NIJEC 4 (2) (2025) 13-30

NUML International Journal of Engineering and Computing

Journal Homepage: NIJEC.NUML.edu.pk
Print ISSN: 2788-9629

A Review of Real-Time Deep Learning–Based Object Detection Models for Resource-Constrained Embedded Systems

Awais Shah[a*]

[a] Department of Computer Science, Wapda Post Graduate College, Terbela, Pakistan

ARTICLE INFO

Article History:
Received 1 January 2026
Accepted 4 January 2026
Available Online 6 January 2026

Keywords:
Object Detection, Embedded Systems, Edge AI, Lightweight Deep Learning Models, Resource-Constrained Devices

Funding:
This research received no specific grant from any agency in public, commercial or non-for-profit sector

Conflict of Interest:
The author has declared no potential conflicts of interest and falsification/fabrication of data with respect to the research, authorship, and/or publication of this article.

ABSTRACT

Computer vision relies heavily on object detection; from autonomous drones to industry monitoring it is everywhere. However, despite these advances when one wants to implement these cutting-edge object detection models in embedded system it proves to be difficult due to the limitations in processing power, memory and energy consumption. A detailed examination of real-time object detection models designed for resource-constrained devices is presented in this paper. We investigate widely used one stage detectors like SSD (Single Shot Multi Box) and YOLO (You Only Look Once). In addition, we also discuss model compression methods like knowledge distillation, pruning and quantization, which enable efficient deployment on embedded systems

1. Introduction

Object Detection is the classifying and localization of objects in images and video streams [1]. The term neural network was first suggested in 1940s [2] to mimic the human brain, to solve the general problem of learning in a principled manner. In the 1980s and early 1990s the introduction of back propagation [3] played a vital role in the widespread popularity of neural networks. However, in the early 2000s it became less popular due to the unavailability of the large-scale annotated data, severe over fitting, scarcity of computational power and negligible results. After addressing these limitations of data and hardware, neural network resurrected with more refined algorithms.

Traditional object detection relied on manually extracted features and classical machine learning techniques such as Viola-Jones [4], SIFT [5], HOG combined with SVM classifier [6]. They influenced the market for decades. However, while effective for constrained scenarios, they struggled in real time scenarios due to the extensive need for manual feature engineering.

The introduction of Deep Learning and CNNs (convolution neural networks) [7] revolutionized object detection entirely. The CNNs showed the capability to learn hierarchical image feature representation directly without the use of manually making features. The game changed with the

https://doi.org/10.52015/nijec.v4i2.110

appearance of AlexNet [8], achieving record-breaking results for image classification, going beyond the capability of human intelligence for some tasks on the benchmark dataset ImageNet.

In spite of these breakthroughs achieved by researchers within the object detection community, there still may be a substantial ‘‘deployment gap’’ remaining. Reliable object detection models are designed in a manner that consumes costly GPUs and tends to be RAM and power-hungry. However, the models need to work in low-end hardware like Raspberry Pi boards, NVIDIA Jetson Nano boards, and other ARM embedded systems on which battery power and time performance, normally expressed in frames per second, typically a value of ≥30 frames per second or better, are the functional specifications of usability and safety. The trade-off between the model's accuracy, measured by metrics like "mAP," and efficiency, measured by "FPS and memory," can be challenging when designing Embedded Systems. There have been two solutions proposed in bridging this particular gap:

1.1 Lightweight Architectural Design

Designing efficient backbone architectures from scratch using approaches like depth-wise separable convolution [9], channel shuffling [10], inverted residuals [11], or neural architecture search while reducing parameters and retaining detection accuracy.

1.2 Model Optimization

This can be achieved through various post-training techniques that aim to reduce computational and memory footprints without substantially reducing the model's accuracy. Such techniques include structured and unstructured pruning [12] which aim at eliminating redundant parameters; quantization [13], meant for precision reduction from FP32 to INT8 or lower; and knowledge distillation [14], knowledge transfer from large teacher networks to compact student models.

In this review paper, we present a deployment-oriented systematic analysis concerning real-time deep learning–based object detection models for resource-constrained embedded systems. Different from existing surveys, which mainly emphasize detection accuracy or architectural design, this work jointly examines the detection performance with practical deployment constraints, including computational capacity, memory footprint, power consumption, thermal limitations, and hardware-specific optimization frameworks. By integrating model architecture, compression strategies, and target hardware platforms within a unified evaluation perspective, this review underlines the trade-offs necessary toward real-time inference on embedded and edge devices.

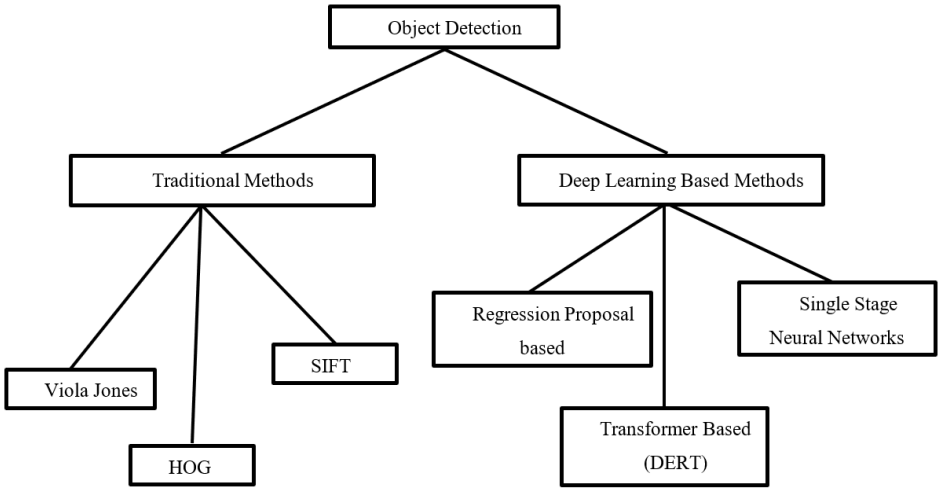

Figure 1. Types of Object Detection Methods

2. Literature Review

The early days of object detection were characterized by hand-designed feature extractors and shallow learning models. Basic techniques such as Scale-Invariant Feature Transform (SIFT) were developed to add robustness to scale and rotation transformations. Histogram of Oriented Gradients (HOG) was established as a norm for pedestrian detection [15] for its simplicity and effectiveness. The Viola-Jones face detector, which used Haar-like features [16] and an AdaBoost classifier [17], was one of the earliest real-time face detectors.

Although these successes were substantial, traditional approaches were plagued by one key limitation: the semantic gap — a lack of ability to bridge the gap between low-level pixel descriptors and high-level semantic understanding. Even in difficult conditions where there was large variability in object appearances, lighting, and presence of occlusions, the performance of such methods plateaued. It wasn't until the arrival of Convolutional Neural Networks (CNNs) that there was a demonstration of the ability to learn such hierarchical feature representations automatically, rendering manual feature design unnecessary. Such capabilities, together with the availability of large amounts of annotated data and GPU computing, brought about the revolution in object detection through deep learning [18].

2.1 Deep Learning-Based Object Detection Architectures

Due to the success of CNNs in image classification, sophisticated object detection frameworks began to emerge. Modern deep learning-based object detectors can largely be divided into two families: the two-stage and single-stage detectors.

a. **Two-Stage Detectors:** Two-Stage Detectors focus on achieving high accuracy by dividing the object detection process into two stages, namely the generation of region proposals and then classifying and refining the boundaries of those proposals. The R-CNN series of detectors are characterized by such a two-stage process. R-CNN [19] initiated the concept of using CNNs for region-proposal-based object detection, but it was computationally intensive because it processed each region separately. Fast R-CNN [20] optimized the concept by using shared convolutional features for each of the proposed regions, thereby reducing processing overhead. Faster R-CNN [21] further enhanced the concept by using Region Proposal Networks (RPNs) for end-to-end processing of the entire detection system, thereby making it faster. Even then, it is computationally intensive because of the multi-stage processing of the system.

b. **One-Stage Detectors:** Single-Stage Detectors directly address object detection tasks through a regression problem, bypassing the need for a region proposal step. YOLO (You Only Look Once) [22] introduced this paradigm where it directly predicts bounding boxes and class scores in a single pass. Later, SSD (Single Shot MultiBox Detector) [23] improved upon it with multiple scale feature maps to enhance detection performance. By bypassing an unnecessary computational step of region proposal, these detectors are considerably faster than two-stage detectors. But even though it is more efficient than their two-stage counterparts, traditional YOLO and SSD detectors are still resource-intensive technologies that cannot be directly implemented on very resource constrained embedded systems.

2.2 Lightweight Architectures for Embedded Systems

Though models such as Faster RCNN, YOLO, and SSD led to the introduction of backbone models in object detection with the help of deep learning techniques, they are still computationally expensive and difficult to implement on embedded systems. The need to develop efficient backbone models on embedded devices led to significant architectural changes and innovations. Some of the main architectures are:

a. **MobileNet:** MobileNet [24] proposed depthwise separable convolutions, which can be viewed as splitting the standard convolution into two operations: depthwise convolution, applying one filter for each input channel, followed by pointwise convolution (a 1×1 convolution for combining channels). Depthwise separable convolutions lowered the computational complexity and parameter count by an 8-9× factor compared to standard convolutions with acceptable accuracy. MobileNet V2 [11] was an extension of this approach with inverted residual bypass connections in linear bottlenecks. MobileNet V3 [25] was optimized through NAS and incorporated Squeeze-and-Excitation attention.

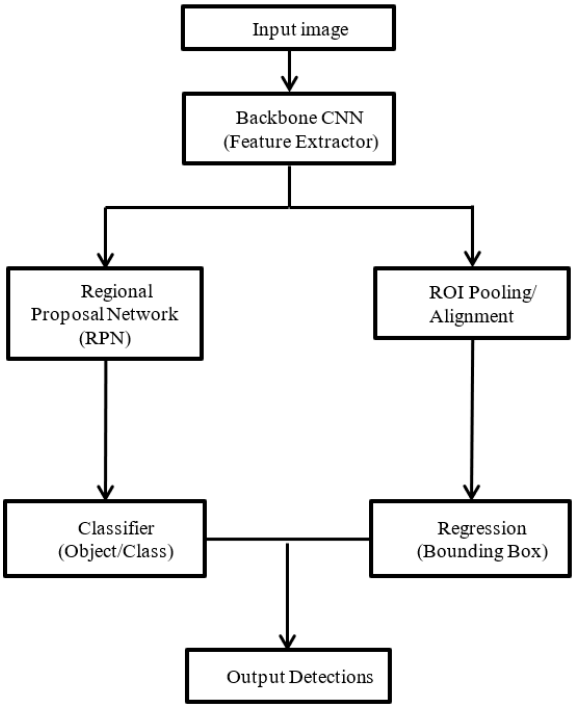

Figure 2. Two Stage Detector

b. **ShuffleNet:** ShuffleNet [26] proposed channel shuffle operations to facilitate efficient information exchange among channels in each group in group convolutions. While group convolutions have low computation costs because the convolution process is done independently among groups of input channels, information cannot be exchanged among groups. The problem with group convolutions is alleviated with shuffle operations, which involve the random shuffle of channels among groups to enable inter-group information exchange with low computational costs. Along with depth-wise separable convolutions and reshuffle connections, ShuffleNet has competitive accuracy with much lower FLOPS compared to MobileNet. ShuffleNet V2 [27] improved their network design incorporating efficient network design heuristics, with low memory access costs and high parallelization levels, which has faster inference speed on mobile platforms with competitive accuracy performance.

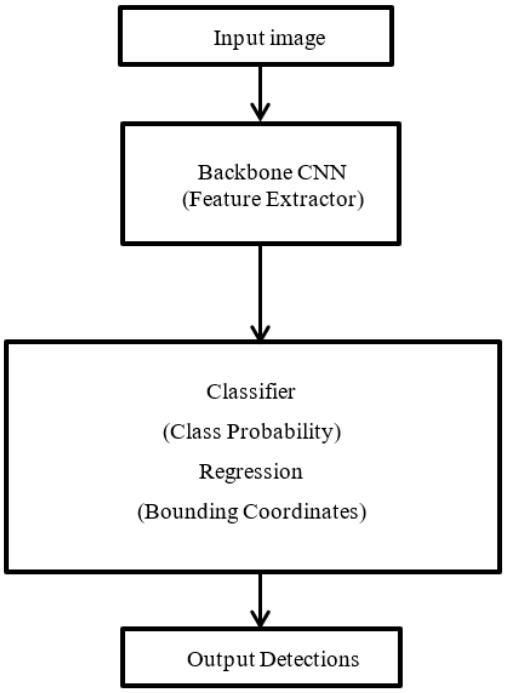

Figure 3. One Stage Detector

c. **EfficientNet/EfficientDet:** EfficientNet [28] proposed compound scaling, where the network depth, width, and resolution are scaled using a compound scaling coefficient for a better balance between accuracy and efficiency. Contrary to other models, EfficientNet balances the three factors through the application of Neural Architecture Search. This gives a set of EfficientNet models (B0-B7) with better accuracy/FLOP ratio than past architectures. EfficientDet [29] further adapts this method for object detection tasks by proposing a weighted bidirectional feature pyramid network (BiFPN) for handling multi-scale features more efficiently. EfficientNet-Lite alters EfficientNet by eliminating operations that are not supported well on edge TPUs like Swish Activation.

d. **NanoDet:** NanoDet [30] is a highly efficient object detection model designed for mobile and edge devices. It supports real-time inference (>97 FPS) on ARM CPUs. The model uses an anchor-free detection head with generalized focal loss. Both operations (generating anchors and performing Non-Maximum Suppression (NMS) are avoided during model execution. The model architecture uses ShuffleNetV2 or GhostNet as its backbone feature extractor and includes light feature pyramid networks. The model size is approximately 0.6-1.8MB with very low computing complexities. Thus, NanoDet is one of the most efficient object detection models for resource-constrained devices

3. Model Compression Techniques for edge deployment

Model compression algorithms allow for efficient execution of deep learning object detectors on edge devices, which have limited resources. These compression algorithms fill the gap that exists between the computational complexity of modern models and limited resources in edge devices. The compression models include knowledge distillation, pruning, and quantization, which significantly reduce model size and inference time and have negligible effect on mean Average Precision (mAP) scores.

3.1 Quantization

It reduces the model size and computational cost by representing weights and activations in lower-bit precision, as stated by Jacob et al. Standard deep learning models are based on 32-bit floating-point representation, but quantization transforms them into lower-precision formats such as 16-bit floating-point or 8-bit integers. INT8 quantization offers approximately 4× memory reductions and notably speeds up the inference, especially on hardware platforms with optimized integer arithmetic units such as NVIDIA Jetson devices or Google Coral Edge TPUs.

There are mainly two ways of quantization: PTQ, which applies quantization to already trained models by converting the weights and activations to lower precision after training is complete. This approach is easy and does not require any retraining but might face accuracy degradation for some models. QAT, on the other hand, simulates quantization effects in the process of training in such a way that the model learns to be robust against reduced numerical precision [13]. Normally, QAT results in better accuracy compared with PTQ; the accuracy loss is often contained at 1-2% in mAP, although it does take more time and computational resources.

One of the most effective combinations for object detection on embedded systems was INT8 quantization with optimized inference frameworks such as TensorFlow Lite or TensorRT, achieving up to 2-4x speedup for FP32 inference while almost maintaining the detection accuracy of the original model.

3.2 Pruning

Pruning can be used to reduce the complexity of a model by eliminating the less important parameters from the trained neural networks [12]. Pruning was proposed based on the insight for deep neural networks, where a few of the parameters make a small contribution to the result. Pruning can be done using two techniques, depending on the level of parameters removed.

Unstructured pruning comprises the removal of individual weights based on their criteria of magnitude or importance. This leads to sparse weight matrices that are sparsely structured. Although in unstructured pruning, prunes have a large number of parameters, up to 70-90%, it does not support traditional embedded processing. This is because sparse matrices are processed using sparse computing libraries.

The method of pruning relies on the elimination of whole structural entities like neurons, channels, and layers. This approach truncates the computational graph directly [31]. The process of pruning whole filters and/or channels results in models with high density. As such, they promote acceleration of inference with immediate results on general hardware without using special hardware support for computations related to sparseness. The pruning method is most appropriate to be used on Raspberry Pi and ARM processors.

The conventional pruning process involves three steps: (1) training the network to completion, (2) thresholding parameters based on a specific criterion (e.g., magnitude-based selection), and (3) fine-tuning the network to compensate for lost accuracy [12]. Iterative methods that alternate between pruning steps and fine-tuning tend to strike a better trade-off between accuracy and compression ratio than single-shot methods [32] discuss the details for object detection models, which use structured pruning to attain a reduction in FLOPs between 40-60% while incurring less than 2% mAP loss accompanied with fine-tuning.

3.3 Knowledge Distillation

Knowledge Distillation [33] is a form of model training where there is a smaller ‘Student’ model that imitates another ‘Teacher’ model. The twist here is that while normal model learning involves finding answers such as “Dog” or “Cat” for an image, with KD, there is learning from ‘soft targets’-probability distributions of how sure the ‘Teacher’ is about each possible answer (like 85% dog, 10% wolf, 5%

fox). This helps to better comprehend the differences and relationships between classes of objects that already exist and understood with expertise by the 'Teacher' model. In object detection learning too, things become very intriguing. In this case, "a small Student model such as 'Tiny-YOLO' can learn to match not only predictions but 'internal feature maps' and 'bounding box changes' of its 'Teacher model' such as 'full YOLOv10"'. By imitating how its 'Teacher' model 'sees' and interprets deep-level image inputs to some degree, "the 'Student' model performs significantly better than its own 'original' model on its 'original' dataset, yet still remains "small enough to run on resource-constrained devices".

4 Deployment on Embedded Systems and Edge Devices

Embedding deep learning models on embedded systems as well as edge devices also poses several challenges compared to inference on cloud platforms. Since cloud platforms support a vast amount of computational power in their infrastructures, as found in the case of data centres, embedding models on systems like the NVIDIA Jetson series, Raspberry Pi, Smartphone processors, possess stringent constraints in terms of memory capacity (typically 2-8GB RAM) as well as power consumption constraints (5-15W) in comparison to GPUs in the server category. This poses a challenge in the trade-off between achieving a runtime inference period in object detection models on such platforms.

This section analyses the essential platforms associated with embedding object detection models.

4.1 Hardware Platforms and Optimization Frameworks

a. **NVIDIA Jetson Platform:** The NVIDIA Jetson series offers support for the hardware-based inference of deep learning using the onboard CUDA-compatible GPU, which qualifies the platforms for edge computing that entail AI operations requiring higher computational rates. The solution of choice for the NVIDIA Jetson series is the TensorRT, an inference optimization platform that, as informed by NVIDIA in the year 2020, enhances inference performance using layer fusion, auto-tune, and mixed precision inference. TensorRT offers fast inference processes using the INT8 inference of the framework. The performance scale varies upon the number of GPU cores as well as the available RAM in the specific module. The power draw could be expected at 1.5A in the Nano family, 0.5A in TX1, 1.5A in TX2, 1.5A in TX2 Plus, 0.5A in AGX Xavier NX, and 0.5A in AGX Xaviator. From a performance standpoint, the Nano has a rather low performance scale at 20-30 FPS for light models such as YOLOv8-Nano or MobileNet-SSD optimized using TensorRT. For a rather higher-performing module such as AGX Xavier NX, however, the performance scale can be up to 60+ FPS applying YOLOv5-Small. Yet for computationally intensive models such as the VGG-based SSD, a lower performance scale is utilized.

b. **Raspberry Pi Platform:** Raspberry Pi (Versions 4 & 5) is a completely different architecture, using only ARM Cortex-A processors, without supporting hard neural nets accelerators. Such a limitation requires highly optimized ML inferencing engines, running natively on these CPUs, like TensorFlow Lite w/ XNNPACK delegation [34] or ONNX Runtime [35]. Raspberry Pi 4 with its quad-core Cortex-A72 processor sustains an average rate of 5-10 FPS in the execution of INT8-quanternized MobileNet-SSD models on TensorFlow Lite [24]. The Raspberry Pi 5 with its enhanced Cortex-A76 processor supports 2-3 times faster processing. Models with full architecture (ResNet50, VGG-16) remain unsuitable because of their complexity in terms of computation and memory bandwidths. In practice on Raspberry Pi, only those models that have light architectures – MobileNetV2, ShuffleNetV2, NanoDet models involving INT8 quantization [11][30] would be used.

4.2 Technical Deployment Challenges

Embedded object detection systems must address several interconnected technical constraints that directly impact inference performance and system reliability:

a. **Quantization Trade-offs:** The transition of switching over from 32-bit Floating-point to 8-bit Integer parameters results in a reduction of memory capacity by a significant extent of 4× with fast integer calculations [13]. The loss of accuracy typically results in a loss of accuracy of a small extent of 1-3%, depending on network architectures and techniques adopted along with parameter quantization [36]. Post-training techniques can effectively be adopted with a likely significant loss in accuracy, with the consideration that accuracy in terms of mAP will likely be sacrificed for longer training complexities.

b. **Thermal Management and Throttling:** Embedded systems employ passive cooling technologies characterized by tight thermal requirements in the form of heat sinks that ensure the system's hardware components are protected against overheating damage. The continuous execution of resource-intensive models translates to a high level of thermal production, hence the need for clock frequency scaling in order to regulate system temperatures (NVIDIA 2019). Dynamic frequency scaling equates to the variation in the latency associated with the model's execution, as evidenced by the stuttering observed when the model processes real-time video content. Applications involving predictable processing throughput relate to thermal necessities through the regulation of workloads or pulsed execution and even the use of active cooling systems in the case of self-driving and industrial inspection applications.

c. **Memory Constraints:** The large backbone models tend to be tens of millions in terms of the number of parameters that surpass the embedded systems RAM capacity, which lie between 2-8GB in most cases. The 4GB memory in the Jetson Nano makes model capacity limited, while in the 8GB memory, issues can be encountered in models that employ an ensemble approach or an ensemble system. This was proposed by [37]. If the situation reaches memory exhaustion, the system will become unstable, inference will fail, and the system will just crash. These limitations make the need for smaller models that employ models such as MobileNet, ShuffleNet, and EfficientNet-Lite necessary to achieve the lowest memory usage possible with the capability retained in the extraction feature. This notion was supported by Howard et al. 2017, Zhang et al. 2018, and Tan & Le 2019 among others.

d. **Power Consumption and Battery Life:** Battery-constrained edge devices such as drones, robots, IoT sensors etc function on a strict energy budget. Continuous inference tends to deplete battery power quickly; for example, a Jetson Nano board performing inference on YOLOv5 at 30 FPS consumes ~10W (NVIDIA 2019). However, power-efficient execution involves careful considerations on adaptive frame rate adjustment depending on the complexity level of the environment, activation/deactivation depending on the tradeoff between detection performance and power saving strategies.

5 Performance Evaluation and Comparative Analysis

Having laid down the theoretical basis for model compression algorithms and limitations imposed by embedded platforms, this section introduces an exhaustive analysis of state-of-the-art models for detecting objects applied in resource-constrained platforms. We will be comparing popular models for resource-constrained platforms based on important performance factors applied on popular embedded platforms. Table 1 presents a comparative analysis of embedded platforms suitable for real-time object detection, highlighting the relationship between hardware capabilities, achievable inference speeds, and power efficiency.

It should be noted that the reported performance figures represent indicative ranges obtained from existing studies and may vary depending on implementation details, model optimization strategies, and hardware configurations.

Table 1: A Comparative Overview of Embedded Platforms Commonly Used for Real-Time Object Detection

Platform	Processor	Hardware	Models	Optimization Approach	Range	Budget	References
GPU-Accelerated Edge Devices	CUDA-enabled GPU	NVIDIA Jetson Nano, Jetson Xavier NX	YOLOv4-tiny, SSD-ResNet, MobileNet-SSD	TensorRT (FP16/INT8), layer fusion	5–60 FPS	5–15W	[38]
CPU-Based Embedded Systems	ARM CPU	Raspberry Pi 4/5	SSD-MobileNet (TFlite), lightweight yolo variants	TFLite (INT8), ARM NN, XNNPACK	2–12 FPS	3–12W	[39]
ASIC-Based Accelerators	Dedicated AI Accelerator (Edge TPU)	Google Coral Dev Board, USB Accelerator	SSD-MobileNetV2 (INT8), EfficientDet-Lite	Edge TPU Compiler (INT8)	70–120 FPS	<2.5W	[40]
Mobile SoC Platforms	Heterogeneous SoC (CPU + NPU/DSP)	Modern Smartphones	YOLO-Lite, MobileNetV2/V3	CoreML, NNAPI, SNPE	20–45 FPS	2–5W	[9]

Table 2: Comparison of Object Detection Benchmark Datasets

Dataset	Images	Object Categories	Average Instances/Image	Primary Metric	Typical Use for Embedded Systems
MS COCO	330K	80	7.7	mAP@0.5:0.95	Primary benchmark, challenging scenes
PASCAL VOC 2007/2012	11.5K	20	2.4	mAP@0.5	Lightweight model evaluation, faster benchmarking
ImageNet Detection	450K	200	Variable	mAP@0.5	Backbone pre-training, category diversity

Dataset	Images	Object Categories	Average Instances/Image	Primary Metric	Typical Use for Embedded Systems
Open Images V6	9M+	600	Variable	mAP@0.5	Large-scale pre-training, generalization testing

Table 3: Performance Characteristics of Lightweight Object Detection Models

Model	Backbone	Dataset	mAP	Inference Speed (FPS)	Model Size	Embedded Platform	Reference
SSD-MobileNetV2	MobileNetV2	PASCAL VOC	~72%	20–30	~14 MB	Jetson Nano	[9]
YOLOv4-Tiny	CSPDarknet-Tiny	MS COCO	~40%	30–45	~23 MB	Jetson Nano	[41]
NanoDet	ShuffleNetV2	MS COCO	~33%	60–90	~1 MB	ARM CPU	[30]
EfficientDet-Lite0	EfficientNet-Lite	MS COCO	~30%	25–40	~4 MB	Edge TPU	[29]
YOLOv5-Nano	CSPDarknet-Nano	MS COCO	~28%	40–60	~1.9 MB	Jetson Xavier NX	[42]

* Performance ranges are indicative values summarized from the literature and may vary depending on model configuration, quantization, input resolution, and hardware optimization.

5.1 Platform-Level Performance Evaluation

As shown in Table 1, object detection tasks experience varying performance capabilities running on different embedded hardware systems. For example, systems that have graphical processing units, like NVIDIA's Jetson Nano and Xavier NX, take advantage of parallel computing capabilities, hence offering real-time inference performance using lightweight models and TensorRT. However, they consume relatively high power.

On the other hand, CPU platforms like Raspberry Pi-class platforms tend to have limited speeds of the inference process due to resource limitations. However, with an optimized inference solution and a quantized model, low-complexity detection tasks can be accomplished on these platforms.

ASIC accelerators, such as Google Coral Edge TPU, prove their supremacy in performance per watt. Fully quantized models, when run on these accelerators, support real-time inference with low power, making it ideal for battery and always-on edge use cases.

5.2 Model-Level Comparative Analysis

Table 3 shows the comparative analysis of lightweight object detection architectures developed for the mobile platform.

SSD-MobileNet architectures strike the right balance between detection performance and efficiency and are chosen as the baseline architecture.

Lightweight variants of YOLO place high importance on inference speed and miniaturized models, even where this means sacrificing detection performance, especially with regard to smaller objects.

The EfficientDet-Lite series of models uses compound scaling and optimized feature fusion to increase model performance with less challenging computational requirements and is applicable to both GPU computing and ASIC solutions.

5.3 Key Observations

The analysis shows that there is not a model and a platform that suits all the deployment scenarios the best. This varies according to the applications and their requirements in terms of accuracy, latency, availability of power sources, and availability of hardware support.

These results demonstrate the importance of evaluation with respect to deployment and hardware considerations in model selection for real-time object detection in embedded systems.

6 Future Trends in Embedded Object Detection

As research progresses within the domain of Edge AI, an evolving trend from generic model compression algorithms to Hardware Aware Neural Architecture Search, commonly referred to as NAS, has been witnessed. The use of Hardware Aware NAS has been found to be an automated method through which the architecture of a neural network can be molded according to the computational capabilities of the hardware platform on which it needs to be implemented. It has been recognized that not all processors are created equal. CPUs are very much apt for tasks that require sequential computations. GPUs are designed in a manner that they are aptly suited to support parallel computations at a larger scale, whereas NPUs are apt for computations that require fixed-point arithmetic operations like matrix multiplications and convolution.

Examples of such hardware-aware designs include more modern object detection designs. The designs of YOLOv10 and the newly formed YOLOv11 are all constructed from the ground up, with the NPU constraints given careful consideration, as opposed to the previous approach that looked at them as an afterthought. From the designs, hardware-aware designs appear, like the usage of depth-wise separable convolution, channel operations, and layering that are low in memory bandwidth, suited to how the NPUs process the computation workloads. Such designs form part of the effort to develop models with the so-called “native efficiency,” which runs fast since they are neither compressed nor pruned but rather hardware architecture suited to the computation performed by the hardware.

The answer to the perpetual gap that always exists between the efficiency of the model, as measured in FLOPs, and the efficiency as measured in latency and throughput on the edge device, lies in this approach, which provides the answer to the development of a novel set of edge applications that were not feasible on edge devices until now.

6.1 Transformer-Based Detectors on the Edge

Though Convolutional Neural Networks (CNNs), as embodied in techniques like YOLO and SSD, have been leading the way in embedded object detection technology due to their efficiency and ability to stay localized, Vision Transformers (ViTs) are on the cusp of being viable alternatives in edge learning too. Unlike conventional CNN techniques used in analysing images, the transformer model involves a radically new paradigm, wherein images are analysed patch by patch.

a. **Hybrid Models:** It was understood that purely Transformer models are currently too resource-intensive for any but the most advanced embeddable systems, and hybrid models that better leverage the complementary strengths of both approaches were proposed as an optimal solution. “MobileViT models combine the best attributes of CNNs’ ability to focus on

local patterns in an image, like edges and texture," said one expert, "with the Net models' ability to analyze the bigger picture of the image and understand the relationship between distant objects." This approach would allow the system to analyze an image at the level of detail required for the edge system, and then analyse the relationship of the various objects within the image. "This makes the 'Net models much more powerful," the expert noted, "because they are able to analyse the bigger picture. Hence, the system could analyze an image at the level of detail required for the edge system, and then analyse the relationship of the various objects within the image.

b. **Memory and Computational Complexity:** The main hindrance in the adoption of the Transformer in edge devices is the quadratic computational complexity with respect to the size of the input. The standard self-Attention concept needs the computing of the relation between all pairs of patches in the image. Therefore, the memory complexity is $O(n^2)$, where n is the number of patches. For a standard high-resolution image partitioned into hundreds of patches, this easily goes beyond the 2-4GB memory requirements of edge devices. The current state of the art mitigates this limit by using "Linear Attention" variants—a model approximation that has reduced the complexity from $O(n^2)$ to O(n), without computing the relation between all pairs of inputs [9]. This optimized form of the Attention idea makes it possible to apply the Transformer-based detector within the memory constraints of edge devices and at the same time maintain the capability to reason about the full model standing accuracy concerning complex scenes that include high levels of occlusions, overlapping instances, or complicated spatial arrangements, which are not possible within the receptive boundaries of conventional YOLO models.

6.2 Real-Time Adaptive Inference

One of the new paradigms in the area of embedded AI is Dynamic Inference, where the ability to adaptively change the computation complexity of a deployed system based on runtime conditions, such as the power budget, processing workload, or difficulty of the data being processed, is utilized. This marks a shift from the current static deployment methodology, where the computation cost remains constant, irrespective of whether it is an empty scene or a busy one.

a. **Adaptive Complexity Mechanisms:** Firstly, dynamic inference systems leverage diverse methods to control the complexity of the computations. One of the simplest methods relies on early-exit models whose architecture involves the design of the system with pre-defined points for decision-making, thereby enabling the system to terminate the inference process for simple inputs without requiring the inputs to pass through all the layers in the system. More complex systems rely on multi-backbone switching for the inference process based on the complexity of the scene [44].For instance, in an embedded security system operating on an embedded battery supply on an Raspberry Pi microcomputer in an IoT setting for securing remote surveillance systems, the system would rely on the following: when the system detects inactivity within the scene based on the cost-effective approach of performing image differencing or simple motion detection, the system would rely on the low-power backbone MobileNet-SSD, possibly dissipating only 2-3 watts of power, thereby suitable for continuous battery life. On the emergence of significant motion within the scene based on the triggers of alerts such as the entrance of an individual within the scene of monitoring, the system would adapt and dynamically switch over to the more complex backbone architecture YOLOv8-medium for accurate identification and tracking of the targets within the scene, performed within the system in the range of milliseconds without any gaps in the computations for the inference task.

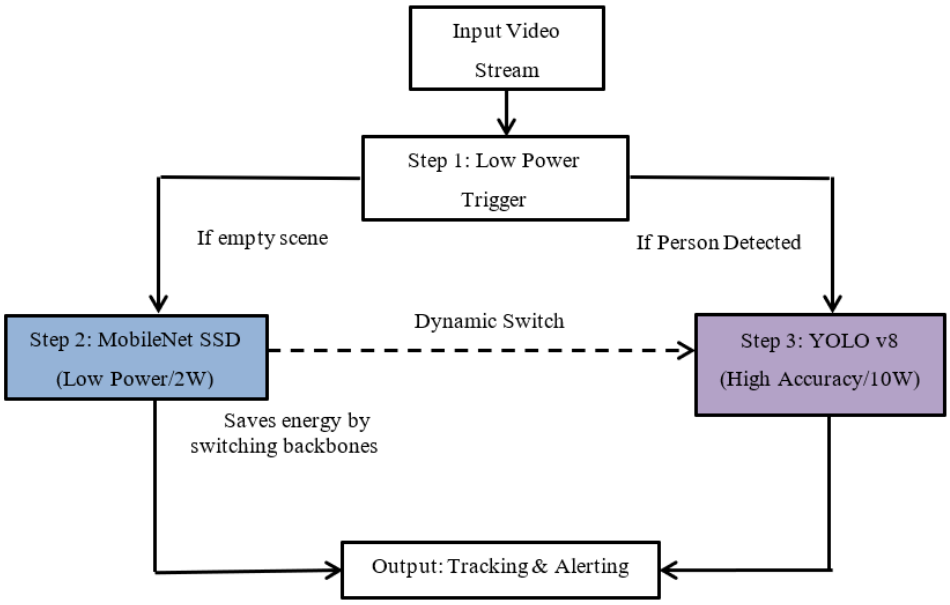

Figure 4. One Stage Detector

b. **Power and Performance Optimization:** The value of dynamic inference goes well beyond power reduction gain. In environments with dynamic changes in scene complexity, such as between driving scenarios such as sparse highways and busy city intersections in autonomous vehicles, dynamic models may deploy computational power resourcefully, performing well in low-complexity regions and still preserving strong accuracy when needed most. It provides a better power- performance characteristic relative to running always with a power-hungry model or with only a light one with accuracy constrained by operating conditions. Difficulty-aware routing, in which models infer input difficulty based on early-layer features, has also emerged in recent studies of this work, which begins to develop them to decide when to burn more power.

The following Table 4 evaluates how these trends are expected to impact on the performance of embedded detectors over the next few years:

Table 4: Expected Impact of The Performance of Embedded Detectors Over the Next Few years

Trend	Expected Impact	Primary Hardware	Maturity Level
NPU Optimization	5-10x Speedup	Jetson Orin / RK3588	High
Edge Transformers	Higher mAP in complex scenes	Jetson Xavier / Orin	Moderate
Adaptive Inference	40% Power Reduction	Raspberry Pi / ESP32-S3	Emerging

7. Conclusion

In this literature analysis, we have reviewed the development of single-stage object detection models, including YOLO and SSD models, and the difficulties involved in migrating them from the powerful environment of High-Performance Computing workstations to the more restricted environment of edge devices. By means of an analysis of development and optimization tests, we have identified how literature has coped with the needs of edge devices for AI functionality.

Analysis highlights a shift in paradigms in what a neural model would prioritize in its exploration, from a focus that was established through models such as AlexNet and VGG as a proof of concept that it was possible to extract features using a deep model in a computer vision setting, to a new set that

has been established as a consequence of edge-computing requirements and has come to prioritize models that are hardware-friendly or models that are able to dynamically change their resource usage based on operating conditions in a way that optimizes inference performance.

From our comparative assessment of hardware platforms, we know that a "one-size-fits-all solution" approach will not work. Embedded object detection requires that model complexity be selected and tuned to accommodate hardware capabilities:

For High-Performance Edge AI Use Cases: For those use cases requiring a combination of the ability to customize the model along with high performance, the NVIDIA Jetson line of products (Nano, Xavier, and Orin) is still the best option. Based on TensorRT optimization along with the use of specialized GPU cores capable of executing CUDA code, the current software environment has the most seamless integration with relatively complex YOLO models with sub-30ms latency performance.

For Low-Cost and CPU-Constrained Deployment: Raspberry Pi models 4 and 5 provide a cost-effective option when coupled with lean backbone models such as MobileNet or the use of dedicated hardware acceleration like Google Coral Edge TPU. However, as evidenced by peer review research, simple CPU computation without these considerations remains inefficient to provide real-time performance at or above FPS=30 for high-resolution detection class models with numbers above 5-10 million. Such models are instead most appropriate for non-real-time applications with lower resolutions and the possibility to use dedicated AI acceleration hardware.

8. Final Recommendation

As for people involved in the research and development of object detection technology and its adoption on embedded systems, the key to success revolves around the concept of achieving the synergy between the computational nature of the model and the intrinsic acceleration abilities of the targeted hardware. Through understanding and analysis, it has been revealed that, though valuable, post-hoc optimization tools are not sufficient for mitigating mismatches between the designed and executed architectures on the targeted hardware.

Looking ahead, new trends in NPU-friendly processor architectures and adaptive inference schemes indicate that the traditional divide between server-side detection quality and edge-side speed will continue to close. Models designed with these patterns in mind, and including techniques such as depth wise separable convolutions, efficient use of attention, and quantizable layer configurations, are already beginning to prove that embedded devices are capable of reaching data center levels of detection quality while staying within very tight power and thermal constraints.
We make the following specific recommendations for furthering this field:

a. **Prioritize Hardware-Conscious Neural Architecture Search (NAS):** Future work should focus on NAS techniques that are hardware-aware and use real-world hardware metrics like latency, energy consumption, and heat generation along with accuracy as first-class objectives during the model learning process Optimizing model deployment on hardware should not be relegated to the end of the model learning process. Instead, model learning should focus on learning architectures that are optimized for hardware. This will deliver models that are natively optimized and not merely optimized and compressed.

b. **Establish Standardized Benchmarking Protocols**: The community can greatly benefit from the adoption of standardized benchmarking tools that provide not only the measurement of mAP but also other deployment-oriented metrics, including FPS, power consumption, and thermal characteristics, on a set of platforms catered to the needs of the embedded community.

c. **Research Energy-Aware Training:** In addition to latency, the next area of research could include the design of training loss functions that take into consideration the energy usage—

this would be especially important for battery-supplied systems in areas like remote health monitoring, agricultural systems, and mobile robotics scenarios. Energy-efficient models could potentially trade off differently on architecture design compared to latency models.

d. **Extend Multi-Task and Adaptive Frameworks:** Research efforts in dynamic inference frameworks with adjustable computational complexity according to the difficulty of the image, power availability, and application necessity appear to be very promising for making optimal use of edge capabilities, even when these capabilities may be constrained and limited. Additionally, such frameworks could be vital for applications of next generation, which need to function constantly under changing conditions.

Moreover, as a result of enhanced image pre-processing, increased architecture support, and enhanced image analysis facilities, Although the processors in embedding systems are also becoming highly sophisticated in their acceleration capabilities for AI and the developments are progressing towards making even more efficient frameworks for the detection of objects, we are of the view that edge computing is set to venture into applications that are currently hampered in the domain of object detection. The fusion of these developments brings the focus to the fact the embedding systems based on AI are no longer a trade-off between the two solutions based in the cloud but rather a new paradigm in which embedded AI applications offer special strengths in terms of privacy, latency, reliability, and cost.

Conflict of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

The research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data Fabrication/Falsification Statement

The author(s) declare that no data have been fabricated, falsified, or manipulated in this study.

Participant Consent

The authors confirm that Informed consent was obtained from all participants, and confidentiality was duly maintained.

References

[1]. P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pp. 1627-1645, 2010.

[2]. W. Pitts and W. S. McCulloch, "How we know universals: The perception of auditory and visual forms," The Bulletin of Mathematical Biophysics, vol. 9, no. 3, pp. 127-147, 1947.

[3]. G. E. Hinton, D. E. Rumelhart, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, pp. 533-536, 1986.

[4]. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 2001.
[5]. D. G. Lowe, "Distinctive image features from scale-invariant key points," International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, 2004.
[6]. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, pp. 886-893, 2005.
[7]. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.
[8]. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Advances in Neural Information Processing Systems, vol. 25, 2012.
[9]. A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.
[10]. X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6848-6856, 2018.
[11]. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4510-4520, 2018.
[12]. S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," International Conference on Learning Representations (ICLR), 2016.
[13]. B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2704-2713, 2018.
[14]. G. Hi006Eton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv preprint arXiv:1503.02531, 2015.
[15]. P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 4, pp. 743-761, 2012
[16]. R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," in Proc. IEEE International Conference on Image Processing (ICIP), vol. 1, pp. I-900-I-903, 2002
[17]. Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting" Journal of Computer and System Sciences, vol. 55, no. 1, pp. 119-139, 1997.
[18]. G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, no. 7, pp. 1527-1554, 2006.
[19]. R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 580-587, 2014
[20]. R. Girshick, "Fast R-CNN," in Proc. IEEE International Conference on Computer Vision (ICCV), pp. 1440-1448, 2015.
[21]. S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in Proc. Advances in Neural Information Processing Systems (NIPS), pp. 91-99, 2015.
[22]. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788, 2016.
[23]. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in Proc. European Conference on Computer Vision (ECCV), pp. 21-37, 2016.

[24]. 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.
[25]. A. Howard, M. Sandler, G. Chu, L. C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," in Proc. IEEE/CVF International Conference on Computer Vision (ICCV), pp. 1314-1324, Oct. 2019.
[26]. Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6848–6856.
[27]. Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. Proceedings of the European Conference on Computer Vision (ECCV), 116–131
[28]. Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Proceedings of the 36th International Conference on Machine Learning (ICML), 6105–6114.
[29]. Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and Efficient Object Detection. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 10781–10790.
[30]. Lyu, Z. (2021). NanoDet: Real-time anchor-free object detection for mobile devices. GitHub. https://github.com/RangiLyu/nanodet
[31]. Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2017). Pruning Filters for Efficient ConvNets. Proceedings of the 5th International Conference on Learning Representations (ICLR).
[32]. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning Efficient Convolutional Networks through Network Slimming. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2736–2744.
[33]. Chen, G., Choi, W., Yu, X., Han, T., & Chandraker, M. (2017). Learning Efficient Object Detection Models with Knowledge Distillation. Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS), 742–751.
[34]. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J, & Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 265–283.
[35]. Microsoft. (2019). ONNX Runtime: Cross-platform, high performance ML inferencing and training accelerator. GitHub. https://github.com/microsoft/onnxruntime
[36]. Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342.
[37]. Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. Journal of Big Data, 6(1), 1-54.
[38]. Cantero, J., Usamentiaga, R., & Molleda, J. (2022). Evaluation of Edge Computing Devices for Object Detection Using Deep Learning. Sensors, 22(19), 7175.
[39]. Zagitov, A., Lavrenov, R., & Magid, E. (2024). Comparative analysis of neural network models performance on low-power devices for a real-time object detection task. Computer Optics, 48(2), 242–252.
[40]. Google. (2019). Coral Edge TPU: A hardware-accelerated platform for local AI. Google Developers. https://coral.ai/
[41]. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.
[42]. Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., Kwon, Y., Michael, K., ... & Fang, J. (2022). ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference. Zenodo. https://doi.org/10.5281/zenodo.6221181

[43]. Mishra, D. P., Rout, K. K., Mishra, S., & Salkuti, S. R. (2025). Various object detection algorithms and their comparison. Indonesian Journal of Electrical Engineering and Computer Science, 48(2).
[44]. Casado-García, Á., Domínguez, C., Heras, J., Mata, E., & Pascual, V. (2023). The Benefits of Close-Domain Fine-Tuning for Table Detection in Document Images. Multimedia Tools and Applications, 82(2), 2055–2075.