# Deep Learning Architectures for Real-Time Object Detection

[1] Ayesha Khan, [2] Muhammad Bilal, [3] Sanaullah Ahmed

[1] School of Electrical Engineering & Computer Science (SEECS), National University of Sciences & Technology (NUST), Islamabad, Pakistan
[2,] Department of Computer Science, University of Engineering & Technology (UET), Lahore,
[3,] Pakistan Department of Computer & Information Sciences, Pakistan Institute of Engineering & Applied Sciences (PIEAS), Islamabad, Pakistan.
***Corresponding Author:** ayesha.khan@seecs.nust.edu.pk*

***Abstract:*** *Real-time object detection—extracting category labels and bounding boxes at video frame rates—has become foundational for autonomous systems, industrial automation, smart cities, and augmented reality. This article surveys mainstream deep learning architectures with an emphasis on latency–accuracy trade-offs, deployment constraints on edge hardware, and training strategies that enable sub-30 ms inference. We compare one-stage (e.g., YOLO, SSD, EfficientDet) and two-stage (e.g., Faster R-CNN) families, as well as transformer-based detectors (e.g., DETR). We synthesize best practices for model compression (quantization, pruning), label assignment, loss design (IoU-aware), and data pipelines to maintain high mean average precision (mAP) while achieving high frames per second (FPS). We conclude with a practical checklist for selecting architectures under compute, energy, and memory budgets, and highlight open challenges such as long-tail detection, multi-sensor fusion, and trustworthy deployment.*

***Keywords: -*** *real-time detection, mAP vs FPS, YOLO, SSD, EfficientDet, DETR, model compression, edge deployment*

## Introduction

Object detection has evolved from sliding-window classifiers to end-to-end convolutional and transformer models capable of near-instantaneous inference. In safety-critical and interactive applications, "real-time" typically targets ≥30 FPS (≤33 ms per frame) on available hardware while sustaining robust accuracy across occlusions, small objects, and illumination changes. Achieving this balance requires system-level thinking: the detector backbone and head, input resolution, pre/post-processing, runtime libraries (TensorRT/ONNX Runtime), memory movement, and batching all influence end-to-end latency. This article analytically reviews the architecture landscape, training heuristics, and deployment toolchains that consistently deliver real-time performance across GPUs, NPUs, and embedded SoCs.

The architectural landscape of real-time object detection continues to evolve through innovations that balance **accuracy, speed, and computational efficiency**. Two-stage detectors such as Faster R-CNN and Mask R-CNN remain gold standards for high-precision applications due to their region proposal mechanisms that isolate likely object areas before classification. However, this multi-step process introduces latency, making them less suitable for time-critical environments like autonomous navigation. Conversely, one-stage detectors like **YOLO**, **SSD**, **RetinaNet**, and **EfficientDet** streamline the pipeline by merging region proposal and classification into a single network pass, drastically improving throughput while maintaining competitive accuracy. They utilize **anchor-based and anchor-free prediction heads**, **focal losses** to handle class imbalance, and **depthwise separable convolutions** for lightweight efficiency.

Modern architectures employ **feature pyramid networks (FPNs)** and their variants—PAFPN, BiFPN—to capture rich multi-scale contextual information, crucial for detecting small and overlapping objects. Innovations such as **decoupled heads** (separating classification from localization tasks) and **cross-stage partial connections (CSP)** improve gradient flow and model convergence, making detectors more efficient during training and inference. Transformer-based models like **DETR**, **Deformable DETR**, and **DINO-DETR** represent the latest shift in paradigm by replacing heuristic anchor mechanisms with **set-based bipartite matching**, ensuring one-to-one object assignments and enabling end-to-end differentiability. However, their training demands large datasets and extensive epochs, which recent distillation and hybrid CNN-transformer backbones (like YOLOv8-DETR hybrids) aim to mitigate.

the trade-off between accuracy and latency depends on the deployment scenario. **Edge devices** prioritize low-complexity, quantized CNNs, while **server-grade systems** can afford transformer or ensemble detectors for superior robustness. Future research is moving toward **unified detection frameworks** that dynamically adjust architecture depth and resolution at runtime, allowing real-time adaptation to changing hardware and environmental constraints.

Loss functions, label assignment strategies, and training mechanics play a pivotal role in achieving the fine balance between accuracy and efficiency in modern real-time object detectors. Traditional Intersection over Union (IoU) metrics measure the overlap between predicted and ground truth boxes, but newer variants such as **GIoU (Generalized IoU)**, **DIoU (Distance IoU)**, and **CIoU (Complete IoU)** have improved the precision of localization by incorporating distance penalties and aspect ratio alignment. These IoU-aware losses enhance gradient stability and lead to faster convergence, especially for small and densely packed objects.

Dynamic label assignment algorithms such as **SimOTA** (used in YOLOX) and **ATSS** (Adaptive Training Sample Selection) have replaced static anchor matching with data-driven mechanisms that automatically determine positive and negative samples based on object quality and confidence thresholds. This adaptive approach allows detectors to generalize better across object scales and categories without manually tuned anchors, improving small-object recall and robustness to occlusion.

In the training phase, **strong data augmentation** remains crucial for achieving high generalization. Techniques like **Mosaic augmentation**, **Copy-Paste**, **MixUp**, and **color/brightness jittering** artificially increase dataset diversity, making models resilient to environmental variations. Optimization strategies such as **cosine learning rate scheduling**, **Exponential Moving Average (EMA) weight updates**, and **batch normalization averaging** stabilize the learning process, particularly in large-scale training with mixed-precision arithmetic. **Multi-scale training**, where the input resolution changes dynamically per iteration, enables models to handle varying image sizes efficiently during inference without fine-tuning.

Furthermore, **knowledge distillation** has emerged as a practical tool to compress high-capacity teacher models (like YOLOv7 or DETR) into lightweight student architectures (like YOLOv8-n) while retaining high mean average precision (mAP). This transfer of "dark knowledge" allows smaller networks to approach the accuracy of larger ones without compromising real-time performance. Together, these advancements in loss design, label assignment, and training

mechanics form the backbone of high-performing, deployable object detectors capable of balancing precision, speed, and computational economy.

Model compression and acceleration are essential steps in transforming deep learning architectures from research prototypes into production-ready systems, especially for **real-time object detection**. The main goal is to retain high accuracy while reducing computational overhead, memory footprint, and latency. **Quantization**—converting floating-point weights and activations to lower precision formats such as **FP16** (half-precision) or **INT8** (integer precision)—is one of the most widely used techniques. When performed using **Quantization-Aware Training (QAT)** or **post-training calibration**, it can achieve up to 4× reductions in model size and inference time with minimal accuracy loss. Modern GPUs and NPUs are specifically designed to exploit these low-precision formats through **tensor cores** and dedicated arithmetic units, significantly accelerating convolution and matrix operations.

Another effective approach is **structured pruning**, where redundant channels, filters, or blocks are systematically removed from the network. This method differs from unstructured pruning by maintaining the model's structural integrity, ensuring compatibility with hardware parallelism. Pruned models, when fine-tuned, can preserve 95–98% of the original accuracy while reducing FLOPs (floating-point operations) and enabling faster inference. **Knowledge distillation** often complements pruning, allowing smaller student networks to learn from larger teacher models while maintaining comparable performance.

**Neural Architecture Search (NAS)** and **compound scaling** represent more advanced optimization paradigms. NAS automatically discovers optimal configurations by balancing width, depth, and resolution to meet hardware constraints. In models like **EfficientDet** and **EfficientNet**, compound scaling enables uniform scaling across all dimensions, yielding scalable performance tiers (D0–D7) suited for diverse devices—from mobile chips to data center GPUs.

Runtime optimizations further enhance deployment efficiency. Frameworks such as **TensorRT**, **ONNX Runtime**, **CoreML**, **TFLite**, and **OpenVINO** perform **graph-level fusion**, **operator kernel optimization**, and **precision lowering**, tailoring execution paths for target hardware. These tools eliminate redundant operations, fuse consecutive layers (e.g., convolution + batch normalization), and streamline memory access patterns.

Finally, attention to **post-processing** is critical. Steps like **batched Non-Maximum Suppression (NMS)**, **soft-NMS**, and **top-k candidate filtering** must be implemented on the GPU to avoid costly CPU-GPU context switching. Efficient post-processing ensures that high theoretical GPU throughput translates into real-world performance gains. Collectively, compression, pruning, quantization, and runtime optimization form the technological foundation of **energy-efficient, high-speed object detectors** capable of real-time inference on both embedded and cloud platforms.

**Edge Deployment: Systems Considerations**

Deploying real-time object detection models on edge devices presents unique challenges that stem from **limited computational resources, memory constraints, and power budgets**. Unlike cloud-based GPUs, embedded systems such as NVIDIA Jetson, Qualcomm Snapdragon, or Apple's Neural Engine operate under stringent **thermal and energy limitations**, requiring careful architectural and system-level optimization. Selecting the right model is a trade-off between

**accuracy, latency, and efficiency**. For embedded GPUs like the Jetson Nano, Xavier, or Orin, mixed-precision inference—especially **FP16 or INT8 quantization using TensorRT**—significantly improves throughput without noticeable degradation in detection accuracy. Reducing input image resolution from 640×640 to 512×512 or 448×448 further enhances performance by lowering computational cost, especially when detecting large objects or when frame rate is prioritized over small-object accuracy.

For dedicated NPUs (Neural Processing Units) such as **Qualcomm Hexagon DSPs, Apple Neural Engine, or Huawei Ascend**, **vendor-specific compilers** and **ops-set compliance** are critical. These platforms require static computation graphs and operator compatibility for optimal execution. **Graph optimization frameworks** like TVM, Glow, and MLIR are increasingly used to tailor models to the instruction sets of such processors.

Pipeline-level efficiency is equally important. Techniques such as **zero-copy I/O**, **asynchronous prefetching**, **pinned memory buffers**, and **multi-threaded pipelines** ensure minimal CPU-GPU data transfer latency. Implementing **concurrent CUDA streams** allows overlapping of data loading, inference, and post-processing, thereby increasing the *effective frames per second (FPS)*. Additionally, maintaining **thermal stability** through dynamic power management and model-switching (adaptive model scaling) prevents throttling during prolonged operation. Robustness in deployment further depends on **domain-specific fine-tuning**—using datasets reflecting real-world conditions such as night-time, fog, or rain. **Temporal smoothing** and **tracking-by-detection** methods (like SORT or DeepSORT) enhance frame-to-frame consistency, reducing flicker and false positives in live video feeds.

**Evaluation, Benchmarks, and Practical Selection**

Evaluating object detectors for real-world applications extends beyond accuracy metrics. The **mean Average Precision (mAP@0.5:0.95)**, popularized by COCO and VOC benchmarks, remains the gold standard for measuring detection precision and recall. However, **production-grade evaluations** incorporate additional metrics such as **end-to-end latency** (including preprocessing and Non-Maximum Suppression), **throughput (FPS)**, **power consumption**, **memory utilization**, and **reliability percentiles (P50/P90/P99)** to assess consistency under load.

**Model selection varies based on deployment conditions:**

- **Tight compute or power budgets** (e.g., mobile or embedded platforms) favor ultra-lightweight architectures like **YOLOv8-n/s**, **SSD-MobileNetV2**, and **EfficientDet-D0**, which provide acceptable accuracy with minimal latency.

- **Balanced accuracy and latency** on mid-range GPUs (e.g., RTX 2060, GTX 1660) are best achieved with **YOLOv5-s/m** and **YOLOX-s/m**, offering high mAP and stable FPS.

- **Maximum accuracy** in research or high-performance settings, where latency is secondary, is attainable with **YOLOv7**, **RetinaNet (optimized)**, or **DETR variants** enhanced with **knowledge distillation** and **graph-level fusions**.

In practice, the most efficient systems adopt **hybrid evaluation frameworks** that dynamically select models based on contextual constraints—switching between high-accuracy and high-speed modes depending on environmental load or battery levels. Future benchmarks are expected to

emphasize **energy per inference**, **sustainability**, and **fairness metrics** to evaluate AI models not just by accuracy, but also by efficiency and ethical deployment readiness.

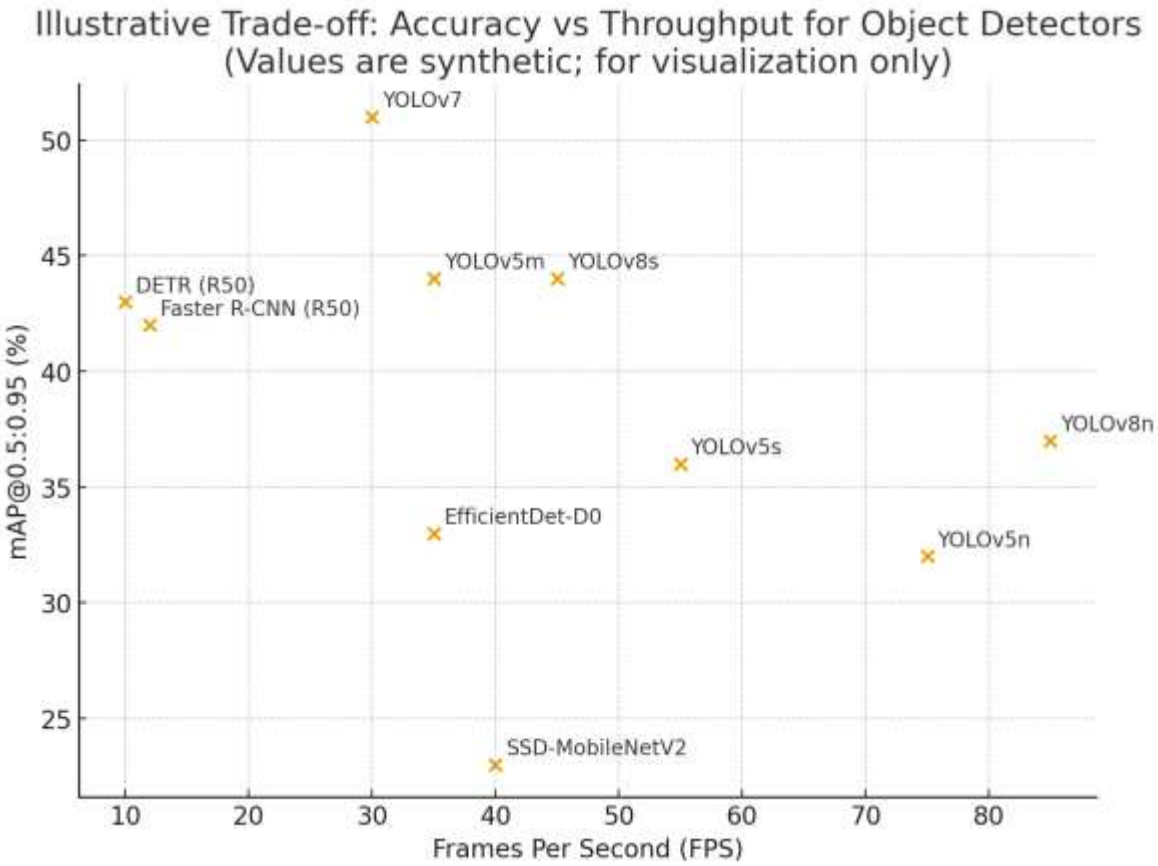
Edge deployment in real-time object detection is one of the most challenging yet impactful stages of bringing deep learning from the lab to practical applications. It requires **balancing performance, energy efficiency, and environmental adaptability** under strict hardware limitations. Edge devices such as **NVIDIA Jetson, Google Coral, Qualcomm Hexagon DSPs, and Apple Neural Engines** operate with limited memory, constrained power budgets, and thermal thresholds that often lead to throttling if unmanaged. Consequently, architectural selection and optimization must align with these physical constraints to ensure continuous, low-latency inference in real-world scenarios.

For embedded GPUs like the **NVIDIA Jetson Nano, Xavier, and Orin**, mixed-precision inference using **FP16 or INT8 quantization via TensorRT** provides the most practical balance between computational speed and detection accuracy. These quantized models significantly reduce memory bandwidth and computational overhead, allowing them to process high-resolution streams efficiently. Additionally, lowering input resolutions from **640×640 to 512×512 or even 448×448** can improve frame rates without severely compromising detection performance, especially when targeting large or medium-sized objects.

On NPUs such as **Qualcomm's Hexagon** or the **Apple Neural Engine**, **compiler-level optimization** becomes critical. Vendor-specific compilers like Qualcomm SNPE or CoreML require static graph definitions and strict operator compatibility, ensuring maximum throughput on custom hardware accelerators. Models often need to be re-exported through frameworks like **ONNX or MLIR** to comply with these deployment toolchains. These optimizations guarantee that computational graphs are fused and parallelized according to hardware instruction sets.

System-level optimizations are equally important. Implementing **zero-copy I/O** minimizes redundant memory transfers between CPU and GPU, while **asynchronous prefetching** and **pinned memory allocation** reduce input latency. Using **concurrent CUDA streams** enables overlapping of data loading, inference, and post-processing, thus increasing the *effective frames per second (FPS)*. Such pipeline-level efficiencies ensure that the theoretical acceleration achieved through quantization and pruning translates into real-world responsiveness.

Lastly, **robustness and reliability** are critical for field deployment. Edge models must handle environmental variability—such as **low-light conditions, rain, fog, or motion blur**—without major degradation in accuracy. Incorporating domain-specific datasets and augmentations during training improves resilience. In live video feeds, **temporal smoothing** and **tracking-by-detection** techniques like **SORT**, **DeepSORT**, or **ByteTrack** help maintain object identity across frames, reducing flicker and false reassignments.




**Summary**

Real-time object detection succeeds when architectural choices, training tactics, and deployment engineering are co-designed. One-stage CNNs (YOLO/SSD/EfficientDet) dominate latency-critical use cases; transformers are closing the gap through distillation and efficient backbones. Quantization, pruning, and compiler-level fusions are the most reliable levers to cross the 30 FPS threshold without unacceptable mAP loss. Future directions include long-tail robustness, uncertainty-aware detection, multi-modal fusion (RGB-Depth-Radar), and standardized **energy-per-frame** metrics to complement mAP and FPS.

**References**

LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep Learning. *Nature*.

Girshick, R. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation (R-CNN). *CVPR*.

Ren, S., He, K., Girshick, R., Sun, J. (2015). Faster R-CNN. *NeurIPS*.

He, K., Gkioxari, G., Dollár, P., Girshick, R. (2017). Mask R-CNN. *ICCV*.

Liu, W. et al. (2016). SSD: Single Shot MultiBox Detector. *ECCV*.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P. (2017). Focal Loss for Dense Object Detection (RetinaNet). *ICCV*.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You Only Look Once (YOLO). *CVPR*.

Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv*.

Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy. *arXiv*.

Wang, C.-Y., Liao, H.-Y. M., Yeh, I.-H. et al. (2021). Scaled-YOLOv4. *CVPR Workshops*.

Ge, Z., Liu, S., Wang, F., Li, Z., Sun, J. (2021). YOLOX. *arXiv*.

Tan, M., Le, Q. (2020). EfficientDet: Scalable and Efficient Object Detection. *CVPR*.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S. (2020). End-to-End Object Detection with Transformers (DETR). *ECCV*.

Rezatofighi, H. et al. (2019). Generalized IoU (GIoU). *CVPR*.

Zheng, Z. et al. (2020). Distance-IoU and Complete-IoU Loss. *AAAI/TPAMI* (var.).

Lin, T.-Y. et al. (2014). Microsoft COCO: Common Objects in Context. *ECCV*.

Everingham, M. et al. (2010). The PASCAL Visual Object Classes (VOC) Challenge. *IJCV*.

Howard, A. et al. (2017). MobileNetV1. *arXiv*.

Sandler, M. et al. (2018). MobileNetV2. *CVPR*.

Tan, M., Le, Q. (2019). EfficientNet: Rethinking Model Scaling. *ICML*.