Int. J. Advanced Networking and Applications
Volume: 17, Issue: 03, Pages : 6960-6969 (2025) ISSN: 0975-0290
https://doi.org/10.35444/IJANA.2025.17309

A Review: Object Detection Algorithm for IoT and Edge Devices

Muhammed Shahil A K
Department of Computer Science, Sullamussalam Science College, Areekode
Email: mdshahilak@gmail.com
Muneer VK
Department of Computer Science, Sullamussalam Science College, Areekode
Email: vkmuneer@gmail.com
Rizwana K T
Department of Computer Science, Sullamussalam Science College, Areekode
Email: ktrizwana@sscollege.ac.in
UA Samshar
Department of Computer Science, Sullamussalam Science College, Areekode
Email: uasamshar@gmail.com

----------ABSTRACT----------

The rapid development of Internet of Things (IoT) and edge computing technologies has made a significant demand for deploying intelligent computer vision systems on limited resource devices. In computer vision, one of the crucial thing is object detection. It has significant developments through deep learning. But, most of the models are designed for high performance servers and GPUs, making them unsuitable for real time prediction on edge devices. This review aims to fill that gap by offering a comprehensive analysis of object detection algorithms optimized for IoT and edge computing environments. The paper present a structured analysis of traditional two-stage detectors like Faster R-CNN, R-CNN, Mask R-CNN, etc. lightweight designs like MobileNet, SqueezeNet, and EfficientDet, etc. as well as one-stage models like YOLO and SSD, etc. among others. The paper compares these models across the performance metrics like accuracy, model size, inference speed, and energy efficiency, within the constraints typical of edge deployments. And also analyze recent innovations including transformer-based detectors, few-shot learning strategies, and anchor free models. The focus is on hardware-aware optimizations, training methods, and technical advances in network architecture. Critical issues like latency, small object identification, model compression, and deployment over IoT devices are also highlighted in this evaluation. This paper provides a useful reference for researchers and developers those who are looking to develop efficient and scalable object detection in IoT and edge devices.

Keywords - **Internet of Things (IoT), Edge Computer, Computer Vision, Object Detection, Deep Learning, Real-Time Prediction, Lightweight designs, Low-Latency Detection, Model Compression.**

Date of Submission: July 03, 2025 Date of Acceptance: October 25,2025

1. INTRODUCTION

In computer vision, the object detection is an important part. It means finding and locating things like people, cars, animals, or other objects in images or videos. This technology is used in many areas such as security cameras, self-driving cars, smart farming, healthcare, and factories [1], [5], [10]. Deep learning has played an important role in object detection during the past ten years. Due to their speed and accuracy, models such as SSD, YOLO, and Faster R-CNN have gained a lot of popularity [4], [15], [24]. However, these versions were designed mainly for use with powerful computers that had a lot of RAM, a high-end GPU, and continuous electricity. That makes them hard to use on small or low-power devices.

There is an important change toward directly putting intelligence on resource constrained devices such smartphones, embedded microcontrollers, Jetson Nano, Raspberry Pi, and others due to the Internet of Things' (IoT) rapid growth and edge computer's development [2], [8], and [13]. Sensors and cameras are now being included into edge devices to enable local real-time decision-making with lowering latency, improving privacy, and using less bandwidth. However, traditional object detection models are computationally expensive, memory-intensive, and thus unsuitable for deployment in these edge settings [6], [12].

To solve this problem, many researchers started working on new object detection models that are smaller and faster, so they can run on edge devices and IoT systems. Some popular models that came from this effort are MobileNet, EfficientDet, YOLOv4 and YOLOv5, and EdgeYOLO [14], [18], [24]–[27]. These models try to give good accuracy while also using less memory and power. They do this by using smart techniques like depthwise separable convolutions, inverted residual blocks, automatic model design (neural architecture search), and training methods that help the model run better on smaller devices like phones or microcontrollers [16], [21], [17].

Int. J. Advanced Networking and Applications
Volume: 17, Issue: 03, Pages : 6960-6969 (2025) ISSN: 0975-0290
https://doi.org/10.35444/IJANA.2025.17309

This review paper gives a complete and clear overview of object detection algorithms that are made or improved to work well on edge devices and in IoT systems. The work looks at many types of object detection models, including those that use anchor boxes and those that don't. These models are compared based on important things like how accurate they are, how big the models are, how fast they work, and how much power they need. The paper also talk about new ideas in model design, the problems people face when using them on small devices, and how these models are being used in special areas like few-shot learning, real-time detection on small devices, and safety systems like smart surveillance [3], [6], [13].

The contributions of this paper are four-fold:

- It presents a systematic classification of object detection models relevant to iot and edge devices.
- It analyzes trade-offs between accuracy and efficiency using reported benchmarks across standard datasets.
- It identifies key challenges such as small object detection, latency reduction, and resource adaptation.

It describes possibilities for future study, such as edge-cloud hybrid deployments, federated learning, and transformer-based detectors.

2. LITERATURE REVIEW

Bhagyashri More and Snehal Bhosale present a detailed overview of various object detection algorithms in their paper, focusing on deep learning methods and architectures. They evaluate the speed and accuracy of models such as R-CNN, YOLO, SSD, and RetinaNet. The study shows that while YOLO and SSD are better for real-time tasks, models like Faster R-CNN give more accuracy. However, the paper mostly focuses on general detection and does not deeply explore edge deployment or IoT-specific constraints, leaving a gap in application for low-power devices [1].

Shaymaa Tarkan Abdullah et al. provide a survey of deep learning-based object detection techniques and also discuss how these techniques are applied in different fields like medical imaging and surveillance. The paper explains how models have evolved, starting from classical methods to CNN-based detectors. They also list open issues like hardware limitations, detection of small objects, and need for real-time solutions. The paper is useful but does not focus much on optimization for edge computing, which is crucial for IoT settings [2].

Yang Cao, Kaijie Jin, and Yaodong Wang analyze different categories of object detection methods, especially focusing on the evolution from R-CNN to more efficient models like YOLOv4. The study also emphasizes the importance of using datasets like COCO and PASCAL VOC to evaluate performance. While they discuss performance trends and technical improvements, the paper does not consider lightweight models or techniques specifically designed for edge or embedded devices [3].

Mohammad Samar Ansari et al. present a modern view of deep learning-based object detection models. Their review focuses on improving speed and detection accuracy and involves both anchor-based and anchor-free methods. The paper also touches on attention mechanisms, feature pyramid networks, and the rise of transformer-based detectors. Although very comprehensive in technical depth, it lacks practical deployment analysis for edge AI and IoT constraints like memory and power limitations [4].

Li Liu et al. offer one of the most detailed reviews of generic object detection, covering over 300 referenced papers. They discuss two-stage detectors (like Faster R-CNN) and one-stage models (like SSD and YOLO), and dive deep into their working principles. Technical elements like IoU, NMS, and loss functions are also explained in the paper. However, it mainly focuses on algorithm design and benchmark performance, without much attention to edge computing deployment or energy efficiency issues [5].

Zhimeng Xin et al. review the area of few-shot object detection (FSOD), which is about training models to detect new object categories using very few examples. The paper highlights major challenges like overfitting, poor generalization, and lack of benchmark datasets for FSOD. They compare existing FSOD methods and mention that many use meta-learning or attention mechanisms to boost learning. Although FSOD helps when training data is scarce, most current models are not ready for real-time or edge deployment due to their complexity. This creates an opportunity for research on lightweight FSOD models suitable for IoT use [6].

Anushka et al. give a summary of object detection methods based on deep learning. They describe important models like R-CNN, Fast R-CNN, Faster R-CNN, YOLO, and SSD. The paper explains how these models detect objects, compares their accuracy and speed, and discusses applications like face detection, surveillance, and robotics. Though informative, the review is poor technically and ignores edge computing-specific issues like model size and power consumption. According to the authors the future research, must focus on increasing accuracy and decreasing latency, both are essential for Internet of Things applications [7].

Shrinath Oza and Sunil Rathod explore how object detection and IoT can be combined to improve road safety. Their paper introduces a system that uses cameras and machine learning models to detect vehicles and alert drivers in real time to prevent accidents. The study uses traditional detection methods and does not deeply integrate advanced deep learning models like YOLO or MobileNet. Though promising, the system's reliance on high-end hardware limits its use on lightweight IoT devices. The paper reveals a gap in designing efficient, low-cost, real-time detection systems for road safety using deep learning on edge platforms [8].

For object detection, Uday C. Patkaret al. compared deep learning with machine learning techniques. They explain basic algorithms like KNN, SVM, and CNN, and then review models such as SSD, YOLO, and Faster R-CNN. The study emphasizes that deep learning requires more computing and training data but offers greater flexibility and accuracy. They briefly mention lightweight models but don't give details on edge deployment or IoT. Overall, the

review offers a good entry point but lacks a focused discussion on how to adapt models for low-power or real-time environments [9].

Zhao et al. provide a detailed review of deep learning-based object detection, dividing models into two categories: two-stage detectors like Faster R-CNN and one-stage detectors like SSD and YOLO. They also explain important concepts like anchor boxes, non-max suppression, and loss functions. The review includes model comparison based on COCO and VOC datasets and mentions improvements such as feature pyramids and focal loss. However, the focus is mostly on performance benchmarks, with little attention to how these models perform on edge devices or constrained hardware. There is limited discussion on energy efficiency or real-time applications [10].

Xiongwei Wu, Doyen Sahoo, and Steven Hoi present an in-depth review of recent advances in deep learning for object detection, covering topics like feature representation, multi-scale detection, context modeling, and hard example mining. They examine one-stage and two-stage models, such as SSD, YOLO, and variations of R-CNN. The paper also discusses new ideas like attention mechanisms and transformer-based detectors. However, while very comprehensive, the review gives little focus to practical deployment on edge devices or energy-efficient environments, which is crucial for IoT applications today [11].

Debani Prasad Mishra et al. offer a comparative study of different object detection algorithms, including YOLO, SSD, and R-CNN. The paper provides a tabular comparison of these models based on speed, accuracy, and dataset performance. It helps beginners understand how these models differ, but lacks depth in implementation details or architecture-specific insights. The paper does not address modern edge-specific models or optimization techniques like quantization or pruning, which are necessary for deployment in IoT and embedded systems [12].

Nimish Awalgaonkar et al. propose a deep learning and IoT-based vision system for improving safety in energy production environments. Their system detects anomalies in real-time using object detection models integrated with IoT infrastructure. The paper highlights the potential of combining CNN-based detectors with edge deployment, although it does not go into detail about the model architecture, optimization, or performance on constrained hardware. It emphasizes the value of real-time AI on-site, yet a clear performance evaluation for low-power devices is missing [13].

Andrew G. Howard et al. introduce MobileNets, a class of efficient deep neural networks designed for mobile and embedded vision applications. The model uses depthwise separable convolutions to reduce the number of parameters and operations, making it ideal for edge deployment. MobileNets allow adjustments using width and resolution multipliers to fit specific latency and accuracy needs. This model became the foundation for many lightweight detection systems, but the original paper mainly focuses on image classification. Adaptations for object detection are done in later models like SSD-MobileNet [14].

Wei Liu et al. present the SSD (Single Shot MultiBox Detector), which performs object detection in a single forward pass without using a region proposal stage. SSD is known for being faster than two-stage detectors like Faster R-CNN and performs well in real-time tasks. The model uses multiple feature maps to detect objects at different scales and is often combined with MobileNet for edge use. While SSD offers good speed-accuracy trade-offs, its performance on small objects is limited, and it requires post-processing steps like non-max suppression [15].

Forrest Iandola et al. introduced SqueezeNet, a lightweight CNN that achieves AlexNet-level accuracy with 50× fewer parameters. It uses a “fire module” made up of squeeze and expand layers, which reduces model size and memory usage significantly. Although originally designed for image classification, SqueezeNet is often used as a backbone for lightweight object detection models on embedded devices. However, the paper does not specifically address object detection performance or integration with detection frameworks like SSD or YOLO, leaving a gap for researchers targeting edge inference [16].

Kaiming He et al. proposed ResNet (Deep Residual Networks), which introduced the idea of residual learning to train very deep neural networks. ResNet can train networks with more than 100 layers by avoiding vanishing gradients using identity skip connections. Despite being designed for classification, ResNet has become a standard foundation for detectors such as Mask R-CNN and Faster R-CNN. ResNet models are highly computational for their capability, and in order to deploy them on edge devices, they typically need to be reduced or quantized to minimize complexity [17].

Mingxing Tan, Ruoming Pang, and Quoc Le proposed EfficientDet, an object detection model built using EfficientNet as a backbone and BiFPN (Bidirectional Feature Pyramid Network) for feature fusion. It achieves an optimal balance between accuracy and efficiency using compound scaling to scale the model size. The D0 to D7 EfficientDet models are suitable for a variety of platforms, including high-end GPUs and mobile devices. D0, D1 are especially optimized for edge deployment. This work is highly relevant to IoT scenarios, offering both lightweight architecture and scalable performance [18}.

Xingyi Zhou et al. introduced CenterNet, a model that views objects as keypoints. Instead of using anchor boxes, it detects object centers and regresses size and offset values. This anchor-free method simplifies training and reduces the need for manual anchor box tuning. CenterNet performs well for small object detection and is suitable for real-time applications. However, its complexity in decoding the final detections may be a challenge for very constrained edge devices unless further optimized [19].

Ross Girshick et al. presented R-CNN, one of the first successful applications of deep learning to object detection. R-CNN uses region proposals and a CNN-based classifier to detect objects. Though very accurate, it is slow and computationally intensive. Faster successors like Fast R-CNN and Faster R-CNN were the result of this. As R-CNN established the groundwork for deep object detectors, its

Int. J. Advanced Networking and Applications
Volume: 17, Issue: 03, Pages : 6960-6969 (2025) ISSN: 0975-0290
https://doi.org/10.35444/IJANA.2025.17309

high latency and usage of memory make its architecture unsuitable for real-time edge applications [20].

Mark Sandler et al. introduced MobileNetV2, a major upgrade over MobileNetV1. The model uses inverted residual blocks and linear bottlenecks to improve feature reuse and reduce memory cost. MobileNetV2 is faster and more efficient, making it ideal for object detection on edge devices. It is commonly used with lightweight detection heads like SSD or YOLOv5-lite. The paper focuses on classification tasks, but the architecture's design makes it well-suited for real-time detection in IoT and embedded systems [21].

Shihan Liu et al. proposed EdgeYOLO, a version of YOLO specifically optimized for edge devices. It removes anchor boxes and uses a lite decoupled head, re-parameterization, and adaptive loss functions to improve inference speed and accuracy. EdgeYOLO shows competitive performance with high FPS on edge platforms like NVIDIA Jetson, while maintaining good mAP. Its focus on small object detection and real-time performance makes it highly relevant for smart cameras, traffic systems, and drones in IoT settings [22].

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao introduced YOLOv4, a model that balances accuracy and speed. It uses CSPDarknet as a backbone, SPP and PAN for feature aggregation, and new training tricks like Mosaic augmentation, CIoU loss, and Self-Adversarial Training (SAT). YOLOv4 significantly outperforms YOLOv3 and is trainable on a single GPU. While powerful, its larger variants require optimization for use on edge devices, though it forms the basis for later lightweight variants like YOLOv5n and EdgeYOLO [23].

Rahima Khanam and Muhammad Hussain provide an in-depth look at the YOLOv5 architecture, give a thorough analysis of the YOLOv5 design, describing its modules, training methods, and organizational structure. Because YOLOv5 is written in PyTorch, it is more accessible and modular. Because it comes in a range of sizes nano, small, medium, big, and x-large users can choose the model that best suits their hardware. The paper highlights the improvements YOLOv5 brings in terms of speed and accuracy. YOLOv5n and YOLOv5s are especially noted for edge deployment due to their compact size [24].

In a deeper follow-up, Khanam and Hussain analyze YOLOv5's internal feature design, such as its CSP backbone, PAN neck, and detection heads. They discuss how different components contribute to YOLOv5’s speed and accuracy. Their study also evaluates performance trade-offs across its variants. The findings show that YOLOv5 offers flexible deployment, with lightweight models balancing mAP and latency for IoT devices. The paper offers valuable insights for researchers aiming to apply detection in constrained environments [25].

Shridhar H. et al**.** present a study on real-time object recognition using a hybrid deep learning model combining Single Shot Detector (SSD) and MobileNet V2. The model aims to balance accuracy and efficiency, overcoming limitations of traditional methods like SVM and HoG+SVM. By leveraging MobileNet V2’s depth-wise separable convolutions with SSD’s single-shot detection, the system achieves 97% accuracy with 0.12-second inference time, outperforming R-FCN and Mask R-CNN. Experiments on COCO and real-time images confirm its suitability for embedded vision. Although lacking detailed training parameters and energy analysis, the research offers a strong applied contribution toward lightweight, mobile-ready detection systems adaptable for future edge and transformer-based architectures [26].

Muhammed Shahil A. K. et al. present an innovative study on developing an automated catalogue system using real-time object detection with Single Shot Multibox Detector (SSD) and MobileNetV3. The model efficiently identifies and classifies humans, animals, and vehicles from CCTV footage, storing details such as timestamp, category, and confidence score using Firebase and OpenCV. A ReactJS-based interface allows easy data access and filtering. Experimental results achieved over 70% confidence accuracy, even with low-end hardware, demonstrating reliability for both security and wildlife monitoring. Although limited by small-scale testing and hardware constraints, the study effectively integrates deep learning with cloud and web technologies for intelligent surveillance automation [27].

3. BACKGROUND

3.1 Object Detection

In object detection, the system locates items in pictures or videos, such as people, cars, fruits, animals, or tools, and then uses boxes to indicate their location. Its not only what the image is, but also where it locate. Many real-world applications, including traffic management, smart farming, security surveillance, and healthcare, benefit from object detection.

There are two main tasks in object detection:

1. Classification – recognizing what the object is.
2. Localization – drawing a bounding box around the object.

Popular deep learning-based models like Faster R-CNN, SSD, and YOLO can do both tasks simultaneously. Hand-crafted features like HOG, SIFT and traditional machine learning techniques like SVM were utilized in object detection in the past, however deep learning has recently out performing these outdated techniques in terms of speed and accuracy.

3.2 IoT and Edge Computer Fundamentals

Instead of transmitting all data to cloud servers, edge computing involves processing data closer to where it is created. This is crucial in Internet of Things (IoT) systems, where a large number of tiny devices, such as sensors, cameras, Raspberry Pi, and smart home appliances, are linked to the internet and regularly collect data.

Sending all data to the cloud for processing in IoT setups can result in latency and bandwidth waste. Edge computing solves this problem by processing data locally on the device or nearby (at the "edge" of the network). This allows:

- Faster decision-making
- Better privacy
- Lower network load

Int. J. Advanced Networking and Applications
Volume: 17, Issue: 03, Pages : 6960-6969 (2025) ISSN: 0975-0290
https://doi.org/10.35444/IJANA.2025.17309

3.3 Resource Constraints in Edge Devices

In opposition to powerful computers and cloud servers, edge devices have constrained resources:

- Low processing power (e.g., ARM CPUs)
- Limited storage and memory (RAM)
- Battery constraints (for portable or remote sensors)
- Heat limits (no cooling fans or heat sinks)

Due to these limitations, using large object identification models on edge devices, such as YOLOv4 or Faster R-CNN, is challenging. Lightweight models such as MobileNet, EfficientDet-D0, YOLOv5n, or SqueezeNet are thereby made to operate faster while using less memory and energy [26], [27].

3.4 Traditional vs. Deep Learning Based Models

Traditional object detection methods used features like:

- Haar cascades
- HOG (Histogram of Oriented Gradients)
- SIFT (Scale-Invariant Feature Transform)

SVM and Decision Trees, two simple machine learning classifiers, were used with these. Because they were quick, they lacked accuracy, particularly when objects changed in size, shape, or illumination.

As a result, object detectors that use deep learning learn straight from data. CNN-based models such as:

- R-CNN, Fast R-CNN, Faster R-CNN
- YOLO (You Only Look Once)
- SSD (Single Shot Detector)

offer much higher accuracy and better adaptability. The problem is: these models are often too large and slow for use on small edge devices without modifications.
To make them suitable, researchers use:

- Model compression
- Quantization
- Lightweight architectures
- Knowledge distillation

In Object detection the first algorithm is Viola-Jones Algorithm in 2001. Now its reach to 2025. The timeline of object detection models are given in the figure 1.

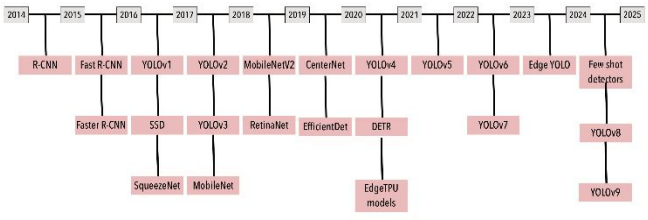

Fig 1: Timeline of Object Detection Models from 2014 to 2024. But the first Object Detection Algorithm is Viola-Jones Algorithm introduced in 2001.

4. CLASSIFICATION OF OBJECT DETECTION ALGORITHMS

Object detection models can be grouped in different ways based on how they work. In this section, it explain the main types how they detect objects, how fast they are, and how good they are for edge devices and IoT. Understanding these categories helps in choosing the right model for real-time and resource-limited systems.

4.1 Two-Stage Detectors

Two-stage object detection models follow a structured pipeline: They first create region suggestions, which are potential places for objects to be found, and then they categorize and improve those suggestions. Although these models are known for their strong performance and high accuracy in complicated situations, their high latency and processing requirements make them unsuitable for deployment on edge devices. R-CNN established the groundwork for two-stage detectors by applying convolutional neural networks to each region suggested by a selective search algorithm, thereby introducing deep learning into object detection [20]. Although it improved detection accuracy over traditional methods, R-CNN was extremely slow because it ran a CNN separately on thousands of regions per image.

This was addressed by Fast R-CNN, which greatly decreased inference time while maintaining accuracy by first applying the CNN to the full image and then utilizing ROI pooling to extract features for each region [10]. A Region Proposal Network (RPN) was then added by Faster R-CNN to significantly improve this pipeline by eliminating the requirement for selective search and automatically producing high quality region suggestions [10], [11]. This resulted in a nearly end-to-end detection process that was far quicker than its predecessors. Even if these models are accurate, they still need powerful GPUs and a lot of memory, which makes them appropriate for server-based or cloud-based settings where accuracy is more crucial than speed in real time, like industrial surveillance systems or medical imaging. However, they are not feasible for deployment on edge devices such as the Raspberry Pi or Jetson Nano due to their high memory and computational demands [12], [23]. The architecture of the Two-stage Object detection is given in the figure 2.

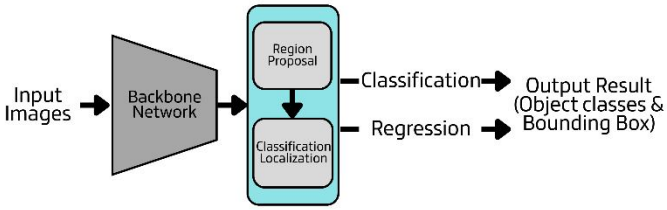

Fig 2: Architecture of Two-Stage object detection model [1].

4.2 One-Stage Detectors

One-stage detectors are much faster than two-stage models since they complete object localization and classification in a single forward pass. They are extensively utilized in applications such as mobile systems, surveillance, and driverless cars due to their real-time performance and straightforward architecture. YOLO is one of the most widely used families of one-stage detectors since it does not require a separate area proposal stage and can predict bounding boxes and class probabilities from the entire image. YOLO has developed over the years to successfully strike a balance between accuracy and speed. Additionally, YOLOv5 is very adaptable for a variety of situations because it provides a number of model variations that may be used on both resource-constrained edge devices and high-end servers [24], [25].

Int. J. Advanced Networking and Applications
Volume: 17, Issue: 03, Pages : 6960-6969 (2025) ISSN: 0975-0290
https://doi.org/10.35444/IJANA.2025.17309

The Single Shot MultiBox Detector (SSD) is another important one-stage approach that improves performance, particularly on medium and large objects, by using feature maps at several scales to detect objects of different sizes [15], [26], [27]. Also, RetinaNet introduced the idea of focused loss, which improves detection performance in dense situations by concentrating more on rare or difficult to detect objects, hence assisting the model in managing class imbalance during training [10]. The accuracy of these models may still fall low of advanced two-stage detectors in situations when they are unable to recognize small, overlapping, or blocked objects, despite their great inference speed. The architecture of the Two-stage Object detection is given in the figure 3.

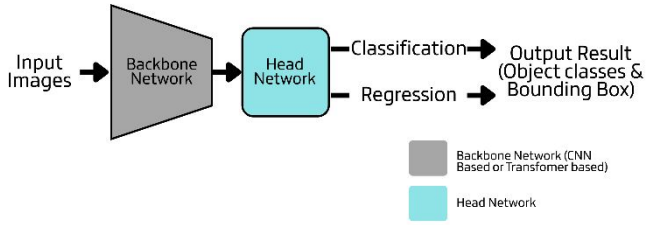

Fig 3: Architecture of One-Stage object detection model [1].

4.3 Anchor-Based and Anchor-Free Models

Whether or not they use anchor boxes, object detection models can also be grouped according to how they clearly create bounding boxes. Anchor-based models assist in object location prediction by using specified anchors, which are bounding boxes of different sizes and aspect ratios, during training. The best-fitting anchors around items in the image are modified and classified by these models. YOLO, SSD, and Faster R-CNN are well-known anchor-based detectors [10], [15], and [24]. In general, these models do well at identifying objects of various sizes and shapes, particularly in intricate scenarios with several occurrences. One disadvantage of anchor-based approaches is that they frequently face higher computational overhead, particularly during training, and call for manual anchor parameter modification. Anchor-free models, on the other hand, do not require anchors at all. They improve the training process by directly looking for critical points or object centers rather than estimating distances from predefined boxes. The two most common are CenterNet and CornerNet, which return to the bounding box dimensions after predicting an object's center or corners [19]. In general, anchor-free models are lighter, simpler to train, and more flexible when applied to new datasets. They might, have trouble detecting items that are extremely small or densely packed, as accurate localization becomes more challenging in the absence of established anchors.

4.4 Transformer-Based Models

By utilizing architectures that were first effective in natural language processing, transformer-based models set up a new path in object detection. Instead than depending entirely on convolutional layers, these models include self-attention mechanisms to understand the links between various picture elements. Being the first to provide an end-to-end object detection framework without the need for post processing processes like non-maximum suppression (NMS), DETR (Detection Transformer) is the most well-known model in this category. Rather, DETR uses transformer blocks to directly transfer input features to final object predictions, which improves efficiency in intricate scenes with overlapping objects [18]. Other models such as EfficientDet with attention modules also integrate transformer ideas to improve detection accuracy. While these models perform well in terms of accuracy and scene awareness, they have significant limitations for edge and Internet of Things applications. Unless they are much reduced in size or simplified, transformer-based detectors are not feasible for deployment on lightweight edge devices because to their high latency, computational cost, and memory requirements.

4.5 Edge-Optimized and Lightweight Models

Some object detection models are specifically made or tuned to function well on low-power edge gadgets like smartphones, Jetson Nanos, and Raspberry Pi. These models are perfect for real-time Internet of Things applications since they are small, require little computing power, and use minimal memory [26]. Among the first models in this category is SqueezeNet. It is extremely compact under 0.5MB while still offering decent accuracy for basic detection tasks [16]. MobileNet and its improved version MobileNetV2 are also widely used as backbones in lightweight detection models like SSD or Tiny-YOLO. They use depthwise separable convolutions to reduce computation, which helps in lowering both size and power consumption [14], [21], [27]. EfficientDet-D0 is the smallest model in the EfficientDet family, offering a strong balance between accuracy and efficiency through compound scaling and the use of BiFPN (Bidirectional Feature Pyramid Network) [18]. More recently, models like EdgeYOLO have been introduced specifically for edge inference. EdgeYOLO is an anchor-free detector with a simplified head and backbone that makes it suitable for devices like the Jetson Xavier or Nano, delivering high FPS while keeping power usage low [23]. Also, because of their tiny size, quick inference speed, and respectable detection performance, YOLOv5n and YOLOv5s the smallest versions of the YOLOv5 family are frequently employed in embedded AI systems [25]. These models are crucial for edge computing scenarios because they demonstrate that effective object detection is achievable even on constrained hardware.

Table captions appear centered above the table in upper and lower case letters. When referring to a table in the text, no abbreviation is used and "Table" is capitalized.

5. COMPARATIVE ANALYSIS

When choosing an object detection model for IoT or edge computing, it can’t look only at accuracy but also need to think about how fast, small, and efficient the model is. A model that runs well on a powerful cloud server may fail on a low-power device like Raspberry Pi. This section compares detection models in terms of accuracy (mAP), speed (FPS), size, and suitability for edge devices, based on hardware and application use.

Int. J. Advanced Networking and Applications
Volume: 17, Issue: 03, Pages : 6960-6969 (2025) ISSN: 0975-0290
https://doi.org/10.35444/IJANA.2025.17309

5.1 Comparison by Accuracy, Speed, and Model Size

Many models give good accuracy, but the difference comes when it look at how fast they run and how heavy they are. For example:

- YOLOv5s reaches 56.8% mAP and runs at over 60 FPS on GPU. It has a size of just ~14 MB, which is quite good for edge devices. It gives a good balance between accuracy and speed [25].
- MobileNet-SSD is smaller, only ~6 MB, and runs well on Raspberry Pi 5, but gives lower accuracy (~31.2% mAP) [14], [15], [27]. Still, it is used in simple applications like fruit detection or smart cameras.
- EdgeYOLO, which is designed for real-time edge inference, achieves about 52.3% mAP and 70+ FPS on Jetson Xavier NX, while being just ~10 MB in size. It is a very strong choice for drones, surveillance, and robotics [23].
- In contrast, Faster R-CNN gives high accuracy (>70% mAP) [10], but it is very large (170+ MB) and slow (7–10 FPS). It is not suitable for real-time or low-power edge deployment.

Comparison of popular object detection models based on mAP, speed and model size are given in the table 1.

Model	mAP (%)	Speed (FPS)	Model Size (MB)	Edge Suitability
YOLOv5n	45.7	90+	~4.5	Excellent
YOLOv5s	56.8	60+	~14	Excellent
EdgeYOLO	52.3	70+	~10	Excellent
SSD-MobileNet	31.2	25–40	~6	Good
EfficientDet-D0	34.5	20–30	~4.1	Good
RetinaNet	39.1	18–24	~145	Moderate
Faster R-CNN	70.4	7–10	170+	No
DETR	42.0	<10	100+	No

Table 1: Popular object detection models based on mAP (accuracy), inference speed (FPS), and model size.

5.2 Hardware Performance Comparison

Object detection models behave differently on different devices. A model that runs well on cloud GPU may not run at all on Raspberry Pi.

- Only very compact models, such as YOLOv5n, MobileNet-SSD, or EfficientDet-D0, can operate on the Raspberry Pi 5 and even then, only at 2-4 frames per second. Since the Pi lacks a GPU, all operations are handled by the CPU, which is slow and power-constrained.
- On Jetson Xavier NX, it get real-time speed. For example, YOLOv5s runs at around 60 FPS, and EdgeYOLO reaches 70+ FPS, both with acceptable power usage [22], [23].
- Depending on the task and resolution, high-end smartphones with Snapdragon CPUs, such as the YOLOv5n or SSD-MobileNet devices, can operate easily with TFLite or ONNX, providing 30-45 frames per second.

`Although they are costly and non-portable, cloud GPUs such as the NVIDIA A100 can execute any model of Faster R-CNN, DETR, or YOLOv8 at 100+ frames per second.

Comparison of different edge devices with cloud GPU are given below in the table 2.

Feature	Raspberry Pi 5	NVIDIA Jetson Xavier NX	Cloud GPU Server (e.g., NVIDIA A100)
CPU	Quad-core Cortex-A76 @ 2.4 GHz	6-core ARM v8.2 64-bit @ 1.4 GHz	AMD EPYC / Intel Xeon (32+ cores @ 2.6+ GHz)
GPU	Broadcom VideoCore VII (No CUDA)	384-core Volta GPU with 48 Tensor Cores	NVIDIA A100 40GB/80GB with 6912 CUDA cores
RAM	4–8 GB LPDDR4	8–16 GB LPDDR4x	40–80+ GB HBM2e (GPU) + 128–256 GB DDR4 (CPU)
Storage	microSD or USB 3.0 SSD	eMMC / SSD	NVMe SSDs / HDDs (10TB+)
Power Consumption	~5–8 W	~10–15 W	~400–600 W (GPU) + 300 W (CPU & system)
Cost (Approx.)	₹7,000–₹10,000 (~$90–$120)	₹45,000–₹60,000 (~$500–$750)	₹8,00,000+ (~$10,000+)
Inference Speed	~1–2 FPS (YOLOv5n)	~10–20 FPS (YOLOv5s, EdgeYOLO)	100+ FPS (YOLOv5x, YOLOv8)
Ideal For	Basic IoT tasks, low-power edge AI	Real-time edge AI, robotics, drones	Large-scale training, cloud inferencing
Limitations	No GPU acceleration, low RAM	Medium GPU, limited thermal cooling	High power cost, needs internet, not portable

Table 2: Comparison of Raspberry Pi vs NVIDIA Jetson vs Cloud/Server for Object Detection.

5.3 Use-Case-Based Comparison

The best model depends on the use case. A farm, a drone, and a factory may need very different types of detection. The best model for different use cases and their reasons are given in the table 3.

Use Case	Recommended Model	Why
Smart Surveillance	YOLOv5s / EdgeYOLO	Real-time detection, good balance of speed and size
Smart Farming / Fruit Tray	EfficientDet-D0 / YOLOv5n	Low latency, works on small edge hardware
Traffic Monitoring	YOLOv5s / SSD-MobileNet	Fast inference, handles moving objects well
Drones / Aerial Imaging	EdgeYOLO / CenterNet	Good FPS and object detection from varied angles
Healthcare Monitoring (IoT)	YOLOv5n / MobileNet-SSD	Low power and lightweight models are ideal
Factory /	YOLOv5s /	Stable accuracy, runs on

Industry Safety	EfficientDet	local Jetson/GPU clusters

Table 3: Best object detection models recommended for different use cases.

6. CHALLENGES

Considering significant advancements, object detection still faces many real-world obstacles when used in edge and Internet of Things devices. Models need to manage small or blurry images, function well with limited hardware, and take security and privacy into account. This section explains some common limitations discussed in recent research.

6.1 Small Object Detection

It is still very difficult to detect little objects like screws, fruits, or license plates, particularly when they are far away or overlap with other objects. Due to the loss of small features in their feature maps at lower resolutions, models such as YOLO or SSD might overlook these items [11], [18]. Even high-accuracy models like Faster R-CNN struggle with very small objects unless trained on high-resolution data. Papers like [10] and [19] mention that newer models use feature pyramid networks (FPN) or multi-scale detection to improve this, but it still remains a problem especially on low-power devices that cannot handle large input sizes.

6.2 Trade-off Between Speed and Accuracy

There is always a balance between making the model faster or making it more accurate it often can't get both at the same time. For example, YOLOv5n runs at 90+ FPS but gives lower accuracy (~45.7% mAP), while YOLOv5s is more accurate (~56.8% mAP) but a bit slower [25]. For real-time applications like surveillance or drone vision, speed is more important. But for tasks like medical detection, accuracy is critical. Choosing the right model always depends on what the user wants to prioritize [14], [23], [27].

6.3 Power and Thermal Constraints

Edge devices like Jetson Nano or Raspberry Pi 5 don't have fans or strong CPUs. Running deep learning models on them can cause overheating or drain the battery quickly [22]. Models like DETR and Faster R-CNN are too heavy and consume too much power for such devices [10], [24]. Even optimized models like EdgeYOLO or EfficientDet-D0 must be carefully selected based on the device's thermal and power limits [26], [27]. Some research now looks into energy-efficient design, like using quantized models or reduced precision (e.g., INT8) to cut power usage.

6.4 Data Annotation in IoT Scenarios

A large amount of labeled data, where each object in each image is accurately marked is required to train detection models. But this type of data is not readily accessible in IoT systems, particularly in remote or customized contexts. Labeling thousands of photos by hand takes a lot of time. In cases like smart farming or factory-specific inspections, researchers must create their own datasets, which limits the scale and speed of development [5], [18]. Some papers suggest using semi-supervised learning or transfer learning to reduce data needs, but this area still needs more work.

6.5 Security and Privacy Issues on Edge Devices

Sensitive data, such as live video feeds, personal movements, or health information, is frequently handled by edge devices. This data can be stolen or misused by hackers if the gadget is not secure. Edge devices could not have advanced firewalls or encryption like cloud servers do. A security breach could leak real-time object detection output, which is dangerous in smart homes or surveillance systems [13], [23]. Researchers now look at secure model deployment, encrypted communication, and model watermarking to prevent misuse, but these are still early solutions.

7. CONCLUSION

In this paper, it reviewed and compared three major object detection models: YOLOv4, YOLOv5, and EdgeYOLO. Each of these models has its own strengths, depending on where and how it is used. YOLOv4, released in 2020, is known for achieving a good balance between accuracy and speed on GPU-powered systems. It uses the CSPDarknet-53 backbone and many training tricks like Mosaic augmentation and Self-Adversarial Training to boost accuracy [24]. However, it is built on the Darknet framework, which makes it harder to export and run on edge devices like Jetson or Raspberry Pi.

Whereas, YOLOv5 is fully constructed in PyTorch and offers a variety of model sizes (n, s, m, l, and x), enabling users to select the one that best suits their device [25]. For instance, YOLOv5x (extra-large) is highly accurate but more appropriate for GPU servers, but YOLOv5n (nano) is small and performs well on mobile or edge devices. The model also includes features like auto-anchor tuning, fast inference, easy exporting to ONNX or TFLite, and half-precision training. It is currently one of the most developer-friendly models and widely used for both cloud and edge deployment [25].

EdgeYOLO is a newer model designed specifically for real-time inference on low-power devices like the NVIDIA Jetson Xavier or Nano. Unlike YOLOv4 and YOLOv5, it is anchor-free, which means it doesn't rely on predefined bounding box sizes. This makes it faster and more suitable for CPU-limited edge environments [23]. It uses a light decoupled head and introduces Hybrid Random Loss, which helps detect small and rare objects more accurately. EdgeYOLO achieves more than 50% mAP on the COCO dataset while maintaining 30–34 FPS on Jetson Xavier a strong result for edge AI tasks.

From this comparison, the paper observed that while YOLOv4 performs well in accuracy, it is not suitable for edge due to its size and complexity. Depending on the model size selected, YOLOv5 offers an adaptable system that can operate on both cloud and edge platforms. While YOLOv5s provides a superior mix of speed and accuracy for real-time applications, YOLOv5n (nano), for example, is less than 5MB and can run at more than 60 FPS on

lightweight devices. When speed is crucial and the target hardware has limited power and memory, as in drones, smart farming, or embedded surveillance systems, EdgeYOLO performs [23], [26].

In terms of deployment, the paper recommend using YOLOv5s or YOLOv5n for general IoT applications that require a good mix of speed and performance. For GPU servers or research setups, YOLOv4 or YOLOv5x is better due to their high accuracy. EdgeYOLO is a great option if real-time detection on edge devices is the goal because it is optimized for low-latency, low-resource situations while maintaining competitive accuracy [27].

There are still certain difficulties, too. First, it is still difficult to detect small objects, particularly in noisy or low-resolution settings. Small objects like tools, fruits, or faraway people are sometimes missed by models. Secondly, precision and speed are constantly traded off. Usually, a model's accuracy decreases as it gets faster. Third, heavy models cannot operate without overheating or draining the battery since many edge devices have power and thermal constraints [14], [18], and [23]. Another issue is data annotation; producing labeled data for IoT contexts is expensive and time-consuming. Lastly, security and privacy are big concerns. Edge devices process sensitive data and often lack strong protection mechanisms [13].

Future studies should concentrate on models for edge use that are accurate, secure, and lightweight [26], [27]. Standardized evaluation tools for practical edge situations are also required, as are privacy-conscious AI methods that safeguard user information. The sector is expanding quickly overall, and the paper could see stronger object detection systems for IoT and edge devices as a result of improved technology and more intelligent models.

8. REFERENCE

[1]. B. More and S. Bhosale, *"A Comprehensive Survey on Object Detection Using Deep Learning," Revue d'Intelligence Artificielle*, vol. 37, no. 2, pp. 149–157, Apr. 2023, doi: 10.18280/ria.370217.

[2]. S. T. Abdullah, B. T. AL-Nuaimi, and H. N. Abed, *"A Survey of Deep Learning-Based Object Detection: Application and Open Issues," International Journal of Nonlinear Analysis and Applications*, vol. 13, no. 1, pp. 607–621, 2022, DOI: 10.22075/ijnaa.2022.6525.

[3]. Y. Cao, K. Jin, and Y. Wang, *"A Survey of Deep Learning-based Object Detection,"* in *Proc. IEEE MLBDBI*, 2021, pp. 77–82, DOI: 10.1109/MLBDBI54094.2021.00120.

[4]. M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, *"A Survey of Modern Deep Learning-Based Object Detection Models," Digital Signal Processing*, vol. 126, p. 103514, Jun. 2022, DOI: 10.1016/j.dsp.2022.103514.

[5]. L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, *"Deep Learning for Generic Object Detection – A Survey," International Journal of Computer Vision*, vol. 128, no. 2, pp. 261–318, Sep. 2019, DOI: 10.1007/s11263-019-01247-4.

[6]. Z. Xin, S. Chen, T. Wu, Y. Shao, W. Ding, and X. You, *"Few-Shot Object Detection – Research Advances and Challenges," Information Fusion*, Jul. 2024, DOI: 10.1016/j.inffus.2024.102307.

[7]. Anushka, C. Arya, A. Tripathi, P. Singh, M. Diwakar, and K. Sharma, *"Object Detection using Deep Learning: A Review," Journal of Physics: Conference Series*, vol. 1854, no. 1, p. 012012, 2021, DOI: 10.1088/1742-6596/1854/1/012012.

[8]. S. Oza and S. Rathod, *"Object Detection using IoT and Machine Learning to Avoid Accident and Improve Road Safety," International Journal of Engineering Research & Technology (IJERT)*, vol. 9, no. 6, pp. 1025–1029, Jun. 2020, DOI: 10.17577/IJERTV9IS060640.

[9]. U. C. Patkar, S. B. Shrives, U. S. Patil, A. J. Patankar, N. Jain, M. Kumari, and A. Chandhoke, *"Object Detection using Machine Learning and Deep Learning," International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 1s, pp. 466–473, Aug. 2023, https://ijisae.org/index.php/IJISAE/article/view/3483.

[10]. Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, *"Object Detection with Deep Learning: A Review," IEEE Access*, vol. 7, pp. 93565–93591, 2019, DOI: 10.1109/ACCESS.2019.2911239.

[11]. X. Wu, D. Sahoo, and S. C. H. Hoi, *"Recent Advances in Deep Learning for Object Detection," Neurocomputing*, vol. 396, pp. 39–64, Jul. 2020, DOI: 10.1016/j.neucom.2020.01.085.

[12]. D. P. Mishra, K. K. Rout, S. Mishra, and S. R. Salkuti, *"Various Object Detection Algorithms and Their Comparison," Indonesian Journal of Electrical Engineering and Computer Science*, vol. 29, no. 1, pp. 330–338, Jan. 2023, DOI: 10.11591/ijeecs.v29.i1.pp330-338.

[13]. N. Awalgaonkar, H. Zheng, and C. S. Gurciullo, *"A Deep Learning and IoT-Based Computer Vision System to Address Safety and Security of Production Sites in the Energy Industry," International Research Journal on Advanced Engineering and Management*, Dec. 2024, DOI: 10.48550/arXiv.2003.01196.

[14]. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv preprint*, arXiv:1704.04861, 2017, DOI: 10.48550/arXiv.1704.04861

[15]. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *"SSD: Single Shot MultiBox Detector,"* in *Computer Vision – ECCV 2016*, Springer, 2016, pp. 21–37, DOI: 10.1007/978-3-319-46448-0_2.

[16]. F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, *"SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size," arXiv preprint*, arXiv:1602.07360, 2016, DOI: 10.48550/arXiv.1602.07360.

[17]. K. He, X. Zhang, S. Ren, and J. Sun, *"Deep Residual Learning for Image Recognition,"* in *Proc. IEEE CVPR*, 2016, pp. 770–778, DOI: 10.1109/CVPR.2016.90.

[18]. M. Tan, R. Pang, and Q. V. Le, *"EfficientDet: Scalable and Efficient Object Detection,"* in *Proc. IEEE/CVF CVPR*, 2020, pp. 10781–10790, DOI: 10.1109/CVPR42600.2020.01079.

[19]. X. Zhou, D. Wang, and P. Krähenbühl, *"Objects as Points," arXiv preprint*, arXiv:1904.07850, Apr. 2019, DOI: 10.48550/arXiv.1904.07850.

[20]. R. Girshick, J. Donahue, T. Darrell, and J. Malik, *"Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,"* in *Proc. IEEE CVPR*, 2014, pp. 580–587, DOI: 10.1109/CVPR.2014.81.

[21]. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *"MobileNetV2: Inverted Residuals and Linear Bottlenecks,"* in *Proc. IEEE/CVF CVPR*, 2018, pp. 4510–4520, DOI: 10.1109/CVPR.2018.00474.

[22]. A. G. Howard et al., *"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv preprint*, arXiv:1704.04861, 2017, DOI: 10.48550/arXiv.1704.04861.

[23]. S. Liu, J. Zha, J. Sun, Z. Li, and G. Wang, *"EdgeYOLO: An Edge-Real-Time Object Detector,"* in *Proc. IEEE Chinese Control Conference (CCC)*, 2023, DOI: 10.23919/CCC58697.2023.10239786.

[24]. A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *"YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv preprint*, arXiv:2004.10934, Apr. 2020, DOI: 10.48550/arXiv.2004.10934.

[25]. R. Khanam and M. Hussain, *"What is YOLOv5: A Deep Look into the Internal Features of the Popular Object Detector," arXiv preprint*, arXiv:2407.20892, Jul. 2024, DOI: 10.48550/arXiv.2407.20892.

[26]. Shridhar H., Sumalata M. V., Thushara M., Ashwini D. N., M. Suma, R. Premananda, and S. S. Harakannanavar, *"Development of Object Recognition Model Using Machine Learning Algorithms on MobileNet V2," International Journal of Advanced Networking and Applications*, vol. 15, no. 2, pp. 5908–5914, 2023. ISSN: 0975-0290.

[27]. Muhammed Shahil A. K., Jalwa V. P., Afnan M. K., Rababa Kareem Kollathodi, and Muneer V. K., *"Automated Catalogue Using Object Detection," International Journal of Advanced Networking and Applications (IJANA)*, vol. 16, no. 1, pp. 6263–6269, 2024. ISSN: 0975-0290.