International Journal of Enhanced Research in Science, Technology & Engineering
ISSN: 2319-7463, Vol. 14 Issue 5, May-2025

Comparative Analysis of Embedded Neural Network Object Detectors In Terms of Latency Accuracy and Power Efficiency

Ravinder Gaja[1], Dr. Mukesh Tiwari[2]

[1]Research Scholar, Department of Electronics & Communication, Sri Satya Sai University of Technology & Medical Sciences, Sehore, M.P, India
[2]Research Supervisor, Department of Electronics & Communication, Sri Satya Sai University of Technology & Medical Sciences, Sehore, M.P, India

ABSTRACT

Embedded neural networks for object identification have seen a development spike due to the increasing demand for AI applications that can operate in real-time even in contexts with limited resources. This research compares three different hardware platforms' object detection model performance using the COCO2017 validation dataset. The evaluation is centered around three important metrics: efficiency (FPS/W), worst-case end-to-end latency (mAP 0.5:0.95), and mean Average Precision. Precisions of FP32, FP16, and INT8 are used for testing the Xavier AGX platform, whilst INT8 and FP32 are used for testing the XCZU9EG and i7-7700, respectively. This research looks at the effects of two different confidence levels on mAP and latency: tc = 0.05 and tc = 0.3. When it comes to most networks, Xavier AGX offers the best mAP. However, when it comes to Mv2(SSD) at INT8, the XCZU9EG performs better. The Xavier AGX has the most fluctuation in post-processing latency, in contrast to the i7-7700's consistent latency performance. An examination of efficiency shows that, when comparing mAP and power efficiency, the Xavier AGX provides the greatest results, whereas the i7-7700 is the least efficient because of its slower performance and greater power consumption.

Keywords: Object Detection, Hardware, Latency, Power, Performance

INTRODUCTION

Driven by the fast developments in convolutional neural networks (CNNs) and deep learning architectures, object identification has changed considerably during the last ten years. When run on strong GPUs in cloud or desktop settings, models such as R-CNN, Fast R-CNN, Faster R-CNN, YOLO (You Only Look Once), and SSD (Single Shot MultiBox Detector) have shown extraordinary accuracy and speed. The difficulty, then, is in adapting these computationally demanding models for embedded systems such Raspberry Pi, NVIDIA Jetson Nano, Google Coral, ARM Cortex CPUs, and other microcontrollers. Often lacking processor power, memory, and energy resources, these platforms call for creative approaches to model compression, optimization, and effective execution.

Lightweight and streamlined versions of deep learning models especially meant to run on embedded hardware are called embedded neural networks. Operating under severe limits, the goal is to maintain a balance between accuracy and computing efficiency. Achieving this balance calls for a variety of methods including model pruning, quantization, knowledge distillation, and the usage of small neural network designs such as MobileNet, SqueezeNet, and EfficientDet. These models are designed to minimize the amount of processes and parameters needed, hence allowing quicker inference and reduced power use. Furthermore, by providing tools for model translation, optimization, and hardware acceleration, specific software frameworks such as TensorFlow Lite, PyTorch Mobile, ONNX Runtime, and NVIDIA TensorRT help to deploy these models on embedded devices.

Real-world situations are quickly seeing the use of embedded neural networks for object identification. Advanced driver-assistance systems (ADAS) in the automobile sector use embedded object detection models to recognize people, cars, traffic signs, and real-time barriers, thereby enhancing road safety and facilitating autonomous navigation. Mobile and service robots in the robotics sector depend on lightweight object identification algorithms to properly sense and interact with their surroundings. Embedded devices with object detection capabilities may monitor public areas and vital infrastructure in smart surveillance while functioning independently and safely without depending on continuous online access. In agriculture as well, drones and smart farming tools use embedded object detection technologies to find crops, spot pests, and evaluate plant health with low latency.

The improvement of data privacy and security is one of the most convincing benefits of including object recognition features in edge devices. The necessity for sending sensitive photographs or videos to the cloud is eliminated since the data is processed locally on the device, hence lowering the danger of data breaches and delay brought on by network transmission. In healthcare applications, where privacy-preserving diagnosis tools are used on embedded devices to spot abnormalities in medical pictures, or in consumer electronics, where smart home sensors recognize objects or activities without exposing user data, this becomes especially crucial.

Embedded neural networks for object detection also creates possibilities for using artificial intelligence in remote or inaccessible locations where connection is inconsistent or nonexistent. Compact AI-powered sensors and cameras may run autonomously to identify animals or threats in wildlife monitoring or disaster response situations, therefore generating alarms and gathering vital data without requiring continuous human supervision or internet connectivity. Important for battery-operated gadgets and solar-powered equipment utilized in such settings, these edge AI systems are also power-efficient.

REVIEW OF LITERATURE

You, Jiahao. (2022)The task of object detection is both vital and challenging in the field of photos and videos. Research on computer vision tasks, including object classification and monitoring, has recently increased. The literature review of this study provides a concise summary of object identification systems and their many applications. We may describe the current research difficulties and future challenges in object detection by evaluating and assessing previous discoveries, gathering relevant data sets and assessment indicators, and then drawing from this.

Wang, Jun et al., (2021) One of the most important and challenging subfields in computer vision, object detection seeks to identify instances of a specific class of semantic objects. Security monitoring, driverless vehicles, and countless other real-world uses are all part of this field's practical applications. Rapid advancements in deep learning algorithms for task detection have greatly improved object detector performance. For the purpose of understanding the field's significant development state, this research provides a comprehensive literature review of target detection together with an overview of the works closely related to it. We detail the various object detection methods, including one-stage and two-stage detectors, and supply the datasets and assessment criteria utilized for object identification in this work. Additionally, object detecting systems' historical development is analyzed. Based on what we know about the current status of target detection, we end by describing the main avenues for future study.

Etxeberria Garcia, Mikel et al., (2020) Research into the possible uses of deep learning in the transportation sector has increased in recent years. Among their responsibilities is the identification of objects, which is essential for autonomous vehicles and, in particular, rail vehicles. Even though deep learning is becoming more popular for object detection in the railway industry, there has been little testing of proposed algorithms on embedded hardware. Using an NVIDIA Jetson AGX Xavier and the industry-standard YoloV3 detector, this article examines the performance of signal inference in the railway domain. Furthermore, several YoloV3 topologies are compared and contrasted to find the best one for that certain dataset. A data augmentation technique called RICAP-DET is employed to create labelled images from a series of image cuts, which are then utilized to build the training dataset. Results show that YoloV3 can be trained with RICAP-DET and can detect rail traffic signals on an embedded platform in real time.

Kung, Jaeha et al., (2018) Deep learning systems have a memory constraint that drastically reduces their efficiency. As the system grows in size, binarizing the activations and weights might be one way to achieve maximum efficiency with minimum accuracy. This study uses and analyzes the binarized neural network to identify people in infrared images. Despite the fact that binarized networks greatly reduce memory cost and simplify computation, our results show that their algorithmic performance is on par with 32-bit floating-point networks. Additionally, we provide a binary representation-optimized system design that outperforms GPU in terms of performance and energy efficiency by a factor of three to four.

Tobias, Luis et al., (2016) Deep Learning (DL) and Convolutional Neural Networks (CNN) have become the go-to solutions for a lot of pattern recognition issues. Thanks to recent developments in processing power, specialized GPUs can be used to expedite the solution of numerical problems. They allow for the conception of much larger networks while also reducing the amount of computer power required. Supercomputers can now run models with greater power, breadth, and depth than ever before. Meanwhile, mobile devices' processing capabilities have improved to the point where CNN models may be deployed in near real-time. Here we investigate whether it is possible to do domain-specific objection identification on mobile devices using convolutional neural network (CNN) methods.

Experimental Setup

Using the 5,000-image COCO2017 validation dataset, the experimental setup assesses object detection models. Efficiency (FPS/W), worst-case end-to-end latency (ms), and mean Average Precision (mAP 0.5:0.95) are the three performance parameters that are taken into consideration. All three hardware platforms were tested: the NVIDIA Jetson Xavier AGX (utilizing FP32, FP16, and INT8), the Xilinx XCZU9EG (utilizing just INT8), and the Intel i7-7700

(utilizing only FP32). The COCO2017 dataset provided the 1,000 training pictures used for INT8 quantization.
Xavier AGX was optimized by activating jetson clocks and MAX N mode. Two cores on the i7-7700 were set aside for benchmarks and two for real-time workloads. Some of the object identification models that were tested were Yolov3, Yolov3-tiny, Centernet-DLA34, Centernet-Resnet101, and Mobilenetv2-SSDLite. Preprocessing, inference via neural networks, and post-processing are the three phases that make up the execution time.

Xavier AGX uses CUDA to speed up pre- and post-processing on GPU for certain models, even though it's done in FP32 on all boards. The efficiency ratio is the power consumption to frames per second. For Xavier AGX and XCZU9EG, powerapp was used to sample power utilization at 40Hz, whereas Open Hardware Monitor 0.9.2 was used for the i7-7700 at 1Hz. A batch size of 1 was utilized in all testing.

RESULTS AND DISCUSSION

Impact of threshold on mAP and latency

When determining a technique's mAP, it is common practice to set the confidence threshold (tc) to 0 (or 0.05) if the method in question does not provide support for a value of 0, as is the case with Yolo and SSD. In most cases, though, a tc of 0.3 is utilized when object detection convolutional neural networks (ODCNNs) are employed. Hence, tc = 0.05 and tc = 0.3 are both taken into account in this research. You may see the outcomes in Figures 1–3.

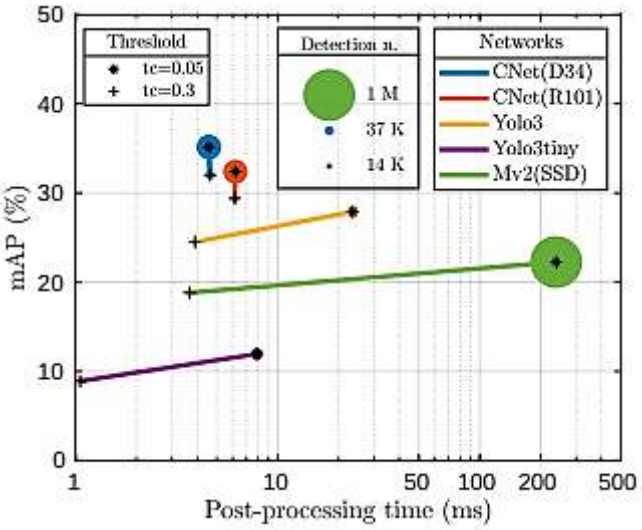

Figure 1: mAP vs. Worst-Case Post-Processing Latency on Xavier AGX (FP32) Using Confidence Thresholds tc = 0.05 (*) and tc = 0.3 (+)

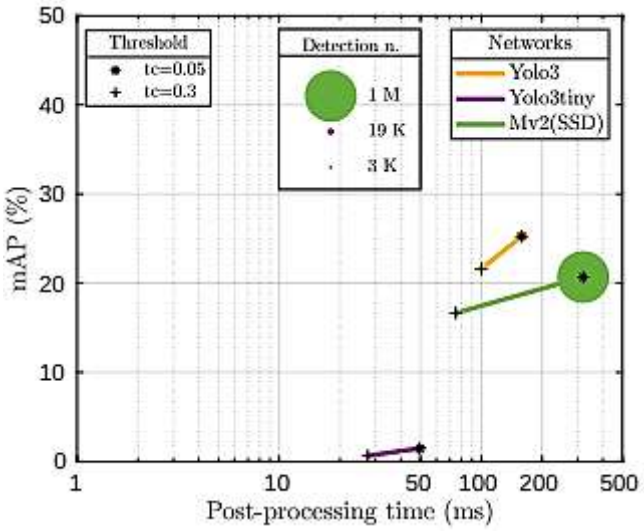

Figure 2: Impact of Post-Processing Latency on mAP for XCZU9EG (INT8) at Confidence Thresholds tc = 0.05 and 0.3

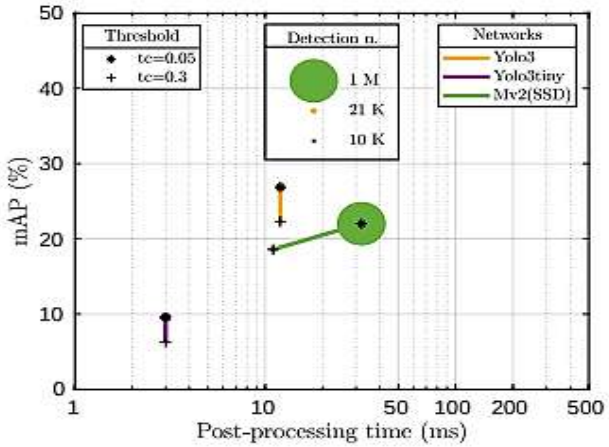

Figure 3: mAP vs. Worst-Case Post-Processing Latency on i7-7700 (FP32) at Confidence Thresholds tc = 0.05 (*) and tc = 0.3 (+)

The following graphs take three things into account: (i) the network's maximum acceptable latency (y-axis), (ii) the worst-case post-processing delay (x-axis), and (iii) the number of detections (radius of the spots). The amount of returned bounding boxes (BBs) is affected only by the post-processing time, which is in turn affected by the confidence threshold (tc). The post-processing delay is increased on all systems when tc is set to 0.05 because mAP and the number of detections are larger. When a lesser threshold is used, the Xavier AGX exhibits the greatest variation in post-processing time, namely a 65× delay for Mv2(SSD).

On the other hand, the i7-7700 has greater stability, with just a 3×delay for Mv2(SSD) at tc = 0.05. Because it has an ARM-based CPU—which isn't as efficient as the Intel processor on the i7-7700—the XCZU9EG has the slowest post-processing time. With a modest variation (below 1-2%) in runtimes and instruction sets between ONNX Runtime on x86-64 and TensorRT on Volta, Xavier AGX produces slightly higher mAP values than the i7-7700 at full accuracy. When compared to other approaches, Yolo methods yield less detections. On the other hand, with tc = 0.05, Mv2(SSD) shows a huge rise in detections. GPU improvements result in more steady post-processing speeds for CNet(D34) and CNet(R101).

Platforms Comparison

Figure 4 shows the duration of the pre-, in-, and post-processing steps. While the XCZU9EG's DPU handles inference, the slower CPU is responsible for pre- and post-processing, which are the most time-consuming steps. The normalization routines are a part of the reason why post-processing takes so long. These operations are executed on the CPU instead of the accelerator, as shown in the Xavier AGX, because some APIs are not available on the DPU or have sluggish implementations, such as sigmoid for Yolo3 and Yolo3tiny or softmax for Mv2(SSD). The Xavier AGX and the i7-7700 both have inference as their primary cause of delay. Keep in mind that the data only reflect the worst-case delays for the sake of brevity, even if the average-case latencies yielded identical findings.

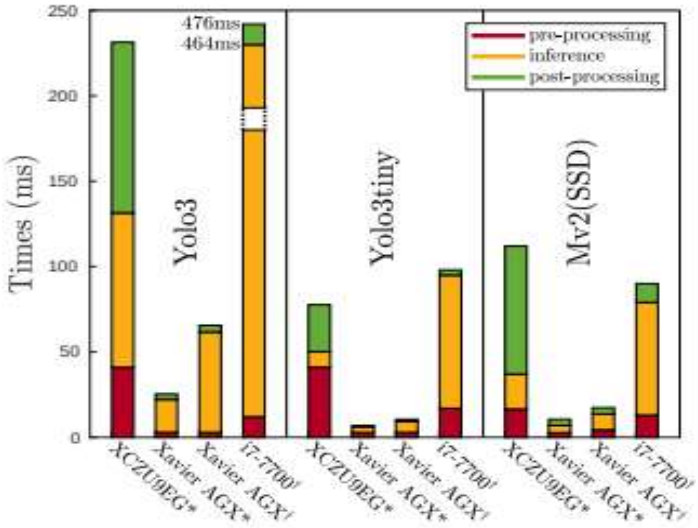

Figure 4: Breakdown of Worst-Case Execution Time

The worst-case end-to-end latency (ms) and maximum achievable packet loss (mAP) (%) are shown side by side in Figure 5. The most accurate and quickest networks are shown in the upper left corner. While FP32 offers slower performance on the Xavier AGX, FP16 gets the same maximum achievable power (mAP). As a matter of fact, the grey-dashed lines representing the Pareto-optimal curve are dominated by FP16 points.

The mAP values on various networks are consistent with one another and with the platforms. With the exception of Mv2(SSD) at INT8, where the XCZU9EG attains a little higher mAP, Xavier AGX provides the maximum achievable power for every network. When it comes to latency, the Xavier AGX is unrivaled across the board. On the other hand, when looking at latency and mAP, the i7-7700 and XCZU9EG do not appear to be the most dominant platforms.

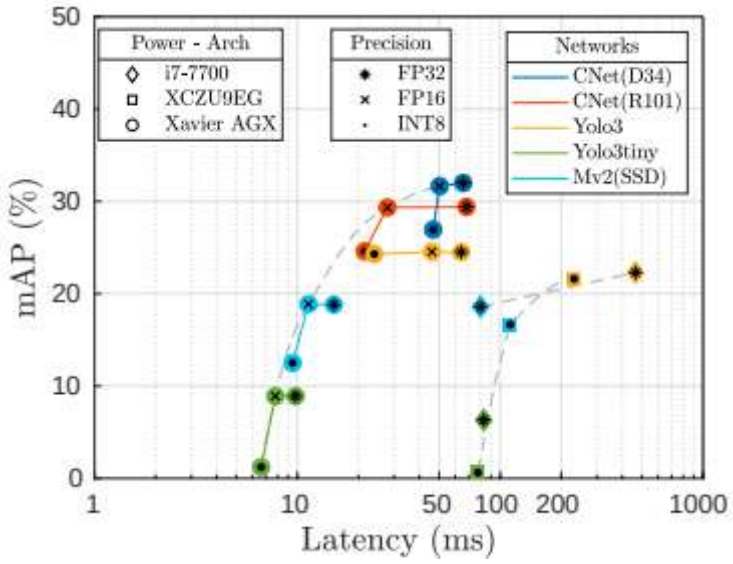

Figure 5: Latency vs. mAP Analysis of Detection Models on Three Hardware Platforms

Figure 6 shows a comparison of efficiency (FPS/Watt) and mAP (%) on two different axes. On top right, you can see the most accurate and efficient networks. Because of its poor performance, the i7-7700 platform has the largest power consumption and is hence the least efficient in this regard. While the XCZU9EG boasts the lowest power consumption, its lackluster frame rate performance renders it inefficient over all. Xavier AGX comes up on top as the most efficient platform, also getting the best mAP. The Pareto-optimal curve is efficient, as FP16 points are dominant once again.

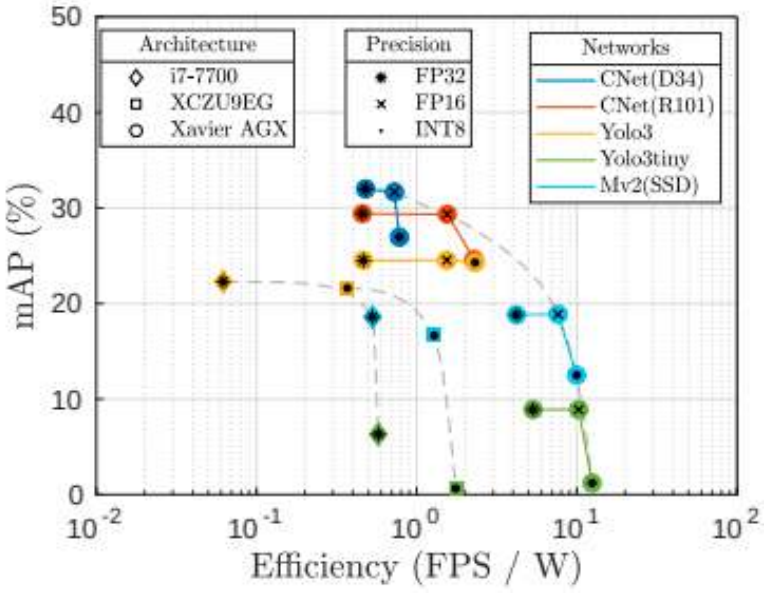

Figure 6: Efficiency vs. mAP Analysis of Detection Models on Multiple Hardware Platforms

In comparison to the i7-7700 and Xavier AGX, the XCZU9EG's power consumption when in idle mode is negligible compared to when inference is active. This is important to note when discussing power usage.

CONCLUSION

Of the three, the Xavier AGX regularly outperforms the others, notably with FP16 accuracy, which dominates the Pareto-optimal frontier, by finding the ideal balance between mAP, inference speed, and energy economy. Although the XCZU9EG is the most power-efficient in terms of raw consumption, its restricted processing capability and slower CPU influence total throughput, especially in pre- and post-processing phases. Conversely, the i7-7700, albeit providing consistent latency, lacks efficiency because of high power consumption and lengthier inference times. The study further shows that confidence thresholds have a major influence on post-processing delay and detection count, hence influencing end-to-end performance. All things considered, the most efficient and powerful way to run deep learning-based object recognition in real-time, power-sensitive situations is the Xavier AGX.

REFERENCES

[1]. D. R. Prado, "The Generalized Intersection Approach for Electromagnetic Array Antenna Beam-Shaping Synthesis: A Review," IEEE Access, vol. 10, pp. 87053–87068, 2022.
[2]. J. You, "Deep Neural Networks for Object Detection," Highlights in Science, Engineering and Technology, vol. 17, no. 2, pp. 159–165, 2022.
[3]. V. K. Reddy, S. Shaik, and S. Rao, "Machine learning based outlier detection for medical data," Indonesian Journal of Electrical Engineering and Computer Science, vol. 24, no. 1, pp. 1–4, 2021.
[4]. J. Wang, T. Zhang, Y. Cheng, and N. Al-Nabhan, "Deep Learning for Object Detection: A Survey," Computer Systems Science and Engineering, vol. 38, no. 2, pp. 165–182, 2021.
[5]. M. Etxeberria Garcia, F. Ezaguirre, J. Plazaola, U. Munoz, and M. Zamalloa, "Embedded object detection applying Deep Neural Networks in railway domain," vol. 1, no. 2, pp. 565–569, 2020.
[6]. X. Yang, S. Chaudhuri, L. Naviner, and L. Likforman, "Quad-Approx CNNs for Embedded Object Detection Systems," pp. 1–4, 2020.
[7]. J. Kung, D. Zhang, G. van der Wal, S. Chai, and S. Mukhopadhyay, "Efficient Object Detection Using Embedded Binarized Neural Networks," Journal of Signal Processing Systems, vol. 90, no. 2, pp. 1–14, 2018.
[8]. B. Kang, S. Tripathi, G. Dane, and T. Nguyen, "Low-complexity object detection with deep convolutional neural network for embedded systems," vol. 60, 2017.
[9]. L. Tobias, A. Ducournau, F. Rousseau, G. Mercier, and R. Fablet, "Convolutional Neural Networks for Object Recognition on Mobile Devices: a Case Study," vol. 1, no. 2, pp. 1–8, 2016.
[10]. H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang, "Towards Real-Time Object Detection on Embedded Systems," IEEE Transactions on Emerging Topics in Computing, vol. PP(99), pp. 1–2, 2016.
[11]. J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in Advances in Neural Information Processing Systems, 2016, pp. 379–387.
[12]. S. Ren, K. He, R. Girshick, et al., "Faster R-CNN: Towards real-time object detection with region proposal networks," in Conference on Neural Information Processing Systems, 2015, pp. 91–99.
[13]. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, no. 2, pp. 1929–1958, 2014.