JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 37 (2025) 379–390
Received: 26 May 2025
Accepted: 19 August 2025

Analysis of the use of object detection systems in edge computing

Analiza wykorzystania systemów detekcji obiektów w przetwarzaniu typu edge computing

Jakub Kozłowski*, Marcin Badurowicz

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This article analyzes the potential of using artificial intelligence for object detection in edge computing environments, which are gaining importance with the growing number of Internet of Things devices. The focus is on evaluating algorithms in terms of accuracy, speed, and energy efficiency. The goal is to identify solutions that minimize latency, which is crucial for autonomous systems and surveillance. Experiments were conducted on three devices using YOLO, SSD, and Faster R-CNN models. The results highlight the most effective object detection methods in edge computing, supporting the development of industry and IoT.

Keywords: edge computing; object detection; Internet of Things

Streszczenie
W artykule przeanalizowano potencjał wykorzystania sztucznej inteligencji w detekcji obiektów w środowiskach edge computing, które zyskują na znaczeniu wraz ze wzrostem liczby urządzeń Internetu Rzeczy. Skupiono się na ocenie algorytmów pod kątem dokładności, szybkości oraz efektywności energetycznej. Celem było zidentyfikowanie rozwiązań minimalizujących latencję, istotną w systemach autonomicznych i monitoringu. Testy przeprowadzono na trzech urządzeniach z wykorzystaniem modeli YOLO, SSD i Faster R-CNN. Wyniki wskazują najskuteczniejsze metody detekcji obiektów w edge computing, wspierające rozwój przemysłu i IoT.

Słowa kluczowe: przetwarzanie brzegowe; detekcja obiektów; Internet Rzeczy

*Corresponding author
Email address: **jakub.kozlowski.701@gmail.com** (J. Kozłowski)

1. Introduction

With the rapid development of the Internet of Things and the growing demand for real-time data processing applications, object detection systems in edge computing environments are becoming increasingly important. In many cases, sending data to the cloud is inefficient or even impossible due to the need for low latency, limited network bandwidth, or concerns related to data privacy. A good example is urban surveillance systems, which must quickly detect dangerous situations, or autonomous vehicles, which cannot wait for a response from a remote server to make an emergency braking decision.

In the era of digitalization and the fourth industrial revolution, the role of edge computing is becoming crucial, especially in areas such as autonomous transportation, smart cities, Industry 4.0, and medicine. Local data processing, without the need to send it to the cloud, not only increases security but most importantly enables near-instant decision-making.

However, designing edge computing systems involves many constraints. Low computing power, limited memory, and the need to minimize energy consumption require careful selection of both algorithms and hardware. The key challenge is to find the right balance between accuracy, performance, and energy efficiency. It is also worth emphasizing that edge environments are highly diverse – from smartphones, through embedded platforms such as Raspberry Pi, to advanced systems like Nvidia Jetson Nano – which further affects the approach to implementing algorithms.

This article focuses in particular on algorithms that enable fast object detection, such as YOLO, SSD, or Faster R-CNN. Their efficiency in edge environments is crucial, for example, in real-time monitoring systems, medical diagnostics, industrial quality control, and many IoT solutions.

The aim of this article is to analyze the performance of various edge devices in image processing and to evaluate the suitability of selected object detection models for real-world applications. The results of this research may contribute to the development of more efficient and energy-saving solutions that can be applied both in industry and educational environments.

2. Literature Review

The rapid development of hardware technologies such as NVIDIA Jetson processors, Raspberry Pi with Neural Compute Stick, and various FPGA platforms has significantly contributed to the evolution of object detection algorithms. Algorithms like YOLO and Faster R-CNN, which were originally designed for powerful computing platforms, are now increasingly implemented on smaller, resource-constrained devices. This is particularly important in the context of edge computing, where local data processing is crucial for applications requiring low latency and fast response times, such as video

surveillance, autonomous vehicles, and real-time object detection systems.

The literature review focuses on five key studies that comprehensively analyze the performance of object detection algorithms on edge devices. The studies highlight important aspects such as inference time, detection accuracy, and energy consumption, which are essential for selecting the appropriate algorithm and hardware platform for a given application.

In the article Benchmark Analysis of YOLO Performance on Edge Intelligence Devices [1], the authors examine the efficiency of running the YOLO algorithm on various edge platforms, including NVIDIA Jetson Nano, Jetson Xavier NX, and Raspberry Pi 4B with Intel Neural Compute Stick 2. The study shows that the Jetson Xavier NX provides the highest computing performance, making it suitable for more demanding tasks. On the other hand, the Raspberry Pi 4B combined with the NCS2 demonstrates lower power consumption, which is advantageous for mobile and energy-efficient applications. The paper emphasizes that selecting the appropriate platform should always depend on whether the priority is maximum performance or minimal energy consumption

The article Performance Analysis of Deep Learning-Based Object Detection Algorithms on COCO Benchmark [2] presents a comparison of popular object detection algorithms, such as Faster R-CNN, Mask R-CNN, and DyHead, based on the COCO dataset. The research indicates that the DyHead algorithm achieved the best detection results, particularly in complex detection scenarios. However, algorithms like NAS-FPN and DetectorRS, although highly effective, require significantly more computational resources. The authors also emphasize that the final choice of algorithm should always consider the application's specific requirements, especially in the context of edge devices with limited computational capabilities.

The article Edge Computing by M. Satyanarayanan [3] provides a broader context for the importance of edge computing, which is essential for the operation of real-time systems, including those based on object detection. The author highlights that edge computing enables local data processing, significantly reducing latency and improving system responsiveness, which is particularly important for applications such as autonomous systems and video monitoring. However, the paper points out that the main challenge remains optimizing algorithms to work effectively on devices with limited computational resources.

The article Deep Learning for Edge Computing Applications: A State-of-the-Art Survey [4] presents an extensive overview of the current trends and challenges in applying deep learning in edge computing. The authors discuss advanced optimization techniques, such as model compression and distributed computing, which are key to adapting algorithms to the limited computing and energy capacities of edge devices. This article is particularly valuable in the context of object detection, as it offers practical solutions for improving the efficiency and performance of algorithms in edge environments.

The article Object Detection using YOLO: Challenges, Architectural Successors, Datasets and Applications [5] thoroughly discusses the evolution of the YOLO algorithm, starting from YOLOv1 to its most recent versions, which have significantly improved both detection accuracy and processing speed. The authors analyze the challenges of detecting small objects and objects on complex backgrounds and emphasize the importance of using datasets such as COCO for training models adapted to real-world conditions. Particular attention is paid to the operation of YOLO on edge devices, where optimizing the model is necessary to maintain a balance between detection quality and real-time processing speed.

The review of the literature clearly shows that object detection algorithms in edge computing environments must be carefully adapted to the hardware limitations of devices such as cameras, sensors, or embedded systems. The choice of model depends not only on accuracy but also on processing speed and energy consumption. YOLO remains one of the most commonly used algorithms due to its speed and versatility, but its deployment on edge devices still requires continuous improvements and optimization.

Future research directions in this field should focus on simplifying network architectures, using automatic optimization techniques, and precisely adapting algorithms to specific devices. This will enable the development of edge computing systems that are fast, energy-efficient, and capable of operating in real-time conditions.

3. Purpose of the Study

The aim of this research is to evaluate the performance of object detection algorithms in the context of edge computing, with particular attention to their deployment on various hardware platforms. The study seeks to identify challenges related to resource optimization, including CPU, RAM, GPU, and power consumption, and to propose recommendations for improving the efficiency of these algorithms. The research will be divided into six stages:

- selecting appropriate algorithms for testing,
- selecting three edge computing platforms representing different device types (smartphones, single-board computers, and desktop computers) and evaluating their technical specifications,
- preparing the testing environment,
- conducting performance tests using the selected algorithms on each platform,
- recording test results, including task execution time, resource utilization, and power consumption,
- analyzing the results, with particular focus on the effectiveness of the algorithms across different hardware platforms.

4. Research methodology

4.1. Research Tools

Object detection is one of the key challenges in the field of image processing, and it has gained particular importance thanks to the rapid development of neural networks and deep learning techniques. The ability to detect objects quickly and accurately is critical in many modern applications, such as autonomous driving, traffic monitoring, industrial automation, and security systems. In edge computing environments, where real-time processing and limited hardware resources are major constraints, choosing the right detection algorithms becomes even more crucial.

In recent years, various object detection architectures have been developed, significantly improving both detection speed and accuracy. Among these, three models have become particularly popular and have been widely implemented in edge and cloud-based solutions: YOLO, SSD, and Faster R-CNN.

- **YOLO (You Only Look Once)** is one of the most widely recognized architectures for object detection. The model is known for its exceptional speed, thanks to its unique approach that treats object detection as a regression problem over a single grid. Each grid cell predicts both the class and the bounding box coordinates in one step. YOLO provides a favorable trade-off between speed and accuracy, which makes it ideal for applications requiring low latency, such as pedestrian detection in autonomous vehicles or monitoring traffic signs in real time [6].
- **SSD (Single Shot MultiBox Detector)** is another model that performs object detection in a single step. Unlike YOLO, SSD uses multiple feature maps to detect objects at different scales, which improves its ability to handle objects of various sizes. Its main advantage is fast detection with relatively good accuracy, which is why SSD is often chosen for edge computing applications, such as mobile-based augmented reality systems and smart city sensors, where quick responses are essential [7].
- **Faster R-CNN** is considered one of the most accurate object detection architectures. The model operates in two stages: first, it extracts image features using a convolutional network, and then it uses a Region Proposal Network (RPN) to generate object proposals. Each proposal is then classified and precisely localized. Faster R-CNN achieves very high accuracy but requires significant computational resources, which makes it less suitable for low-power edge devices. However, it remains valuable in scenarios where high precision is critical, such as medical imaging or security systems with strict accuracy requirements [8].

These three models were selected for this study because they represent a diverse spectrum of object detection strategies from highly optimized for speed to those prioritizing accuracy. The choice allows for a comprehensive comparison of their performance on edge devices under varying computational constraints.

All selected models were trained on the COCO dataset, which is one of the most widely used datasets in the field of object detection. COCO contains thousands of images representing everyday scenes and objects, with detailed annotations for detection, segmentation, and keypoint recognition. The diversity and realism of this dataset make it particularly suitable for developing and testing algorithms intended for real-world applications [9].

For this study, the STOP sign was selected as the detection object. This choice is not accidental, it is based on the distinctive shape, color contrast, and critical importance of traffic signs in transportation and safety systems. STOP signs are present in various environments, from urban intersections to rural roads, and detecting them accurately is crucial for applications such as driver assistance systems, autonomous vehicles, and traffic monitoring. Additionally, STOP signs often appear near other objects (trees, buildings, vehicles), making their detection a realistic challenge for object detection algorithms.

4.2. Research Platforms

The study was conducted on three different edge computing platforms to reflect the diversity of devices commonly used in real-world scenarios. The selected platforms vary in terms of computational power, memory, and intended usage, which allows for a meaningful evaluation of how object detection algorithms perform across a wide range of hardware.

a) Motorola Moto G50

The Motorola Moto G50 is a smartphone that represents a category of mobile edge devices. Smartphones are widely used in edge computing, particularly in real-time applications such as pedestrian tracking, environmental monitoring, mobile health diagnostics, and smart city management. Testing on this device simulates typical mobile edge scenarios, where computational resources are limited, but low latency and portability are critical. Table 1 shows the technical specifications of the Motorola Moto G50.

Table 1: Motorola Moto G50 – technical specifications

Technical specifications Motorola Moto G50	
CPU	Qualcomm Snapdragon 480
RAM	4 GB
Operating system	Android 13
GPU	Adreno 619

b) Router NanoPI R5c

The NanoPi R5c is a compact single-board computer commonly used in networking and IoT edge solutions. Its low power consumption and ability to locally process data make it suitable for applications such as smart home systems, industrial automation, or edge gateways for distributed sensor networks. Evaluating performance on this device reflects use cases where local processing is needed to reduce latency and bandwidth usage in distributed

systems. Table 2 presents the technical specifications of the NanoPI R5c.

Table 2: NanoPI R5c – technical specifications

Technical specifications NanoPI R5c	
CPU	Rockchip RK3568B2
RAM	4 GB
Operation system	FriendlyWrt
GPU	GPU Mali-G52 MP2

c) Lenovo Thinkpad T480

The Lenovo ThinkPad T480 is a mobile workstation that represents more powerful edge computing platforms capable of handling more complex tasks. This type of device is often used in edge applications that require on-site data analysis, including real-time video processing in surveillance systems or advanced quality control in manufacturing. Testing on this device helps evaluate whether heavier algorithms can realistically run outside of centralized data centers. Table 3 presents the technical specifications of the Lenovo Thinkpad T480.

Table 3: Lenovo Thinkpad T480 – technical specifications

Technical specifications Lenovo Thinkpad T480	
CPU	Intel Core i5-8350U
RAM	32 GB
Operating system	Linux Mint 22
GPU	Intel HD Graphics 620

4.3. Experimental Setup

The work involved the implementation and evaluation of selected object detection algorithms on three representative edge computing devices. Given the increasing demand for intelligent processing directly on edge devices, this study aims to assess how these models perform in terms of accuracy, inference speed, and resource consumption under realistic conditions. The PyTorch framework was chosen for its flexibility and efficient support for deploying deep learning models across diverse hardware platforms. The implementation process was divided into several stages to ensure systematic preparation, execution, and analysis of the experiments:

- **Environment Preparation** - Each tested device was carefully prepared with all necessary software components for running deep learning algorithms. This included installing the Python interpreter, the PyTorch framework, and additional packages such as NumPy, OpenCV, torchvision, and psutil. These tools facilitated image loading and processing, as well as system resource monitoring during inference.
- **Model Configuration** - Three widely-used object detection models: YOLO, SSD, and Faster R-CNN were employed. Versions available in public libraries that are designed to work efficiently on devices with limited computational resources were selected. This approach allowed the models to run without modifications to their architectures, ensuring a fair and practical evaluation of their out-of-the-box performance on edge hardware.
- **Test Routine Implementation** - To compare model performance across devices, a set of test scripts was developed enabling:
  - **a)** Loading of models and test images.
  - **b)** Measurement of single-image processing times to assess latency.
  - **c)** Recording of results in a structured CSV format for subsequent analysis.

  The test dataset consisted of representative im-ages with varying complexity levels, ensuring uniform and realistic testing conditions.
- **Hardware Resource Monitoring** - Throughout the tests, critical hardware resources such as CPU, RAM, and GPU usage were monitored using the psutil library, which tracks processor load, memory occupancy, and active process durations. On devices equipped with suitable drivers and sensors, power consumption data was also recorded. This was particularly important for mobile and single-board computers where energy efficiency is a key parameter.
- **Data Analysis** - The collected data was analyzed to evaluate each model's detection effectiveness using metrics including processing time, accuracy, and resource efficiency. This analysis allowed detailed comparison of the models' strengths and weaknesses regarding their applicability to edge computing scenarios.
- **Conclusions and Recommendations** - Based on the experimental results, the relative efficiency of each algorithm on edge devices was assessed. The models were compared in terms of detection accuracy, processing speed, and resource consumption, which enabled identification of optimal solutions for systems with constrained computing power. Key trade-offs between detection precision, inference latency, and energy usage were highlighted, providing guidance for practical implementation.

For example, during testing, an image containing a STOP sign under various lighting conditions was processed individually by each model on all devices. The inference time, resource usage, and detection accuracy were recorded and analyzed. This practical scenario illustrates how the setup captures real-time performance and accuracy trade-offs relevant for edge applications like traffic sign detection in autonomous vehicles or smart city monitoring.

4.4. Comparative Metrics

The research was conducted based on accordance with a predefined research methodology, involving the measurement of three key parameters: processing time, accuracy and resource efficiency. The aim of this research was to evaluate the performance of selected object

detection models under different edge computing device conditions. The tests were carried out on three different platforms, which provided a complete picture of the performance of the algorithms in diverse environments. The following aspects were taken into account in the study:

a) Processing Time

Measuring processing time was one one of the key components of the study, as algorithm response time is critical in edge computing systems, where delays can affect application performance.

Preprocessing time - The time taken to prepare the model for inference was measured, which included loading the model, preprocessing the image (e.g., changing resolution, normalizing) and setting up the environment.

- **Inference time** - The main measure, denoting the time it takes the algorithm to process the image and make a prediction.
- **Final time** - Included all operations related to post-processing of results, such as detection filtering, drawing frames around objects or calculating results (e.g., precision, recall).
- **Latency** - The total delay from the time the image enters the system to the time the results are obtained.

Measuring the times is crucial in assessing which models are best suited for applications requiring low latency, such as object detection in real time on edge devices.

b) Accuracy

The accuracy of object recognition was assessed using detection quality indicators, particularly the Confidence index, which indicates how confident the model is about its prediction.

- **Confidence** - The average value of the confidence index for all object detections in the image was measured.
- **Number of correct detections** - The number of correctly identified objects was determined in relation to all detected objects, which allowed the effectiveness of the models to be assessed in various test scenarios.
- **Number of incorrect recognitions** - The number of incorrectly identified objects was measured containing elements resembling the target object.

c) Resource Efficiency

Monitoring resource efficiency has made it possible to assess how different models affect computing power, memory and energy consumption, which is crucial in the context of edge devices, which typically have limited resources.

- **Power consumption** - Using power consumption monitoring tools, it was measured how much power the device was consuming during image processing.
- **CPU and RAM consumption** - By measuring CPU consumption, it was possible to assess how intensely each model was taxing the device's CPU.
- **CPU temperature** – It was monitored to determine how intensely the models were loading the hardware.

5. Results

The colors used in the tables are significant: green indicates the NanoPI R5c device, purple represents the Motorola Moto G50, and maroon refers to the Lenovo Thinkpad T480. Figure 1 presents the relationship between detection rate and false positives for the analyzed models, with colors indicating the average confidence score. These visualizations provide insight into the performance of each model variant, revealing the balance between correct detections and false alarms. The size and positioning of each point allow for evaluation of both effectiveness and computational efficiency across different hardware platforms.

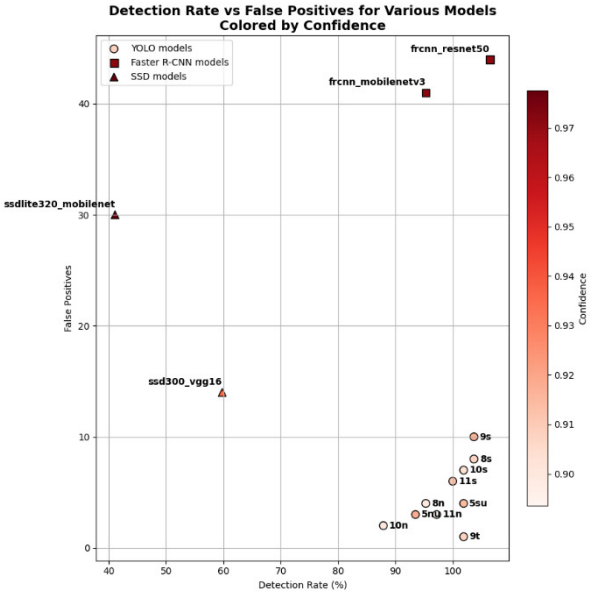

Figure 1: Detection results on the NanoPi R5c device.

Figure 2 presents the same type of analysis as Figure 1, focusing this time on the Motorola Moto G50 device. While the structure of the plot remains consistent, illustrating the relationship between detection rate and false positives, with color encoding the average confidence, the data were collected entirely on a mobile platform. This comparison highlights how the models behave under more constrained computational resources, characteristic of smartphones. Observing shifts in confidence levels, detection accuracy, and the distribution of false positives offers valuable insight into the limitations and capabilities of edge devices like the Moto G50 when running object detection algorithms.

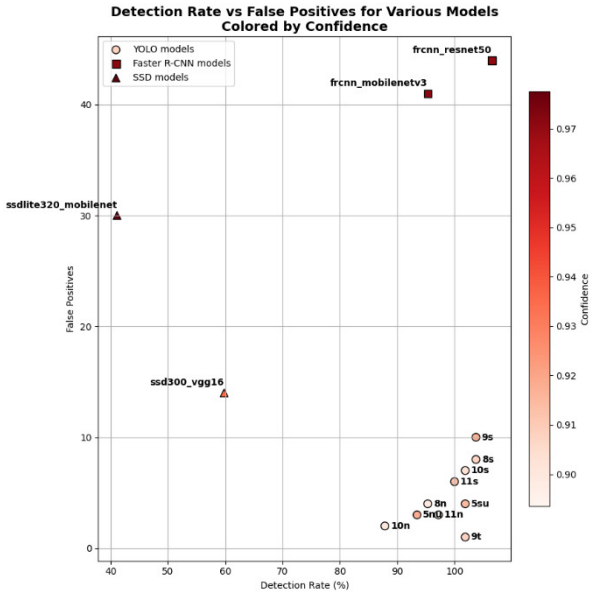

Figure 2: Detection results on the Motorola Moto G50 device.

Figure 3 shows the same set of results as the previous figures, this time obtained on the Lenovo ThinkPad T480. As a laptop-class device with significantly more computing power, the T480 allows for smoother execution of detection models. The visualization highlights how increased hardware capabilities affect detection accuracy, confidence scores, and the rate of false positives, offering a useful point of reference when comparing mobile and embedded platforms to standard computing environments.

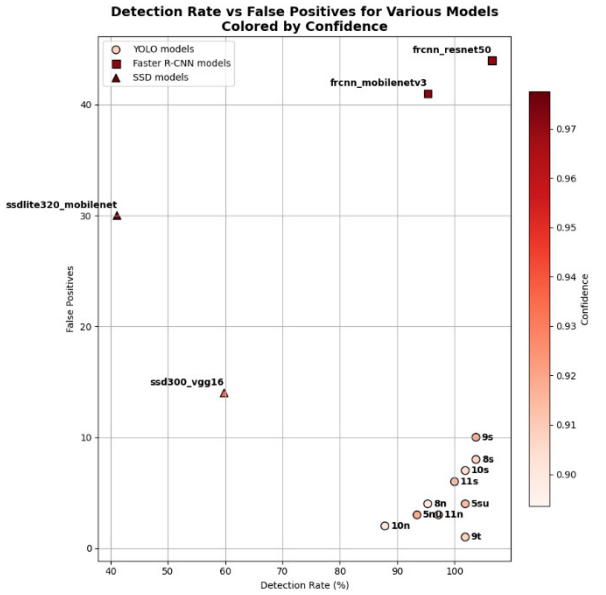

Figure 3: Detection results on the Lenovo Thinkpad T480 device.

Figure 4 presents latency measurements for selected object detection models tested on three different hardware platforms. This analysis provides a comprehensive understanding of the trade-offs between model complexity, inference speed, and hardware limitations. Consequently, it offers valuable insights into which models are most suitable for deployment in specific contexts, whether on resource-constrained edge devices, moderately powered mobile platforms, or more advanced laptop systems, thereby supporting informed decision-making for practical applications of object detection technology.

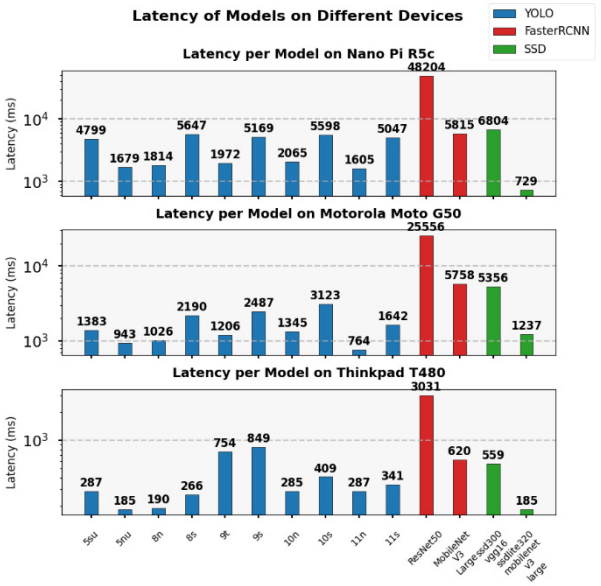

Figure 4: Latency of Models on Different Devices.

Figures 5–10 compare the models in terms of resource efficiency, including energy consumption, memory usage, and hardware optimization. This analysis highlights how effectively the models use resources during inference, which is crucial for devices with limited capabilities. It helps identify the most suitable models for different environments, from edge devices to mobile platforms and laptops.

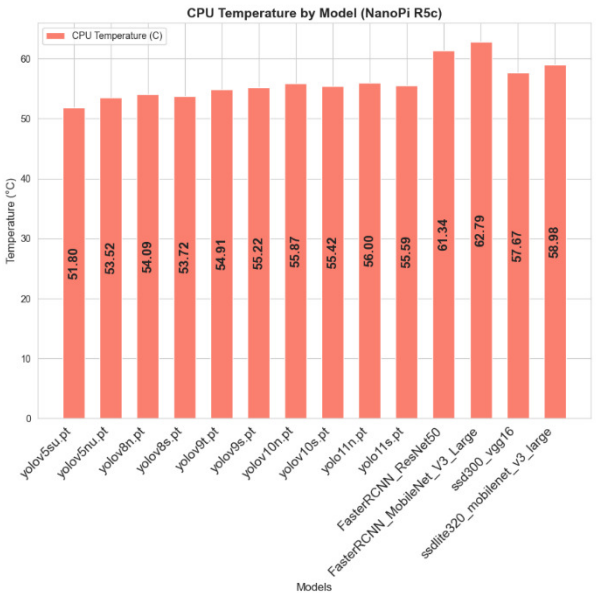

Figure 5: CPU Temperature by Model on the NanoPi R5c device.

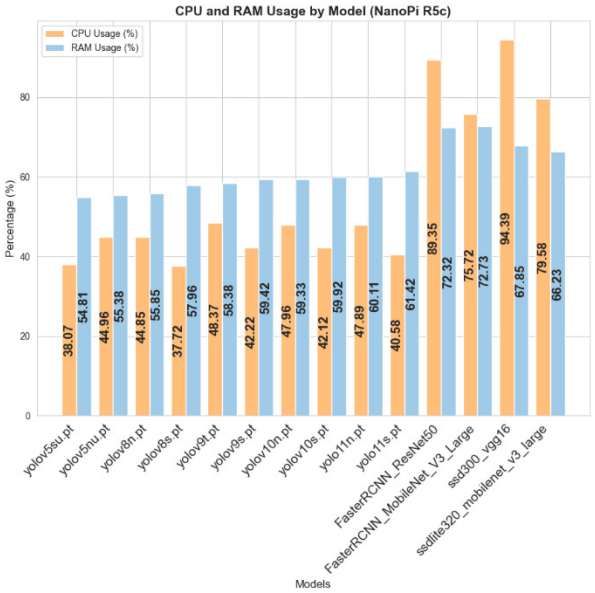

Figure 6: CPU and RAM Usage by Model on the NanoPi R5c device.

Figures 5 and 6 present the CPU temperature as well as CPU and RAM usage for various object detection models running on the NanoPi R5c device. Analyzing these parameters allows assessment of how each model impacts the computational load and thermal behavior of this edge device, which is characterized by limited processing power and cooling capabilities. These results are important for understanding which models can operate efficiently without risking overheating or excessive energy consumption in a resource-constrained environment. Monitoring these metrics also helps identify potential bottlenecks that could affect real-time performance.

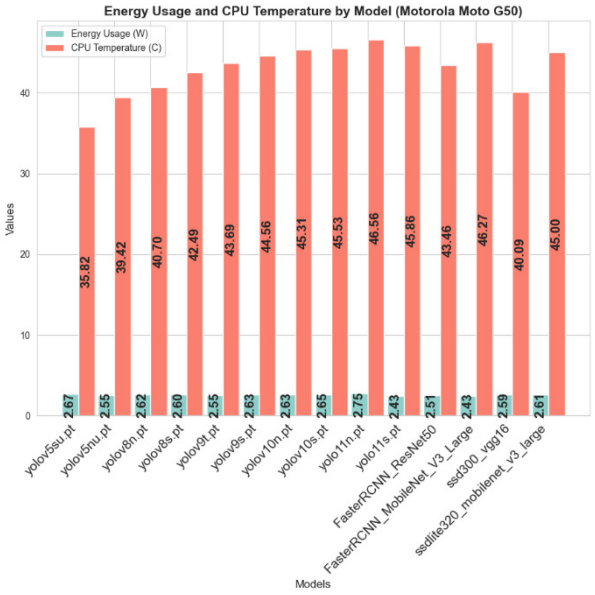

Figure 7: Energy Usage and CPU Temperature by Model on the Moto G50 device.

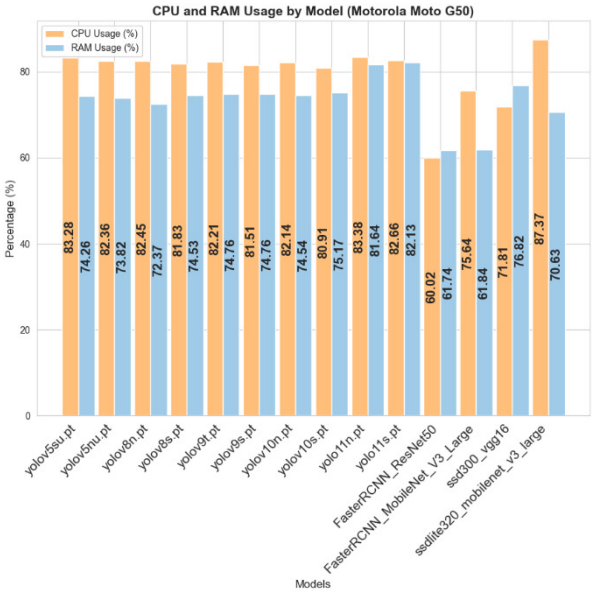

Figure 8: CPU and RAM Usage by Model on the Moto G50 device.

Figures 7 and 8 show energy consumption, CPU temperature, and memory usage for models tested on the Motorola Moto G50 smartphone. These figures provide a detailed view of how each model balances performance with power efficiency on a mobile platform, where battery life is a critical factor. The analysis highlights variations in energy demands and thermal output among models, which directly influence user experience through device responsiveness and battery longevity. Additionally, the memory usage data offer insight into the models' footprint on limited mobile resources, helping to identify those that maintain optimal operation without causing slowdowns or excessive power drain. This comprehensive assessment is essential for selecting models suitable for real-world mobile applications.

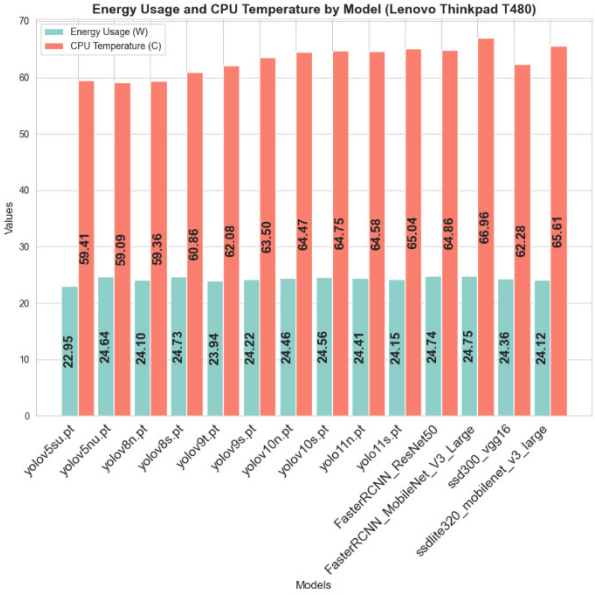

Figure 9: Energy Usage and CPU Temperature by Model on the Thinkpad T480 device.

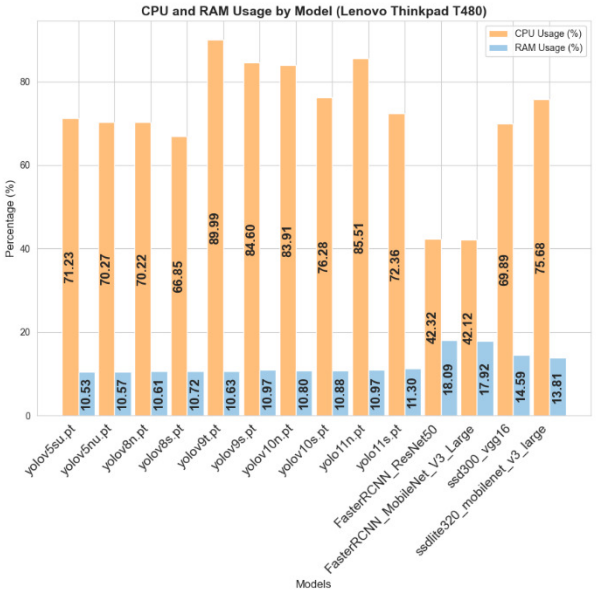

Figure 10: CPU and RAM Usage by Model on the Thinkpad T480 device.

Figures 9 and 10 illustrate energy consumption, CPU temperature, and RAM usage for models running on the ThinkPad T480 laptop. The data reflect the resource demands on a more powerful system, providing insight into model efficiency in less constrained environments.

The performance metrics in Tables 4–6 relate to different ratios, such as CPU/Energy, RAM/Energy or Confidence/Latency. This makes it possible to analyse in detail the impact of individual parameters on the performance of the models, allowing them to be optimised in the context of different resources.

Table 4: Performance metrics on the NanoPi R5c device

Model	Confidence / Latency	Latency / Model Size
ssd300_vgg16	0.000138	50.0
ssdlite320_mobilenet_v3_large	0.001342	55.5
FasterRCNN_MobileNet_V3_Large	0.000167	78.6
yolov8s	0.000161	262.2
yolov5su	0.000191	270.8
yolo11s	0.000181	274.0
yolov8n	0.000495	290.3
yolo11n	0.000557	299.9
FasterRCNN_ResNet50	0.000020	302.6
yolov5nu	0.000546	316.2
yolov9s	0.000177	352.1
yolov10s	0.000161	353.2
yolov10n	0.000436	369.4
yolov9t	0.000460	416.1

Table 5a: Performance metrics on the Motorola Moto G50 device

Model	(1)	(2)	(3)	(4)
yolo11n	3.1	2.1	30.3	29.7
yolov5nu	2.8	2.4	32.3	28.9
yolov8n	2.9	2.7	31.5	27.6
yolov9t	2.8	3.1	32.2	29.3
mobilenet_v3_large	2.7	3.2	33.5	27.1
yolov10n	2.9	3.5	31.2	28.3
yolov5su	2.9	3.7	31.2	27.8
yolo11s	2.7	4.0	34.0	33.8
yolov8s	2.9	5.7	31.5	28.7
yolov9s	2.9	6.5	31.0	28.4
yolov10s	2.9	8.3	30.5	28.4
ssd300_vgg16	2.8	13.9	27.7	29.7
MobileNet_V3_Large	2.5	14.0	31.1	25.4
ResNet50	2.6	64.1	23.9	24.6

Legend:

1. Energy / Confidence
2. Energy Consumption (J)
3. CPU / Energy
4. RAM / Energy

Table 5b: Performance metrics on the Motorola Moto G50 device

Model	(5)	(6)	(7)
yolo11n	0.51	0.001170	142.8
yolov5nu	0.48	0.000973	177.6
yolov8n	0.42	0.000876	164.1
yolov9t	0.54	0.000753	254.3
mobilenet_v3_large	0.20	0.000790	94.3
yolov10n	0.47	0.000670	240.7
yolov5su	0.15	0.000661	78.1
yolo11s	0.13	0.000555	89.1
yolov8s	0.12	0.000414	101.7
yolov9s	0.18	0.000369	169.4
yolov10s	0.17	0.000289	197.0
ssd300_vgg16	0.02	0.000175	39.4
MobileNet_V3_Large	0.03	0.000169	77.9
ResNet50	0.02	0.000038	160.4

Legend:

5. Model Size / Energy
6. Confidence / Latency
7. Latency / Model Size

Table 6a: Performance metrics on the Lenovo Thinkpad T480 device

Model	(1)	(2)	(3)	(4)
mobilenet_v3_large	24.7	4.5	3.1	0.6
yolov5nu	26.9	4.6	2.9	0.4
yolov8n	26.8	4.6	2.9	0.4
yolov8s	27.3	6.6	2.7	0.4
yolov5su	25.1	6.6	3.1	0.5
yolov10n	27.2	7.0	3.4	0.4
yolo11n	27.3	7.0	3.5	0.4
yolo11s	26.5	8.2	3.0	0.5

yolov10s	27.2	10.0	3.1	0.4
ssd300_vgg16	26.0	13.6	2.9	0.6
MobileNet_V3_Large	25.5	15.3	1.7	0.7
yolov9t	26.4	18.0	3.8	0.4
yolov9s	26.4	20.6	3.5	0.5
ResNet50	25.5	75.0	1.7	0.7

The legend for Tables 6a and 6b is identical to that of Tables 5a and 5b and therefore has not been repeated.

Table 6b: Performance metrics on the Lenovo Thinkpad T480 device

Model	(5)	(6)	(7)
mobilenet_v3_large	1.84	0.005273	14.1
yolov5nu	4.64	0.004947	34.9
yolov8n	3.86	0.004734	30.4
yolov8s	1.15	0.003410	12.4
yolov5su	1.30	0.003189	16.2
yolov10n	4.38	0.003156	51.1
yolo11n	4.56	0.003113	53.7
yolo11s	1.31	0.002678	18.5
yolov10s	1.55	0.002210	25.8
ssd300_vgg16	0.18	0.001675	4.1
MobileNet_V3_Large	0.33	0.001567	8.4
yolov9t	5.05	0.001204	159.0
yolov9s	1.65	0.001080	57.8
ResNet50	0.16	0.000320	19.0

6. Discussion

6.1. Detection Result

The article analyzes object detection results obtained on three different devices: NanoPI R5c, Motorola Moto G50 and Thinkpad T480. They were compared in terms of four key parameters: Confidence, number of correct detections, number of incorrect detections and model size.

Most of the models show a high level of detection confidence between 0.89 and 0.98, indicating their ability to accurately recognise objects. The yolo11n model recorded the lowest Confidence value (0.8935), while ssdlite320_mobilenet_v3_large achieved the highest value (0.9776). Models with higher Confidence, such as FasterRCNN_ResNet50 (0.9705) and FasterRCNN_MobileNet_V3_Large (0.9711), theoretically provide higher prediction quality.

An interesting observation is that some models, such as FasterRCNN_ResNet50, yolov8s and yolov9s, achieve correct detections above 100% (106.54%, 103.74% and 103.74%, respectively). This indicates the presence of false positives, i.e. cases, in which models incorrectly classify objects outside the analysed set as correct. For this reason, despite their high Confidence, their practical usefulness may be limited in environments requiring higher precision.

In contrast, models such as yolov5su (Confidence: 0.9142, correct detections: 101.87%) or yolov9t (Confidence: 0.9079, correct detections: 101.87%) show more balanced results, with a relatively low number of False positives. These are examples of models that combine well high precision with a low number of misclassifications, making them more effective in applications requiring reliability.

Models with correct detections well below 100%, such as ssd300_vgg16 (59.81%) and ssdlite320_mobilenet_v3_large (41.12%), do not meet high detection standards and have limited effectiveness in practice, although their Confidence remains high.

The models differ significantly in terms of the number of erroneous detections. The best results in this category were achieved by the models yolov9t (3 wrong detections) and yolov5nu (5 wrong detections), which at the same time maintain a high ratio of correct detections. In contrast, models such as FasterRCNN_ResNet50 (46 erroneous detections) and FasterRCNN_MobileNet_V3_Large (43 erroneous detections), despite high detection confidence, are less effective in environments where detection errors are crucial.

In terms of model size, yolov9t (4.74 MB) and yolov5nu (5.31 MB) are the most compact, making them ideal for use on edge computing devices with limited resources. In contrast, the FasterRCNN_ResNet50 (159.28 MB) and ssd300_vgg16 (135.96 MB) models require significantly more memory, limiting their use on devices with lower computing power.

6.2. Processing Time Results

Analysing the average image processing times, including preparation, inference and postprocessing times, there are several significant differences between the models that affect the object detection performance on the different devices.

Latency time for different models shows marked differences, depending on the type of model and the device on which it is run. YOLO models generally achieve lower latency times, making them more efficient in real-time detection.

In contrast, more advanced models, such as the FasterRCNN_ResNet50 and ssd300_vgg16, have higher latency, which may affect their suitability in resource-constrained systems.

YOLO models, such as yolov5su, yolov5nu, and yolo11n, offer the best latency times. For yolov5su, the latency on the NanoPi R5c device is 4,798.67 ms and on the Motorola device is 1,383.40 ms. yolov5nu and yolov8n also show short latency times of 1,679.09 ms and 1,814.28 ms on the NanoPi R5c, and 943.05 ms and 1,025.74 ms on the Motorola, respectively. In the case of yolov9s, the latency on the Motorola device is 2,487.11 ms, while on the T480 it reaches 849.09 ms, making it one of the less efficient in this group.

Compared to YOLO models, the more advanced Faster R-CNN models show noticeably higher latency times. FasterRCNN_ResNet50 on a NanoPi R5c device achieves a latency time of 48,204.22 ms, a significant difference from YOLO models such as yolov5su (4,798.67 ms) or yolov9s (5,168.67 ms). Even on a Motorola device, the FasterRCNN_ResNet50 still requires 25,555.80 ms to process an image, making it one of the slowest

models. The ssd300_vgg16 also shows higher latency, at 6,804.04 ms on the NanoPi R5c and 5,355.55 ms on the Motorola, making it slower than most YOLO models, but faster to process than the Faster R-CNN.

SSD architectures perform significantly better in terms of latency compared to Faster R-CNN models, although they are still slower than most YOLO models. On the NanoPi R5c device, the ssd300_vgg16 model achieves a processing time of 6,804.04 ms, making it slower than YOLOv5su and YOLOv9s, but still faster than FasterRCNN_ResNet50. The lighter version – ssdlite320_mobilenet_v3_large – significantly improves performance, reaching 728.61 ms, bringing it closer to YOLO models in terms of speed. All models on Motorola and T480 have differences in latency, but a noticeable trend is that the YOLO models offer better speed in detection, especially on devices with limited resources. However, the FasterRCNN_ResNet50 may offer better detection performance, despite the higher latency. Tables 7–9 show the top five models in terms of latency, evaluated for three different devices.

Table 7: Top 5 models in terms of latency for a NanoPi R5c

Model	Latency (ms)
ssdlite320_mobilenet_v3_large	728.61
yolo11n	1604.55
yolov5nu	1679.09
yolov8n	1814.28
yolov9t	1972.43

Table 8: Top 5 models in terms of latency for a Moto G50

Model	Latency (ms)
yolo11n	763.89
yolov5nu	943.05
yolov8n	1025.74
yolov9t	1205.59
ssdlite320_mobilenet_v3_large	1237.42

Table 9: Top 5 models in terms of latency for a Thinkpad T480

Model	Latency (ms)
ssdlite320_mobilenet_v3_large	185.41
yolov5nu	185.47
yolov8n	189.87
yolov8s	266.02
yolov10n	285.43

6.3. Resource Efficiency Results

This chapter presents an analysis of the resource efficiency of individual models in edge computing environments. The focus is on parameters such as CPU consumption and RAM consumption, power consumption and CPU temperature to assess how each model performs with hardware limitations in the context of object detection in edge computing systems.

The YOLO family models show relatively moderate CPU (in the 37-48% range) and RAM (in the 54-61% range) consumption on the NanoPi R5C device. These models also achieve relatively low CPU temperatures (in the 51-56°C range). Models based on the Faster R-CNN architecture, such as the FasterRCNN_ResNet50, have significantly higher CPU (89.35%) and RAM (72.32%) consumption, as well as higher CPU temperatures (61.34°C), which may indicate a higher system load during processing. The ssd300_vgg16 has very high CPU (94.39%) and RAM (67.85%) consumption and relatively high CPU temperatures (57.67°C), which may affect system performance when processing for long periods of time.

Table 10 shows the top five models in terms of resource efficiency for the NanoPi R5c device.

Table 10: Top 5 models in terms of resource efficiency – NanoPi R5c

Model	CPU (%)	RAM (%)	CPU Temp (°C)
yolov8s	37.72	57.96	53.72
yolov5su	38.07	54.81	51.80
yolo11s	40.58	61.42	55.59
yolov10s	42.12	59.92	55.42
yolov9s	42.22	59.42	55.22

Similar to the NanoPi R5c, YOLO models on Motorola show fairly high CPU (in the 81-83% range) and RAM (in the 72-74% range) consumption, with CPU temperatures ranging from 35.82°C to 43.69°C. yolo11n and yolo11s show the highest RAM consumption (81.64% and 82.13%) and the highest CPU temperatures (46.56°C and 45.86°C). Faster R-CNN models, such as the FasterRCNN_ResNet50, have relatively lower CPU consumption (60.02%) and RAM consumption (61.74%), with a CPU temperature of 43.46°C, suggesting that they are less taxing compared to other models, especially for devices such as Motorola.

The ssd300_vgg16 and ssdlite320_mobilenet_v3_large models show high CPU (71.81% and 87.37%) and RAM usage (76.82% and 70.63%), reflecting their computing demands. However, ssdlite320_mobilenet_v3_large has a relatively low CPU temperature (45 °C). Table 11 lists the top five most resource-efficient models for the Motorola Moto G50.

Table 11: Top 5 models in terms of resource efficiency - Moto G50

Model	Energy (W)	CPU (%)	RAM (%)	CPU Temp (°C)
FasterRCNN_MobileNet_V3_Large	2.43	75.64	61.84	46.27
yolo11s	2.43	82.66	82.13	45.86
ResNet50	2.51	60.02	61.74	43.46
yolov9t	2.55	82.21	74.76	43.69
yolov5nu	2.55	82.36	73.82	39.42

YOLO models achieve moderate CPU usage (66.85%–85.51%) and RAM usage (10.53%–11.3%), with CPU temperatures ranging from 59.09°C to 65.04°C on the ThinkPad T480 device. This suggests that these models are suitable for high-performance hardware, as they do not generate high loads on the CPU or memory.

Faster R-CNN models achieve lower levels of CPU (42%) but higher levels of RAM (18%) consumption.

The SSD architecture occupies a middle ground. It uses a similar or lower amount of CPU compared to YOLO models but requires more RAM. Compared to Faster R-CNN, SSD models consume more CPU resources but are more memory-efficient, using less RAM.

There is a large discrepancy in CPU and RAM consumption depending on the model. For example, for yolov5su the CPU consumption on the ThinkPad is only 71.23%, but the CPU temperature rises to 59.41°C. The FasterRCNN_ResNet50 has higher CPU consumption, indicating the higher computing requirements of this model compared to the others. Table 12 shows the top five models in terms of resource efficiency for the Thinkpad T480.

Table 12: Top 5 models in terms of resource efficiency - T480

Model	Energy (W)	CPU (%)	RAM (%)	CPU Temp (°C)
yolov5su	22.95	71.23	10.53	59.41
yolov9t	23.94	89.99	10.63	62.08
yolov8n	24.10	70.22	10.61	59.36
ssdlite320_ mobilenet_v3_large	24.12	75.68	13.81	65.61
yolo11s	24.15	72.36	11.30	65.04

6.4. Performance Metrics

Choosing the right model for object detection on edge computing devices requires consideration of both performance and resource consumption. Key factors include latency, CPU consumption, RAM consumption and model size, all of which have a direct impact on real-time application performance. Models that are more optimised for performance can offer a better balance between accuracy and response time. The following analysis shows the best and worst models that differ in these criteria, as well as their suitability in the context of edge computing devices.

Best Models:

- **yolo11n:**

This model achieved very good detection results of 97.30%, while having a high confidence rating of 0.8935. Its unique advantage is its small size of only 5.35 MB, which significantly distinguishes it from other models weighing 159 MB, 135 MB or 73 MB, respectively. In terms of latency, it ranked second on the NanoPI R5c device, while on the Motorola Moto G50 it was the best, coming in first place.

- **yolov5nu:**

This model also achieved high detection scores of 93.46%, with the highest confidence rating of 0.9142. Its size (5.31 MB) is similar to yolo11n, making it equally efficient in terms of memory resources. In terms of latency, yolo5nu came in third place to the NanoPI R5c device, second place to the Motorola Moto G50 and second place to the ThinkPad T480 laptop.

7. Conclusion

The conducted analysis confirms that YOLO-based models, particularly are well-suited for edge computing scenarios due to their favorable trade-off between detection accuracy, latency, and resource consumption.

Yolo11n stands out as one of the most efficient models in terms of low latency and modest hardware demands. Despite slightly higher memory usage compared to some other lightweight models, it consistently delivers fast and reliable performance across various devices. This makes it a strong candidate for real-time applications deployed on resource-constrained edge devices.

Yolov5nu offers a balanced approach, achieving good accuracy while maintaining acceptable CPU and RAM usage. Although its latency is relatively higher, its overall stability and detection performance justify its use in applications where consistent object recognition is more important than minimal delay.

Both models stand out for their excellent balance of high detection accuracy, computational efficiency and minimal hardware requirements. Their small size and low latency make them an ideal choice for real-time detection systems in environments with limited hardware resources, such as IoT devices, single board computers or smartphones.

When comparing the YOLO architecture to other approaches such as SSD and Faster R-CNN, YOLO consistently delivered the best overall performance on the tested dataset. For this reason, the recommended YOLO-based models are the most advisable choices. However, it is important to note that model behavior may vary depending on the specific characteristics of the target device.

References

[1] H. Feng, G. Mu, S. Zhong, P. Zhang, T. Yuan, Benchmark Analysis of YOLO Performance on Edge Intelligence Devices, Cryptography 6(2) (2022) 1-16, https://doi.org/10.3390/cryptography6020016.

[2] J. Tian, Q. Jin, Y. Wang, J. Yang, S. Zhang, D. Sun, Performance analysis of deep learning-based object detection algorithms on COCO benchmark: a comparative study, Journal of Engineering Applications of Science 71(76) (2024) 1-18, https://doi.org/10.1186/s44147-024-00411-z.

[3] M. Satyanarayanan, Edge Computing, Computer 50(10) (2017) 36-38, https://doi.org/10.1109/MC.2017.3641639.

[4] F. Wang, M. Zhang, X. Wang, X. Ma, J. Liu, Deep Learning for Edge Computing Applications: A State-of-the-Art Survey, IEEE Access 8 (2020) 58322–58336, https://dx.doi.org/10.1109/ACCESS.2020.2982411.

[5] T. Diwan, G. Anirudh, J. V. Tembhurne, Object detection using YOLO: challenges, architectural successors, datasets and applications, Multimedia Tools and Applications 82 (2023) 9243–9275, https://doi.org/10.1007/s11042-022-13644-y.

[6] A. Nazir, M. A. Wani, You Only Look Once – Object Detection Models: A Review, Proceedings of the 2023 10th International Conference on Computing for Sustainable Global Development (INDIACom) (2023) 1088–1095.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, A. C. Berg, SSD: Single Shot MultiBox Detector, Proc. European Conf. Computer Vision (ECCV) (2016) 21-37, https://doi.org/10.1007/978-3-319-46448-0_2.

[8] S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 39(6) (2017) 1137–1149, https://doi.org/10.1109/TPAMI.2016.2577031.

[9] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft COCO: Common Objects in Context, Proceedings of the 13th European Conference on Computer Vision (ECCV) (2014) 740-755, https://doi.org/10.1007/978-3-319-10602-1.