liluo et al. [7] focused on optimizing kernel operations for higher power efficiency. Their CNN-based FPGA accelerator, evaluated on the Intel Arria 10 FPGA, achieves a remarkable 740 Giga operations per second (GOPS) at 200 MHz, with a kernel power of 12.2 watts. The system can perform object detection tasks at 105 fps with 56.5 mAP or 25 fps with 73.6 mAP on the VOC dataset, boasting greater power efficiency than traditional GPU and CPU setups.

Anj et al. [8] proposed an Efficient Residual Bottleneck for real-time object detection, improving the existing CSP bottleneck. Trained on the MS COCO dataset, their model achieved a 0.367 mAP, outperforming the RefinedeLite's 0.296 mAP and attaining a higher FPS of 23.010. The most recent related research was by Ullah [9], which introduced a CPU-based YOLO model tailored for non-GPU computers. This optimization allows real-time object detection on lower-end computers, achieving 10.12 - 16.29 FPS in video object detection, making YOLO accessible to users with less advanced hardware.

The study in [10] introduced a novel multiscale domain adaptation framework (MS-DAYOLO) that enhances the widely used real-time object detector YOLO. This framework allows YOLO to adapt to different target domains without annotation. Experimental results demonstrated that the MS-DAYOLO architecture successfully adapts YOLO to new domains and surpasses the performance of the advanced YOLOv4 and various other methods based on the Faster R-CNN object detector.

The previous work by Ary et al. [11] introduces a novel system that elaborates an object detection algorithm based on deep learning and a resource-aware framework (named the RAViS framework) to recognize objects from continuous images taken by IP cameras, which are regarded as data streams. Keeping resources available throughout the object detection process is the goal of the resource-aware framework integration. In order to ensure that the object detection process continues without interruption, the RAViS framework is in charge of managing the RAM, CPU, and storage.

III. Methodology

This research involves a methodical approach starting with model selection based on optimal K-Means clustering results (Figure 1). This research focuses on implementing adaptive FPS and resolution within the YOLO object detection framework to enhance resource utilization efficiency (Algorithm 1).

A. Object detection using YOLO

YOLO employs a unique approach where a single neural network predicts bounding boxes and class probabilities for objects within an image. After processing the entire image, this process results in the simultaneous output of all detected objects, each with its corresponding class probabilities and bounding boxes. The network segments the input image into an SxS grid. Each grid cell is responsible for projecting a set number of bounding boxes. The number of classes to be detected determines the quantity of bounding boxes.

This study uses the YOLOv5s model and performs a comparative analysis involving two devices of varying capabilities, i.e. a Raspberry Pi and a laptop. The YOLOv5s model is selected for its compatibility with less powerful devices, providing a viable alternative to the YOLO tiny model. This study ensures that the model runs efficiently even on hardware with limited computational resources.

Algorithm 1 Adaptive FPS + Resolution YOLO Object Detection

```
 1: procedure MAIN
 2:     INITIALIZE mixer
 3:     INITIALIZE siren_sound using 'fire alarm.mp3'
 4:     INITIALIZE YOLOv5 model
 5:     INITIALIZE cap
 6:     SET initial resolution of cap to 1920x1080
 7:     INITIALIZE variables: skip_frames_no_obj, skip_frames_obj, counter, latencies, fps_values, delays, times, confidences, elapsed_time
 8:     INITIALIZE resolution_changes list
 9:     while TRUE do
10:         READ frame from cap
11:         $counter \leftarrow counter + 1$
12:         RECORD start_time
13:         RUN model on frame
14:         $latency \leftarrow$ TIME.TIME() - start_time
15:         APPEND latency to latencies
16:         UPDATE elapsed_time
17:         APPEND elapsed_time to times
18:         EXTRACT show and filter results
19:         CHECK check for resolution changes
20:         UPDATE confidences list
21:         HANDLE detection result cases:
22:             - ADJUST skip_frames and play/stop
23:             - CALCULATE fps and _time/frame
24:             - CALCULATE real_time_delay
25:         APPEND real_time_delay and fps
26:         DRAW bounding boxes and frame labels
27:         DISPLAY FPS on frame
28:         SHOW frame and check loop condition
29:     end while
30:     CLOSE windows and release video capture
31: end procedure
```

B. Clustering Using K-Means

We explore various combinations of augmentations to pre-train each model. These pre-trained models are utilized for initial object detection inference. This step is crucial for evaluating the model's effectiveness in detecting objects under different lighting conditions. We assess the models' performance by examining their confidence values in bright and low-light scenarios. The model demonstrating the highest confidence value, derived from its specific augmentation combination, is then selected for further use in object detection.

C. Adaptation Mechanism

1) Adaptive FPS Mechanism
   Our adaptive framework utilizes OpenCV, torch, and others to process real-time video at 1920x1080 resolution. The method sets adaptive

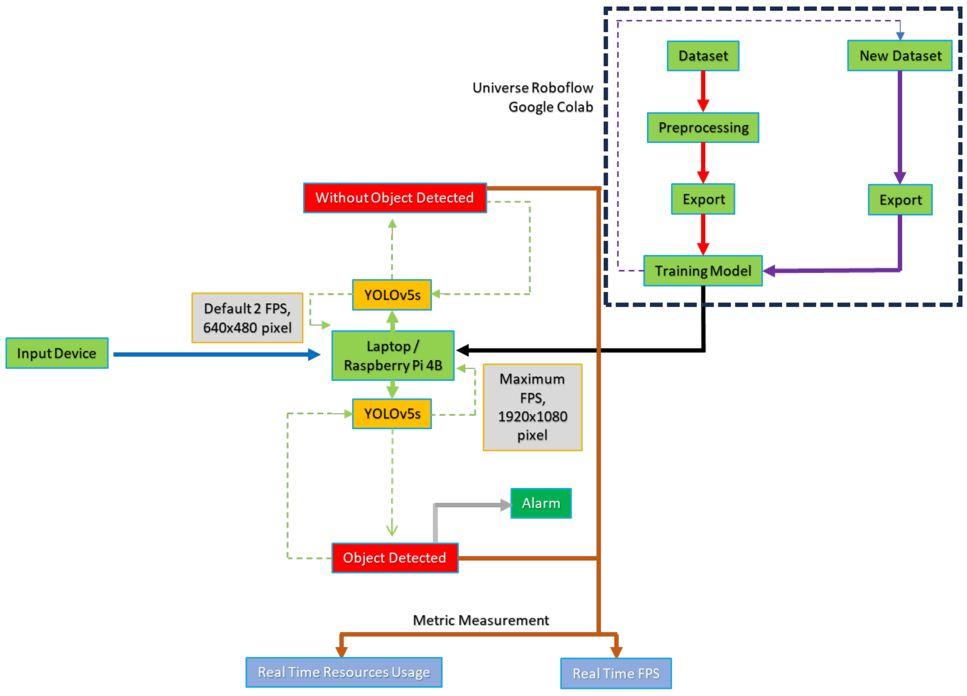


Figure 1: Architecture System Adaptive YOLO Object Detection

frame skipping based on object presence, typically skipping four frames by default and one during detection. It involves a loop function for continuous detection, increasing FPS, drawing bounding boxes upon object detection, and triggering an alarm. If no objects are detected, FPS reverts to its default, conserving resources. This cycle continues automatically until the program ends.

2) Adaptive FPS + Resolution Mechanism
This technique employs the same libraries and YOLO model as the Adaptive FPS Mechanism while introducing new algorithms for adaptive resolution and frame rate. Initially, the video frame is set at 1920x1080 pixels. The video capture object's resolution drops to 640x480 pixels when no objects are detected or confidence values hit zero, reducing FPS to 2 and increasing skip frames to 15. Adaptive YOLO object detection adjusts parameters dynamically based on real-time video analysis. Upon object recognition, the system outlines the object with a bounding box at a minimum 0.5 confidence threshold, escalating the video resolution back to 1920x1080 pixels, minimizing skip frames to 1, and maximizing FPS according to device capacity. Should detection cease, FPS and resolution revert to their initial settings, continuously adapting until further detections.

IV. Experiment And Result

The Yolov5 object detection program runs on a laptop and Raspberry Pi 4B, utilizing a dual-thread process. The first thread handles object detection and measures real-time confidence and FPS values. The other thread monitors resource usage. All data, including detection metrics and resource consumption, are recorded in CSV files and visualized in PNG graphs. The detection of a fire object triggers an alarm system.

Table I: Combination Code

No.	Combination Method	Combination Code
1	One Letter	A (K1), B (K2), C (K3), D (K4)
2	Double Letters	AB (K5), AC (K6), AD (K7), BC (K8), BD (K9), CD (K10)
3	Triple Letters	ABC (K11), ABD (K12), BCD (K13), CAD (K14)
4	Quad Letters	ABCD (K15)
5	Without Augmentation	K16

A. Dataset

Our research commenced with selecting a dataset from a public source, available at https://universe.roboflow.com/nova-3000c/fire-detection-f7a05/model/1. This dataset is selected due to its initial pre-trained results, indicating promising mAP, precision, and recall values of 77.1%, 85.2%, and 71.1%, respectively. It comprises 5623 image files, which we divided into 70% for training (3936 images), 20% for validation (1125 images), and 10% for testing (562 images). The dataset includes four annotation labels; however, we focus solely on labels 2 and 3 concerning fire objects. To determine the most effective augmentation combination, we referenced several comparative studies as detailed in Table I.

Each combination of combinations of augmentation above is written with a letter indicating the presence of the associated technique as shown in Table II.

B. Clustering Using K-Means

Based on K-means clustering results (Figure 2), the K2 combination emerged as the most optimal,

Table II: Letter meanings for augmentation techniques

Letter	Technique used
A	Blur on augmentations level image
B	Noise on augmentations level image
C	Brightness on augmentations level bounding box
D	Exposure on augmentations level bounding box

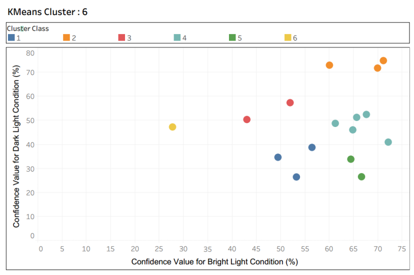


Figure 2: Clustering Using K-Means

achieving the highest confidence values in both bright (71.1%) and dark light conditions (74.9%).

C. Training Model

Before model training, a preliminary experiment demonstrated the most effective batch size among batch 1, 16, 32, 64, and 80, with 80 being the upper limit for the Google Colab platform's 15 GB GPU memory. The experiment revealed batch 16 as the optimal choice, demonstrating initial performance metrics of 0.583 Precision, 0.257 Recall, 0.254 mAP50, and 0.0959 mAP50-95.

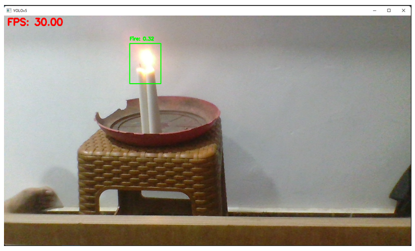
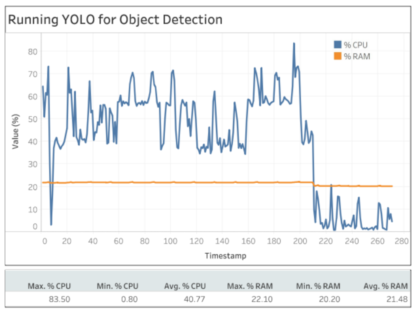


Figure 5: Running YOLO With Object Detected

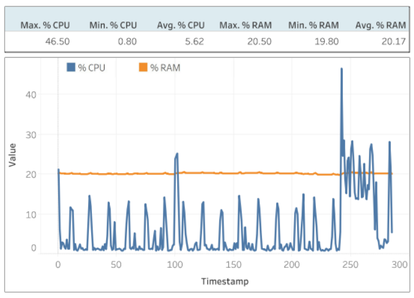


Max. % CPU	Min. % CPU	Avg. % CPU	Max. % RAM	Min. % RAM	Avg. % RAM
83.50	0.80	40.77	22.10	20.20	21.48

Figure 6: Running YOLO With and Without Object Detected

D. Testing on Laptop (Adaptive FPS + Adaptive Resolution)

The testing phase starts with initializing resources before launching the YOLO program. In this initial state, the average CPU utilization is recorded at



Max. % CPU	Min. % CPU	Avg. % CPU	Max. % RAM	Min. % RAM	Avg. % RAM
46.50	0.80	5.62	20.50	19.80	20.17

Figure 3: Initialisation Resources Before Running YOLO Adaptive FPS + Adaptive Resolution

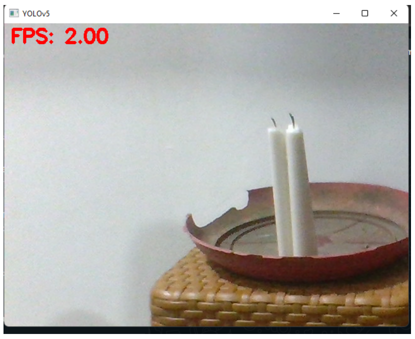
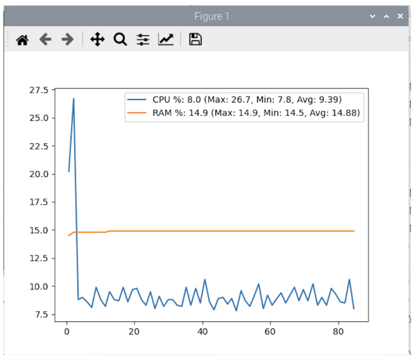


Figure 4: Running YOLO Without Object Detected

5.49%, and the RAM usage stands at 20.17% (Figure 3). Figure 4 indicates the Frames Per Second (FPS) value is set to 2 FPS when running YOLO without detecting objects. If an object is detected, the FPS value increases to the maximum rate the device can handle. Conversely, the FPS value escalates to 30 FPS upon object detection (Figure 5). The required CPU resources increase when an object is detected with specific settings, such as maximum FPS and a resolution of 1920 x 1080 pixels. (Figure 6), when the YOLO software operates without detecting objects, the average CPU utilization is 40%, and RAM usage is 21.5%. Upon object detection, CPU usage increases to approximately 60% and RAM usage to about 22.1%. Implementing the adaptive YOLO method results in a 20% reduction in CPU resource utilization and a 0.6% decrease in RAM usage.

E. Testing on Raspberry Pi 4B (Adaptive FPS + Adaptive Resolution)



CPU %: 8.0 (Max: 26.7, Min: 7.8, Avg: 9.39)
RAM %: 14.9 (Max: 14.9, Min: 14.5, Avg: 14.88)

Figure 7: Before running YOLO

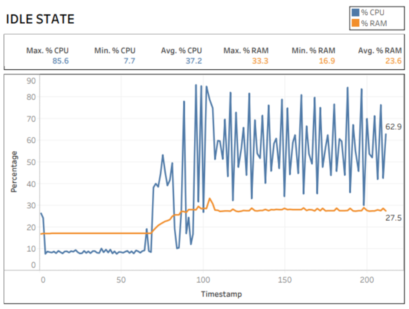


Figure 8: Idle State

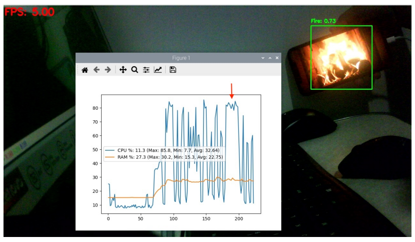

FPS: 0.00
Fire: 0.73
CPU %: 11.3 (Max: 85.8, Min: 7.7, Avg: 32.64)
RAM %: 27.3 (Max: 30.2, Min: 15.3, Avg: 22.75)

Figure 9: Detecting Fire

Before initiating YOLO object detection (Figure 7), the resource usage of the Raspberry Pi exhibits a reasonably flat curve with no significant peaks. This consistent curve is due to the absence of other processes during the testing phase, except for metric recording. The research's testing methodology focuses on implementing adaptive FPS in object detection. In its default state, or when no objects are detected, the FPS is set to a maximum of 2 FPS. Upon activating YOLO for object detection, the resource usage curve becomes noticeably more dynamic, with frequent upward and downward movements. This fluctuation occurs even though the system has not recognized fire objects yet (Figure 8). This phenomenon is due to the increased demand for Raspberry Pi's resources during active object detection (Figure 9 and 10). Due to the constrained resources of the Raspberry Pi, the resulting changes are relatively subtle.

V. Conclusion

Our research demonstrates the effectiveness of a system combining adaptive FPS and resolution. This method significantly reduces CPU usage by approximately 33% and RAM by 0.5-1% (200-400 MB), proving efficient for laptops and Raspberry Pi devices. We also discovered that the K2 augmentation method, which applies only image-level noise, delivers the highest confidence in varying lighting—71.1% in bright and 74.9% in dark conditions. Batch 16 was identified as the optimal value for model training, yielding 0.583 Precision, 0.257 Recall, 0.254 mAP50, and 0.0959 mAP50-95. These findings further enhance the system's applicability and effectiveness across different scenarios.

Acknowledgment

We would like to express our gratitude to Applied Science University in Bahrain (https://www.asu.edu.bh/) for offering free registration to the students who authored this academic paper.

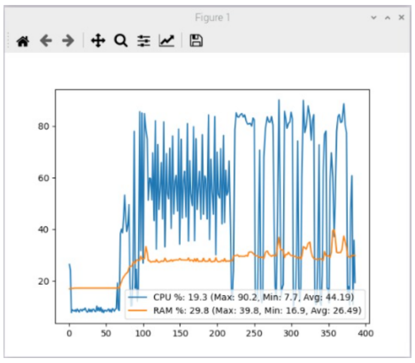


Figure 10: Running YOLO Process

References

[1] Aria Bisma Wahyutama and Mintae Hwang, "YOLO-Based Object Detection for Separate Collection of Recyclables and Capacity Monitoring of Trash Bins," Electronics 2022, Vol. 11, Page 1323, Multidisciplinary Digital Publishing Institute, 2022.

[2] Peiyuan Jiang and Daji Ergu and Fangyao Liu and Ying Cai and Bo Ma, "A Review of Yolo Algorithm Developments," Procedia Computer Science, Elsevier, 2022.

[3] Pu Li and Wangda Zhao, "Image fire detection algorithms based on convolutional neural networks," Case Studies in Thermal Engineering, Elsevier Ltd, 2020.

[4] Dewi PL. and Rifki K. and Tri H. and Murni. and Ilmiyati S. and Achmad F, "Fire Hotspots Detection System on CCTV Videos Using You Only Look Once (YOLO) Method and Tiny YOLO Model for High Buildings Evacuation," IEEE, 2019.

[5] Wanbo Luo, "Research on fire detection based on YOLOv5," IEEE, 2022.

[6] Moh Noor Al-Azam and Mochamad Mizanul Achlaq and Cahyo Darujati, "Moving Image YOLOv3 Process Comparison with Multiple Resolutions and Frames-per-Second," 2022 International Seminar on Intelligent Technology and Its Applications: Advanced Innovations of Electrical Systems for Humanity, ISITIA 2022 - Proceeding, Institute of Electrical and Electronics Engineers Inc., 2022.

[7] Shuai Li and Yukui Luo and Kuangyuan Sun and Nandakishor Yadav and Kyuwon Ken Choi, "A Novel FPGA Accelerator Design for Real-Time and Ultra-Low Power Deep Convolutional Neural Networks Compared with Titan X GPU," IEEE Access, Institute of Electrical and Electronics Engineers Inc., 2020.

[8] Jinsu An and Muhamad Dwisnanto Putro and Kang Hyun Jo, "Efficient Residual Bottleneck for Object Detection on CPU," Proceedings - IWIS 2022: 2nd International Workshop on Intelligent Systems, Institute of Electrical and Electronics Engineers Inc., 2022.

[9] Md Bahar Ullah, "CPU Based YOLO: A Real Time Object Detection Algorithm," 2020 IEEE Region 10 Symposium, TENSYMP 2020, Institute of Electrical and Electronics Engineers Inc., 2020.

[10] Hnewa, Mazin and Radha, Hayder, "Integrated Multiscale Domain Adaptive YOLO," IEEE Transactions on Image Processing, Institute of Electrical and Electronics Engineers Inc., 2023.

[11] Ary Mazharuddin Shiddiqi and Edo Dwi Yogatama and Dini Adni Navastara, "Resource-aware video streaming (RAViS) framework for object detection system using deep learning algorithm," MethodsX., 2023.