

Designing Object Detection Models for TinyML: Foundations, Comparative Analysis, Challenges, and Emerging Solutions

CHRISTOPHE EL ZEINATY, Univ. Rennes, INSA Rennes, CNRS, IETR - UMR 6164, France

WASSIM HAMIDOUCHÉ, KU 6G Research Center, Khalifa University, UAE

GLENN HERROU, Univ. Rennes, INSA Rennes, CNRS, IETR - UMR 6164, France

DANIEL MENARD, Univ. Rennes, INSA Rennes, CNRS, IETR - UMR 6164, France

Object detection (OD) has become vital for numerous computer vision applications, but deploying it on resource-constrained internet of things (IoT) devices presents a significant challenge. These devices, often powered by energy-efficient microcontrollers, struggle to handle the computational load of deep learning-based OD models. This issue is compounded by the rapid proliferation of IoT devices, predicted to surpass 150 billion by 2030. TinyML offers a compelling solution by enabling OD on ultra-low-power devices, paving the way for efficient and real-time OD at the edge. Although numerous survey papers have been published on this topic, they often overlook the optimization challenges associated with deploying OD models in TinyML environments. To address this gap, this survey paper provides a detailed analysis of key optimization techniques for deploying OD models on resource-constrained devices. These techniques include quantization, pruning, knowledge distillation, and neural architecture search. Furthermore, we explore both theoretical approaches and practical implementations, bridging the gap between academic research and real-world edge artificial intelligence (AI) deployment. Finally, we compare the key performance indicators (KPIs) of existing OD implementations on microcontroller devices, highlighting the achieved maturity level of these solutions in terms of both prediction accuracy and efficiency. We also provide a public repository to continually track developments in this fast-evolving field: [Link](#).

CCS Concepts: • Computing methodologies → Object detection.

Additional Key Words and Phrases: Model Compression, Embedded Systems, Edge Computing, TinyML

1 Introduction

Technological advancements in computer vision have made OD a crucial task for applications ranging from autonomous vehicles to medical imaging and security. The primary objectives of OD are twofold: identifying the location of objects within an image and classifying them according to predefined categories (see Fig. 1a). Historically, OD relied on handcrafted features and classical machine learning methods, such as the widely known Viola-Jones detector [147] or histogram of oriented gradients (HOG) [29] combined with support vector machines (SVM). These early approaches were computationally efficient but lacked the accuracy needed for modern, large-scale applications. The advent of deep learning (DL) has transformed OD by enabling the creation of more powerful and accurate models, which typically

Authors' Contact Information: Christophe El Zeinaty, christophe.el-zeinaty@insa-rennes.fr, Univ. Rennes, INSA Rennes, CNRS, IETR - UMR 6164, Rennes, France; Wassim Hamidouche, wassim.hamidouche@@ku.ac.ae, KU 6G Research Center, Khalifa University, Abu Dhabi, UAE; Glenn Herrou, Univ. Rennes, INSA Rennes, CNRS, IETR - UMR 6164, Rennes, France; Daniel Menard, daniel.menard@insa-rennes.fr, Univ. Rennes, INSA Rennes, CNRS, IETR - UMR 6164, Rennes, France.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM



(a) OD examples showcasing a variety of detected objects and their corresponding bounding boxes.



(b) Diverse deployment scenarios of Edge AI and TinyML, including autonomous driving, UAVs, smart farming, and XR glasses illustrating varying compute, memory, and energy constraints across real-world applications.

Fig. 1. Two examples showcasing different aspects of object detection and their TinyML applications.

follow a pipeline consisting of three key components: the backbone, the neck, and the head (see Fig. 2). The backbone network is responsible for extracting feature maps from the input image, effectively acting as a feature extractor. Positioned between the backbone and the head, the neck further processes these feature maps. This often involves techniques like feature pyramid network (FPN) [90] or path aggregation networks (PANets) [98], which enable the detection of objects at various scales by merging feature maps from different layers of the backbone. Finally, the head is responsible for generating the final predictions, including bounding boxes and object classifications. Early DL-based methods, such as region-based convolutional neural networks (R-CNN) [83], introduced the concept of region proposal networks to detect objects in multiple stages, incorporating a backbone for feature extraction. This approach was followed by advancements like Faster R-CNN [124], which improved efficiency by combining region proposal generation and classification within a single framework. One-stage detectors, such as you only look once (YOLO) [123] and single shot multiBox detector (SSD) [99], further streamlined the pipeline by eliminating the region proposal step, utilizing a backbone, neck, and head to predict both class labels and bounding boxes simultaneously, achieving real-time performance. These models set new standards for OD, achieving remarkable accuracy and efficiency. However, they are computationally expensive and resource-intensive, making them impractical for deployment on edge devices with limited resources.

The growing proliferation of IoT devices, forecasted to exceed 150,55 billion such units by 2030 [67], has shifted the focus of OD research towards more resource-efficient models capable of operating on constrained hardware. Unlike cloud-based solutions, where computational power is abundant, edge devices such as smart cameras, drones, and embedded systems operate under stringent constraints in terms of memory, processing power, and energy consumption [3, 53, 122, 168]. These devices are required to perform OD tasks in real-time, often with minimal latency, to support applications such as autonomous driving, unmanned aerial vehicle (UAV) based monitoring, smart farming, and extended reality.

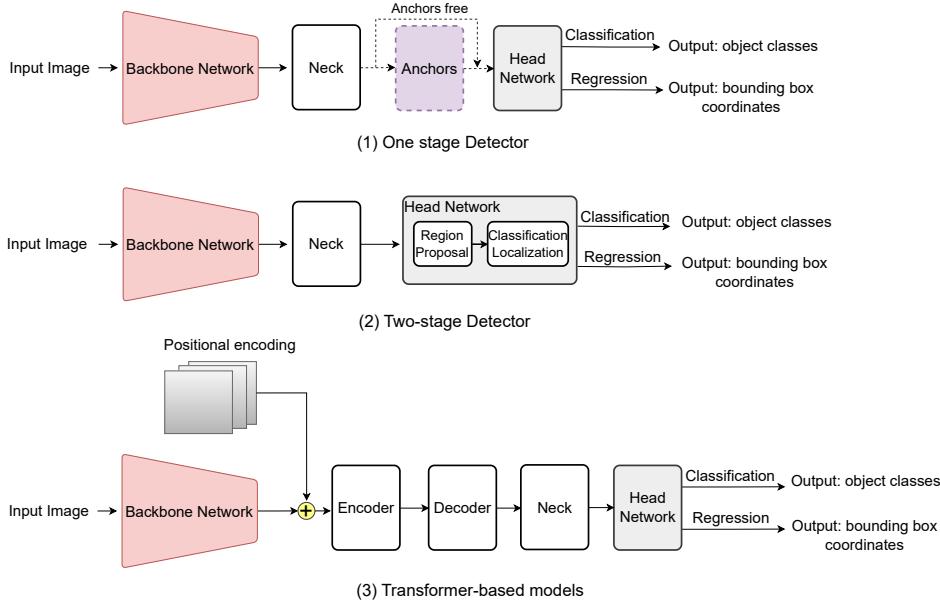


Fig. 2. Overview of object detection architectures.

(XR) interaction (see Fig. 1b). As the number of such devices continues to grow exponentially, there is an increasing demand for efficient OD models that can run locally, thus reducing reliance on cloud computing while addressing issues of latency, bandwidth, and privacy. The emergence of tiny machine learning (TinyML) has opened new avenues for enabling DL inference on ultra-low-power devices like microcontrollers (MCUs), which typically operate with less than 1MB of memory. TinyML focuses on optimizing machine learning (ML) models to fit the constraints of such devices while maintaining acceptable levels of accuracy. Techniques such as quantization, pruning, knowledge distillation (KD), and neural architecture search (NAS) [36, 56, 77, 111] have become central to this effort. Although these optimization techniques are broadly applicable across TinyML models, this paper specifically demonstrates their application to OD models, which serve as an illustrative domain to showcase their effectiveness in real-world, resource-constrained scenarios. This survey is aimed at researchers and practitioners working on optimizing DL models for deployment in resource-limited environments, particularly those focused on OD applications. By structuring our discussion around OD-specific challenges and optimization strategies, we provide targeted insights that set this work apart from general surveys on lightweight model compression. These methods enable the deployment of OD models on highly resource-constrained environments without compromising too much on performance. While several surveys [66, 89, 94, 107, 131, 151, 163, 177] have explored various aspects of OD and edge computing, many lack comprehensive coverage of key optimization techniques and fail to address the specific challenges associated with deploying OD models in resource-constrained environments, particularly with TinyML. Setyanto *et al.* [131] provide a thorough review of compression techniques across the detection pipeline, focusing on the backbone, head, and neck, but do not delve into the theoretical explanation of broader optimization techniques or TinyML deployment. Similarly, Mittal [107] explores lightweight OD models and the architecture optimization of the backbone, head, and neck but lacks a detailed analysis of optimization strategies. Huang *et al.* [151] offers a more in-depth overview of optimization methods but is limited to the backbone, with little focus on practical deployment scenarios or TinyML applications,

Table 1. Recent surveys on model compression techniques and their applications in embedded systems.

Authors	Year	Focus on OD	Optim. parts	Optim. insights [†]	Hardware Discussion	Edge AI	TinyML
Zaidi <i>et al.</i> [163]	2021	✓	Backbone	✗	✗	✓	✗
Kang <i>et al.</i> [66]	2022	✓	Backbone	✗	✓	✓	✗
Zou <i>et al.</i> [177]	2023	✓	Backbone	✗	✗	✓	✗
Setyanto <i>et al.</i> [131]	2024	✓	Backbone, head, and neck	✗	✓	✓	✗
Mittal [107]	2024	✓	Backbone, head, and neck	✗	✓	✓	✗
Huang <i>et al.</i> [151]	2024	✗	Backbone	✓	✓	✓	✗
Liu <i>et al.</i> [94]	2024	✗	Backbone	✓	✓	✓	✓
Lin <i>et al.</i> [89]	2024	✗	Backbone	✗	✓	✓	✓
Ours	2025	✓	Backbone, head, and neck	✓	✓	✓	✓

[†] The Opt. Insights column indicates whether the survey discusses in-depth optimization techniques such as quantization, pruning, KD, and NAS theoretically.

and without specifics on how these techniques can be applied to OD in particular. Liu *et al.* [94] includes a broader discussion of hardware acceleration and compression techniques, touching on TinyML challenges, but does not address OD specific optimization techniques in depth. Lin *et al.* [89] provide comprehensive insights into TinyML, including the evaluation of models on MCUs. However, in their paper, they focused on optimizing a backbone via NAS and then tested this backbone for an OD application, making it clear that OD was not the primary purpose of their work. Additionally, their focus was primarily on their own framework in the explanations provided, rather than offering a broader discussion of optimization techniques. Meanwhile, Kang *et al.* [66] provides valuable benchmarking for edge devices but focuses narrowly on the backbone and lacks theoretical explanations of optimization techniques necessary for constrained edge AI devices. Similarly, Zaidi *et al.* [163] provide a comprehensive review of lightweight networks for edge devices but only discuss backbone architectures and do not explore optimization techniques. Zou *et al.* [177] offer another review of edge AI with a focus on backbones, but they also lack a discussion on optimization techniques for OD in TinyML environments.

In this paper, we present a survey that extends beyond previous works by covering the full range of optimization techniques for OD models, with OD serving as a representative domain within TinyML environments. A detailed examination of novel optimization strategies, tailored to the unique constraints of MCUs with limited power and memory, is provided, bridging the gap between theoretical understanding and practical deployment in edge AI applications [35]. As summarized in Table 1, this comparative analysis highlights the key focus areas of prior surveys, illustrating the gaps that our work aims to fill. The remainder of this paper is structured as follows. In Section 2, an introduction to traditional and DL-based OD methods is presented, outlining the transition from handcrafted features to modern deep neural networks (DNNs). Various datasets are discussed in Section 3, while the critical evaluation metrics commonly used in OD research are detailed in Section 4 followed by a comparative analysis of different TinyML hardware in Section 5. An in-depth discussion of optimization strategies for OD models in resource-constrained environments is provided in Section 6. Furthermore, the deployment of OD models on embedded systems, with a particular focus on TinyML, is explored in Section 7. Finally, the current challenges and future research directions in this area are highlighted in Section 8, followed by the conclusion in Section 9.

2 Introduction to Object Detection

OD has undergone significant evolution, transitioning from early handcrafted feature-based methods to modern DL-driven approaches. While comprehensive surveys have explored advancements in OD in depth, this section aims to provide a structured introduction that contextualizes the optimization strategies discussed later in the paper. By outlining key developments, we emphasize how OD techniques have been shaped by the need for computational efficiency, particularly in resource-constrained environments such as embedded systems and TinyML applications.

In the realm of computer vision, OD stands as a pivotal task, representing the fundamental capability to identify and precisely locate objects within digital images or video frames. Unlike mere image classification, which assigns a single label to an entire image or a patch, OD ventures further. It not only discerns the categories of objects present but also draws bounding boxes around each object, pinpointing their spatial coordinates within the visual field. This dual functionality empowers computer systems to not only recognize objects but to understand their spatial context, opening the door to a multitude of applications that require nuanced scene understanding, making OD essential for applications in autonomous driving, surveillance, medical imaging, and robotics. In TinyML, a common application is person detection, which simplifies the task by focusing on binary classification: determining whether a person is present or not, without requiring localization. This makes person detection less computationally intensive than full OD, which involves both classification and localization. While person detection is widely used in real-time applications such as surveillance, where precise localization may not always be necessary, the ability of OD to accurately localize objects remains critical in more complex industrial use cases. Benchmarks like MLPerf Tiny [6], which include person detection as one of the use cases, underscore the importance of efficient models for resource-constrained devices like MCUs. This highlights the value of studying and optimizing OD on such devices, where both real-time performance and localization accuracy are essential.

The inception of OD can be attributed to the pioneering work of Viola and Jones, who introduced the groundbreaking Viola-Jones framework [147], a significant milestone in computer vision. This framework employed Haar-like features [106] and a cascade classifier to achieve real-time face detection [28]. Haar-like features are simple rectangular filters that efficiently detect variations in brightness, while the cascade classifier enabled rapid rejection of non-object regions, leading to substantial speed improvements. Despite its breakthrough in real-time face detection, early OD methods, including the Viola-Jones framework, faced challenges in handling complex object categories and variations in object appearance. These limitations spurred the quest for more sophisticated approaches as computer vision tasks grew in complexity.

The landscape of OD underwent a significant transformation with the emergence of DNNs. DL techniques showcased remarkable capabilities in feature learning and abstraction, rendering them highly suitable for OD tasks. DNN, particularly convolution neural networks (CNNs), brought about a paradigm shift in OD. They not only surpassed traditional methods in performance but also enabled end-to-end learning, eliminating the need for handcrafted features. CNN-based OD models became adept at automatically learning discriminative features from data, resulting in substantial improvements in detection accuracy.

In recent years, the architecture of OD models has evolved significantly, with major developments categorized into traditional methods and DL-based methods.

2.1 Traditional Methods

Traditional OD methods, such as the Viola-Jones framework, HOG [29], and deformable parts model (DPM) [41], relied on handcrafted features and simple classifiers. Nevertheless, these methods often struggled with the complexity and variability of real-world objects.

In the 1990s, early OD algorithms were built upon handcrafted features due to the absence of effective learned image representation techniques at that time. These algorithms necessitate sophisticated feature engineering and various acceleration techniques to achieve acceptable performance. In 2001, Viola and Jones [147] achieved real-time detection of human faces using a 700-MHz Pentium III CPU, making it tens or even hundreds of times faster than previous methods. Their approach used a sliding window technique to scan all possible locations and scales in an image for

human faces. They dramatically improved detection speed by incorporating three key techniques: the "integral image," "feature selection," and "detection cascades."

In 2005, Dalal and Triggs [29] introduced the HOG descriptor, which improved upon previous feature descriptors like the scale-invariant feature transform (SIFT) and shape contexts. HOG balanced feature invariance and nonlinearity, primarily used for pedestrian detection. The detector rescales the input image multiple times to detect objects of different sizes while keeping the detection window size unchanged. The HOG detector became foundational for many object detectors and computer vision applications. In 2008, Felzenszwalb *et al.* [41] proposed the DPM, which extended the HOG detector using a "divide and conquer" approach. This method involves decomposing objects into parts (e.g., a car's window, body, and wheels) and detecting these parts individually. Girshick [47] subsequently extended the DPM framework to incorporate "mixture models," enhancing its capacity to handle the greater variability inherent in real-world objects. The DPM's influence reverberated through the development of many modern detection techniques, contributing valuable insights such as mixture models [148], hard negative mining [62], bounding box regression [104], and context priming [146]. In recognition of their seminal work, Felzenszwalb and Girshick were honored with the "lifetime achievement" award from pascal visual object classes (PASCAL VOC) in 2010.

2.2 Deep Learning-based Methods

DL-based methods, utilizing DNNs such as CNNs, have revolutionized OD by facilitating end-to-end learning and feature extraction. This paradigm shift has led to substantial improvements in the accuracy and efficiency of OD systems, empowering them to learn intricate features directly from the data. Within the realm of DL, OD models can be broadly categorized into two architectural designs: two-stage detectors and one-stage detectors, as illustrated in Fig. 2.

2.2.1 Two-Stage Detectors. Two-stage detectors, exemplified by models such as R-CNN [83] and Faster R-CNN [124], employ a two-step process. In the initial stage, these models generate a set of region proposals that are likely to contain objects of interest. Subsequently, in the second stage, these proposals undergo classification and refinement to achieve precise object localization. This approach allows higher detection accuracy, as the model can concentrate its computational resources on promising image regions. However, the additional processing step often leads to slower inference times compared to one-stage detectors.

2.2.2 One-Stage Detectors. One-stage detectors, represented by models such as YOLO [123], and SSD [99], directly predict object categories and bounding boxes in a single evaluation step. These models are optimized for real-time applications due to their high speed and efficiency, eliminating the need for a separate region proposal stage. However, while one-stage detectors generally offer faster inference, they may exhibit a slight trade-off in accuracy compared to their two-stage counterparts.

Earlier one-stage methods primarily relied on anchor boxes (i.e., prior boxes) to provide reference points for classification and regression tasks. However, the inherent variability of objects in terms of number, location, scale, and aspect ratio often necessitated the use of a large number of reference boxes to achieve a close match with ground-truth annotations and attain high performance. To address this limitation, several anchor-free detectors, such as [34], and CornerNet [76], were proposed. These models eliminate the reliance on predefined anchor boxes, thereby simplifying the detection pipeline and reducing computational overhead. By design, anchor-free models offer improved generalization and flexibility in detecting objects of varying sizes and shapes.

Recently, transformer-based models such as DETR [16] and RT-DETR [171] have emerged, leveraging attention mechanisms to improve detection performance and robustness. These models directly model spatial relationships

between different image regions, enabling a more sophisticated contextual understanding of OD. Expanding on this paradigm, Grounding DINO 1.5 [125] introduces linguistic grounding into OD, significantly enhancing flexibility in open-set scenarios. The Grounding DINO 1.5 Pro model scales both the architecture and training datasets to over 20 million images, achieving state-of-the-art zero-shot transfer capabilities. Meanwhile, the Edge variant, optimized for real-time applications, achieves 75.2 frames per second (FPS) with an input size of 640×640 using a TensorRT-optimized pipeline without compromising detection robustness. These advancements underscore the shift towards more adaptable and efficient models for both closed-set and open-world OD applications. Building on the query-based detection framework established by DETR [16], diffusion models [133] have introduced a novel approach to OD, further streamlining the detection pipeline. Wang *et al.* [21] proposed DiffusionDet, which formulates detection as a denoising diffusion process, iteratively refining random initial boxes into precise object locations. Unlike traditional models that depend on predefined object priors, DiffusionDet eliminates hand-designed queries and learnable object representations, offering a simplified yet highly effective detection pipeline that pushes the boundaries of OD performance. Beyond traditional OD, generative AI is increasingly being explored for zero-shot detection tasks. RONIN [112], for instance, introduces a diffusion-based zero-shot object-level out-of-distribution (OOD) detection framework. By leveraging context-aware inpainting, RONIN detects whether an object is in or out-of-distribution without requiring access to the original training set, making it particularly suited for black-box models where training data is unavailable. This generative approach further exemplifies how diffusion models can enhance object-level understanding in a zero-shot setting. Meanwhile, the YOLO family continues to evolve, with recent advancements such as YOLOv12 [145] and YOLO-E [149] enhancing real-time OD. YOLOv12 improves detection efficiency through dynamic label assignment and attention-based mechanisms, while YOLOE extends the traditional YOLO framework with open-set prompt mechanisms, including text and visual prompts, broadening its applicability beyond predefined categories. Additionally, YOLOE integrates novel components like re-parameterizable region-text alignment (RepRTA) and semantic activated visual prompt encoder (SAVPE), which strengthen vision-language alignment and enable robust zero-shot detection. These advancements reinforce the trend toward highly adaptable and efficient models capable of operating in both constrained and open-world settings. Further bridging the gap between convolutional and transformer-based architectures, hybrid models like RT-DETR [171] and YOLO-World [24] combine the efficiency of convolutional backbones with the contextual learning power of transformers. This fusion balances speed, accuracy, and generalization, making them promising candidates for diverse OD applications. By leveraging advances in transformers, diffusion models, convolutional networks, and hybrid architectures, modern OD systems continue to enhance efficiency and adaptability, paving the way for further breakthroughs in real-world detection tasks.

2.3 Loss Optimisation

OD models generally strive to optimize a combination of classification and localization tasks. The classification task entails predicting the category of objects present within proposed regions, whereas the localization task involves predicting the precise coordinates of the corresponding bounding boxes. Recent models leverage DL to learn these tasks jointly, thereby achieving substantial improvements in accuracy and efficiency compared to traditional methods.

Mathematically, the overall loss function L employed for training an OD model is typically formulated as a weighted sum of two components: the classification loss L_{cls} and the localization loss L_{loc} . To allow for a more flexible balance between these two components, we introduce a hyperparameter β for the classification loss and α for the localization

loss. This relationship is expressed as follows:

$$L = \beta L_{cls} + \alpha L_{loc}, \quad (1)$$

where α and β are hyperparameters that determine the relative importance of the localization and classification loss components, respectively.

2.3.1 Classification Loss. The classification loss L_{cls} is typically computed using the cross-entropy loss function, which quantifies the discrepancy between the predicted class probabilities and the corresponding ground-truth class probabilities. For each predicted bounding box, the classification loss can be formally defined as:

$$L_{cls} = - \sum_{i=1}^C y_i \log(\hat{p}_i) \quad (2)$$

where C is the total number of classes, y_i represents the ground-truth probability for class i (1 for the correct class and 0 for others), and \hat{p}_i denotes the predicted probability for class i .

In addition to the standard cross-entropy loss, several variations are commonly used to address issues such as class imbalance and model overfitting. One such variation is Focal Loss, which down-weights the easy examples and focuses more on hard-to-classify examples. This is particularly useful in OD tasks where certain classes may dominate the training data, causing the model to overlook harder, less frequent classes. Focal loss can be expressed as:

$$L_{focal} = - \sum_{i=1}^C \alpha_i (1 - \hat{p}_i)^\gamma y_i \log(\hat{p}_i) \quad (3)$$

where α_i is a balancing factor for class i , and γ is a focusing parameter that reduces the relative loss for well-classified examples. Another common approach is Label Smoothing, which modifies the ground-truth labels by softening the targets. This can help prevent the model from becoming too confident about its predictions and improve generalization. Label smoothing can be incorporated into the cross-entropy loss by adjusting the ground-truth labels y_i to

$$\tilde{y}_i = (1 - \epsilon)y_i + \frac{\epsilon}{C}, \quad (4)$$

where ϵ is the smoothing factor and C is the number of classes. These modifications to the classification loss function allow the model to perform better on imbalanced datasets, reduce overfitting, and improve generalization to unseen examples.

2.3.2 Localization Loss. The localization loss L_{loc} measures the error in the predicted bounding box coordinates. A common choice is the Smooth L1 loss (Huber loss), which is less sensitive to outliers than the L2 loss. For a predicted bounding box with coordinates $(\hat{x}, \hat{y}, \hat{w}, \hat{h})$ and the ground-truth box (x, y, w, h) , the Smooth L1 loss is defined as follows:

$$L_{loc} = \sum_{i \in \{x, y, w, h\}} S(\hat{i} - i), \quad (5)$$

where $S(x)$ is the Smooth L1 function, defined as:

$$S(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1, \\ |x| - 0.5 & \text{otherwise.} \end{cases} \quad (6)$$

Table 2. Comparison of Object Detection Datasets

Dataset	Year	Classes	Train		Validation		Test
			Images	Objects	Images	Objects	
PASCAL VOC [38]	2012	20	5,717	13,609	5,823	13,841	10,991
ILSVRC [128]	2015	200	456,567	478,807	20,121	55,501	40,152
MS-COCO [92]	2017	80	118,287	860,001	5,000	36,781	40,670
OpenImage [75]	2020	600	1,743,042	14,610,229	41,620	204,621	125,436

In addition to Smooth L1 loss, several other regression loss functions are commonly used in OD to improve bounding box prediction accuracy. For instance, L2 loss (Euclidean distance) is a basic method but can be more sensitive to outliers. It computes the Euclidean distance between the predicted and ground-truth bounding box coordinates. While simple and effective in many cases, it may be more sensitive to large errors than Smooth L1 loss, especially when the predicted bounding boxes are far from the ground truth. Furthermore, IoU-based loss functions, such as generalized IoU (GIoU), distance IoU (DIoU), and complete IoU (CIoU), have been proposed to directly optimize the overlap between predicted and ground-truth boxes, thereby improving localization performance. These functions focus on increasing the overlap between the predicted and ground-truth boxes. For example, GIoU extends IoU by penalizing the distance between the bounding boxes centers and increasing robustness to boxes with no overlap. Similarly, DIoU and CIoU refine bounding box predictions by incorporating additional geometric information, such as aspect ratio and center distance.

In summary, OD models employ a multi-task loss function that effectively balances the objectives of classification accuracy and localization precision. This dual optimization empowers the models to not only identify the categories of objects present in an image but also to accurately pinpoint their spatial locations within the visual scene.

3 Datasets

3.1 Object Detection

For OD, diverse and well-annotated datasets are essential as they provide the foundation for training, evaluating, and benchmarking models. Table 2 summarizes several well-known datasets frequently used in OD research. In this review, the primary focus will be on two prominent datasets that have significantly contributed to the advancement of the field: **Common objects in context (COCO)** [92]. COCO is a widely recognized dataset in the computer vision community. It contains a vast collection of images, each meticulously annotated with object categories and precise object bounding box coordinates. COCO's richness lies not only in the diversity of objects but also in the complexity of scenes, making it an invaluable resource for training and evaluating OD models. With 80 object categories and over 330,000 images, including more than 1.5 million annotated instances, COCO has become a benchmark for measuring the performance of modern OD algorithms.

PASCAL VOC [38]. The PASCAL VOC dataset stands as another seminal benchmark that has played a pivotal role in the advancement of OD research. Featuring a diverse range of object categories, it provides annotated images accompanied by corresponding object bounding boxes. PASCAL VOC has served as a standard benchmark dataset for numerous years, enabling the rigorous evaluation and comparison of various OD methods. The dataset encompasses 20 object categories, with approximately 11,530 images containing 27,450 annotated objects. While it may not be as extensive as the COCO dataset, its historical significance and contribution to shaping the early years of OD research remain undeniable. In addition to these two datasets, large-scale datasets such as ImageNet large scale visual recognition challenge (ILSVRC) [128] and OpenImage [75] have also contributed significantly to OD research. ILSVRC, with 200

Table 3. Comparison of TinyML Datasets

Dataset Name	Year	Use Case	Number of Samples	Evaluation Metric
Speech Commands [157]	2018	Keyword Spotting	105,829	Accuracy
ToyAdmos Dataset [70]	2019	Sound Anomaly Detection	5,939	AUC-ROC, Precision-Recall
VWW Dataset [26]	2019	Person Detection	115,000	Accuracy
Wake Vision Dataset (Quality) [5]	2024	Person Detection	1,322,574	Accuracy
Wake Vision Dataset (Large) [5]	2024	Person Detection	5,760,428	Accuracy

object categories and over 1.2 million training images, has been instrumental in advancing both classification and detection tasks. Similarly, the OpenImage dataset, with its vast 600 object categories and more than 9 million annotated objects, is frequently used for training models that generalize well to real-world OD scenarios. Although this survey focuses on COCO and PASCAL VOC, the availability of large-scale datasets like ILSVRC and OpenImage has broadened the scope of OD research, facilitating improvements in model robustness and scalability.

3.2 Other TinyML Datasets

While this paper primarily focuses on OD models for object classification and localization within images, it is essential to highlight other well-established use cases in TinyML. As discussed in [6], some of the most prominent applications include person detection, image classification, keyword spotting, and anomaly detection. Table 3 provides a summary of these key use cases along with the most widely used datasets in each domain.

Speech Commands Dataset [157]. The Speech Commands dataset is designed for keyword spotting, enabling efficient on-device speech recognition. Released by Google Brain in 2018, it contains 105,829 utterances from 2,618 speakers, covering 35 words. The dataset primarily supports small-footprint models for detecting predefined words such as "Yes," "No," "Up," "Down," and digits. It also includes background noise samples to improve robustness against false activations. The dataset has been widely adopted in TinyML applications, particularly for wake word detection in smart assistants.

Visual Wake Words (VWW) Dataset [26]. The Visual Wake Words dataset is specifically designed for TinyML applications, particularly for low-power vision models deployed on MCUs. Introduced by Google Research in 2019, the dataset addresses the common TinyML use case of determining whether a person is present in an image or not. It is derived from the COCO dataset by filtering and relabeling images based on the presence of a person occupying at least 0.5% of the image area. The dataset contains approximately 115,000 images split into training and validation sets, making it a realistic benchmark for evaluating tiny vision models with extreme memory constraints (e.g., models under 250 KB). VWW has been widely adopted in TinyML benchmarks to assess the trade-offs between model accuracy, memory footprint, and computational efficiency on MCUs.

Wake Vision Dataset [5]. Wake Vision is a large-scale dataset specifically designed for person detection in TinyML applications. It contains over 5 million images and is provided in two variants: Wake Vision (Large) and Wake Vision (Quality). The large variant focuses on dataset scale for pretraining and KD, while the quality variant prioritizes high-quality labels for final model performance. The dataset is derived from Open Images v7 and improves accuracy by 1.93% over existing datasets such as Visual Wake Words (VWW). Additionally, Wake Vision introduces a benchmark suite evaluating model robustness across real-world conditions, including lighting variations, camera distances, and demographic characteristics.

ToyADMOS Dataset [70]. The ToyADMOS dataset is a large-scale dataset designed for anomaly detection in machine operating sounds (ADMOS), making it highly relevant for TinyML applications. Released in 2019, it includes over 180

hours of normal machine-operating sounds and more than 4,000 samples of anomalous sounds. The dataset is divided into three sub-datasets: (1) product inspection using a toy car, (2) fault diagnosis of a fixed machine using a toy conveyor, and (3) fault diagnosis of a moving machine using a toy train. Sounds were recorded with four microphones at a 48-kHz sampling rate under controlled conditions. Anomalous sounds were generated by deliberately damaging machine components, allowing for realistic fault detection evaluation. The dataset is also used in the MLPerf Tiny benchmark suite to evaluate TinyML models for sound anomaly detection, demonstrating its importance in resource-constrained machine learning applications.

4 Evaluating Object Detection Models

In this section, the evaluation criteria and metrics employed to assess the performance of OD models are examined, highlighting the compromise between quality and cost. The metrics used to assess general performance in terms of OD accuracy are first presented (§ 4.1), followed by an exploration of the efficiency metrics tailored for embedded devices, where both aspects will be discussed (§ 4.2).

4.1 Accuracy Performance

In the context of OD, average precision (AP) and mean average precision (mAP) are commonly employed in conjunction to evaluate the performance of models. AP is calculated by plotting the precision-recall curve for each individual class. This curve graphically depicts the trade-off between precision (the proportion of true positive detections among all positive detections) and recall (the proportion of true positive detections among all actual positive instances) at varying confidence thresholds. In OD, a detection is considered a true positive (TP) if its intersection over union (IoU) with the ground truth bounding box is greater than a specified threshold (e.g., 0.5), and a false positive (FP) if the IoU is below that threshold. A false negative (FN) occurs when an object in the ground truth is not detected. AP is then computed as the area under curve (AUC) of this precision-recall curve, providing a single numerical value that summarizes the model's performance for that specific class. In the PASCAL VOC challenge, AP is computed using interpolated precision at a set of 11 recall levels, which can be represented by the set $R = \{0, 0.1, 0.2, \dots, 1.0\}$. The interpolated precision $p_{interp}(r)$ at a particular recall level $r \in R$ is defined as the maximum precision achieved for any recall level greater than or equal to r :

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \quad (7)$$

The AP for a particular class c is then given by:

$$AP_c = \frac{1}{11} \sum_{r \in R} p_{interp}(r) \quad (8)$$

The use of 11 recall levels in this evaluation protocol is designed to strike a balance between computational efficiency and adequately representing the precision-recall curve. The 11-point interpolation smooths out small fluctuations, or "wiggles," in the precision-recall curve that arise from minor variations in the ranking of detection results. These variations are often insignificant in large evaluation datasets, and the 11-point method is sufficient to provide a stable and reliable measure of model performance.

Complementarily, mAP provides a comprehensive metric that encapsulates both precision and recall aspects of model performance by averaging the AP values across all classes. This process involves calculating AP for each individual class and then computing the arithmetic mean of these values.

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (9)$$

where N is the number of object classes.

In PASCAL VOC, the mAP@0.5 threshold is used, where a detection is considered correct if its IoU with the ground truth is greater than 50%. This threshold was deliberately set low to account for inaccuracies in bounding boxes in the ground truth data.

As highlighted in recent literature, such as the comprehensive review of YOLO architectures by Terven and Cordova-Esparza [144], the mAP metric has evolved in tandem with advancements in OD frameworks. For instance, YOLO models typically leverage the COCO dataset for benchmarking, which employs a more sophisticated method for calculating AP, incorporating multiple IoU thresholds. The introduction of 101 recall points and multiple IoU thresholds by COCO represents a natural evolution of evaluation techniques in OD. As OD models began to saturate, it became clear that models with equivalent scores were not performing equally. COCO addressed this by increasing the granularity of evaluation, with mAP@[0.5:0.95], calculated for 10 different IoU thresholds ranging from 0.5 to 0.95 in 0.05 increments. This evolution provides a more detailed performance profile, allowing for more accurate comparisons between models by assessing their localization accuracy across varying levels of overlap with ground truth boxes. This method provides a finer evaluation of model performance compared to the simpler 11-point interpolation method used in PASCAL VOC, enabling a better understanding of detection performance under stricter localization constraints.

4.2 Efficiency evaluation on embedded systems

Evaluating the performance of OD models on embedded systems necessitates a comprehensive understanding of several key metrics. Each metric provides insights into how effectively the system operates under the constraints characteristic of resource-limited edge computing environments. This section outlines the essential performance indicators used in benchmarking OD models deployed on embedded systems. Some of these metrics are **target-independent**, providing insights that apply regardless of the specific hardware used, while others are **target-dependent**, as their values can vary depending on the capabilities and constraints of the hardware platform.

Computational complexity (Target-independent). Floating point operations per seconds (FLOPs) and multiply-accumulate operationss (MACs) are crucial metrics for evaluating the computational complexity of OD models. FLOPs provide a measure of the total number of floating-point operations required encompassing additions, multiplications, and other arithmetic operations while MACs, a specific type of operation, represent the combined multiplication and addition operations frequently encountered in neural networks (NNs) (especially in convolutional layers and matrix multiplications), often serving as a proxy for overall computational complexity. Both FLOPs and MACs directly correlate with the processing power and computational demands of a model. It is important to note that while FLOPs and MACs are target-independent measures (quantifying the total number of operations performed by a model), their practical impact on runtime and energy efficiency depends on hardware-specific factors. Efficiency is often associated solely with metrics such as FLOPs/W (operations per second per Watt), but relying exclusively on these measures can be misleading. In practice, actual performance is also governed by factors like parallelism, memory bandwidth, and data movement overhead which these metrics do not capture [139]). For clarity, these metrics are typically expressed in MegaFLOPs (10^6 FLOPs) and MegaMACs (10^6 MACs), providing a theoretical measure of a model's computational load. Benchmarking models on these metrics aids in selecting architectures that balance accuracy with computational efficiency. Reference

Table 4. Comparison of TinyML Hardware Platforms

Platform	Core	Frequency	SRAM	Flash	AI Accelerator	Approx. Cost
STM32H743	Cortex-M7	480 MHz	1 MB	2 MB	✗	\$5–\$10
STM32F746	Cortex-M7	216 MHz	320 KB	1 MB	✗	\$4–\$8
STM32N6	Cortex-M55 + NPU	800 MHz	4.2 MB	8MB (external)	✓	\$20–\$25
GAP9	8-core RISC-V cluster	175 MHz	1.6 MB	2 MB	✓	\$10–\$15
Kendryte K210	Dual-core RISC-V (RV64GC)	400 MHz	512 KB	8 MB (external)	✓	\$5–\$10
MAX78000	Cortex-M4F	100 MHz	128 KB	512 KB	✓	\$10–\$15
ESP32	Dual-core Tensilica LX6	240 MHz	520 KB	4 MB (external)	✗	\$2–\$4
Himax WE-I Plus	ARC EM9D DSP	400 MHz	2 MB	2 MB	✗	\$10–\$20

models such as MobileNets [57] and ShuffleNet [169] are designed to significantly reduce both FLOPs and MACs while maintaining robust performance, making them well-suited for resource-constrained environments.

Memory Footprint (Target-dependent). Given the stringent memory constraints often imposed on embedded systems, optimizing the memory footprint of OD models is of paramount importance. Efficient utilization of static random-access memory (SRAM) ensures the system can effectively manage its resources to perform tasks without necessitating additional hardware enhancements. The unit for memory usage is typically in kilobytes (KB) in TinyML applications, as these are common for low-power, edge devices with limited memory resources. Techniques for reducing memory requirements include model compression and the adoption of streamlined NNs architectures. Notably, evaluations of models like MCUNetV1 [88], MCUNetV2 [87], EtinyNet-SSD [162], TinyissimoYOLO [110], and XiNet [4] have demonstrated significant progress in this area, showcasing ultra-efficient designs that maintain high performance even within the tight memory limitations characteristic of MCUs. These models will be discussed in more detail in Section 7.

Runtime (Target-dependent). Runtime is an important metric, especially for real-time applications such as autonomous navigation and interactive systems. It is crucial to minimize response times without compromising accuracy to ensure the system remains both reliable and efficient under operational conditions. The unit of runtime is typically in seconds (s) or milliseconds (ms). Research endeavors, such as those presented by Han Cai *et al.* [13] on dynamically adjustable networks, demonstrate innovative methodologies for optimizing runtime while preserving high levels of accuracy.

Energy Consumption (Target-dependent). Energy efficiency is of paramount importance in embedded systems, particularly those dependent on battery power or situated in remote locations. Judicious energy utilization directly translates to extended operational lifespan and enhanced system reliability. The unit for energy consumption in TinyML applications is typically in millijoules (mJ) or microjoules (μ J), as these are standard for measuring small-scale power consumption in embedded devices. Benchmarking energy consumption, as shown in the research by Moosmann *et al.* with TinyissimoYOLO [110], involves a meticulous assessment of how model adaptations can curtail power usage while preserving functional integrity.

5 TinyML Hardware Platforms

TinyML hardware platforms are essential for enabling the deployment of DL models in resource-constrained environments. Unlike cloud-based architectures, TinyML platforms must efficiently balance computational throughput, memory constraints, and energy consumption to achieve real-time inference. These platforms mainly consist of MCUs with limited resources, though some integrate specialized AI accelerators to enhance efficiency for OD tasks. Table 4 presents a comparative overview of key TinyML hardware platforms. General-purpose MCUs, such as the STM32H743,

integrate a Cortex-M7 core running at 480 MHz with 1 MB of SRAM and 2 MB of Flash, providing sufficient processing power for lightweight NNs. The STM32F746, featuring a lower clock speed of 216 MHz and reduced memory capacity (320 KB SRAM), prioritizes power efficiency. Recent advancements, such as the STM32N6 series, introduce a Cortex-M55 core with an integrated neural processing unit (NPU), leveraging hardware acceleration to optimize inference performance while maintaining stringent power constraints. Ultra-low-power platforms such as the ESP32 and Himax WE-I Plus further expand TinyML applications to energy-sensitive environments. Beyond traditional MCUs, some platforms incorporate dedicated AI accelerators to enable low-power, high-performance inference. For instance, the MAX78000 combines a Cortex-M4F core with an embedded CNN accelerator, facilitating efficient low-power OD. Similarly, the Kendryte K210, a dual-core RISC-V processor with an integrated Kendryte Processing Unit (KPU), is specifically optimized for CNNs, providing an efficient balance between power consumption and performance. Among advanced TinyML accelerators, the GreenWaves GAP9 stands out as a highly efficient platform capable of executing complex DL workloads while maintaining low power consumption. Unlike traditional MCUs, GAP9 features an 8-core processing cluster, 1.6MB of SRAM, and a dedicated neural accelerator, enabling real-time execution of models such as TinyissimoYOLO for smart glasses [109]. Its energy-efficient design, coupled with optimized memory hierarchies and parallel processing, makes it particularly suited for OD applications requiring real-time inference in wearable and embedded devices.

The acceleration of OD on TinyML hardware is fundamentally dependent on specialized processing architectures designed to optimize key mathematical operations. Traditional MCUs, which rely solely on scalar CPU cores, face inherent limitations in executing tensor-intensive computations. To address this, modern TinyML hardware integrates NPUs and CNN accelerators, which leverage parallelism and optimized execution pipelines to enhance performance. These accelerators significantly improve the efficiency of operations such as matrix multiplications, convolutions, and activation functions, which are computationally demanding on resource-constrained devices.

Mathematically, the computational complexity of convolutional layers in CNN-based OD models is expressed as:

$$C = O_h O_w C_o K_h K_w C_i, \quad (10)$$

where O_h, O_w denote the output feature map dimensions, C_o is the number of output channels, K_h, K_w are the kernel dimensions, and C_i represents the number of input channels. The presence of dedicated CNN accelerators, such as in the MAX78000 and Kendryte K210, reduces this complexity by executing convolution operations in parallel, thereby lowering computational latency from $O(n^2)$ to $O(n)$ per pixel in optimized hardware.

To further optimize execution, many TinyML hardware platforms incorporate specialized memory hierarchies and direct memory access (DMA) mechanisms to facilitate efficient data movement. For example, the GAP9 integrates a tightly coupled memory system that minimizes data transfer overhead between its multi-core cluster and neural accelerator, reducing inference latency. Efficient scheduling of memory access patterns also allows for pipelined execution, improving overall throughput. Further efficiency gains are achieved through specialized instruction sets tailored for DL inference. Many NPUs and CNN accelerators support vectorized operations, which allow simultaneous execution of MAC operations. For example, the STM32N6 leverages mixed-precision arithmetic, dynamically adjusting computation precision between 16-bit and 8-bit representations to optimize both accuracy and power efficiency. This approach is particularly beneficial in OD tasks, where object localization computations often require higher precision than classification. As summarized in Table 4, the selection of TinyML hardware significantly influences the feasibility of deploying OD models in real-world applications. While high-performance MCUs, such as the STM32H743, provide the computational capacity to execute OD algorithms, they lack dedicated acceleration hardware. In contrast, platforms

Table 5. Summary of recent lightweight OD models and their performance on COCO and PASCAL VOC datasets. All results are compiled from the corresponding papers, and readers are encouraged to consult these papers for a comprehensive understanding of various hyperparameters and specific model details.

Optim.	Model	Year	Optim. parts	Params (M)↓	AP50:95 (COCO)↑	mAP (Pascal VOC)↑
Model Design	MobileNetV1 + SSDLite [129]	2018	backbone	5.1	22.2	68.0
	MobileNetV2 + SSDLite [129]	2018	backbone	4.3	22.1	70.9
	Tiny-DSOD [82]	2018	backbone, neck	1.15	23.2	72.1
	LightDet [142]	2020	backbone, neck, head	-	24.0	75.5
	GnetDet [135]	2021	backbone, head	-	-	66.0
	YOLOv5-N [63]	2022	backbone, neck, head	1.9	28.0	-
	YOLOv5-S [63]	2022	backbone, neck, head	7.2	37.4	-
	MobileDenseNet [51]	2023	backbone	5.8	24.8	76.8
	YOLOv11-N [64]	2024	backbone, neck, head	2.6	39.5	-
Pruning	YOLOv12-N [145]	2025	backbone, neck, head	2.6	40.6	-
	Lite-YOLOv3 [174]	2023	backbone, neck, head	15.3	52.4	74.1
BNNs	Faster-RCNN-Bi-Real Net [103]	2018	backbone	20.1	-	58.2
	Unified network [136]	2018	backbone	-	-	44.3
	ResNet-50 GroupNet-c [176]	2019	backbone	-	33.9	74.4
	ResNet-34 GroupNet-c [176]	2019	backbone	-	32.8	69.3
	ResNet-18 GroupNet-c [176]	2019	backbone	-	30.1	63.6
	Faster R-CNN-BiDet-Resnet18 [156]	2020	backbone, head	20	15.7	59.5
	Faster R-CNN-DA-BNN [170]	2020	backbone, head	-	-	64.0
NAS	Faster-RCNN-XNOR-Net [170]	2022	backbone, head	22.2	-	48.9
	DetNASNet (FPN) [23]	2019	backbone	-	36.6	81.5
	DetNASNet (RetinaNet) [23]	2019	backbone	-	33.3	80.1
	NATS-C [117]	2019	backbone	-	38.4	-
	NAS-FCOS [153]	2019	neck, head	-	46.1	-
	NAS-FPNLite + MobileNetV2 [45]	2019	neck	2.16	24.2	-
	MobileNetV2 + MnasFPN [17]	2019	head	1.29	23.8	-
	Hit-Detector [49]	2020	backbone, neck, head	27	41.4	-
	YOLO-NAS-Small [2]	2021	backbone, neck, head	19	47.5	-
KD	MCUNetV2 [87]	2021	backbone	0.67	-	68.3
	DarkNet-YOLO-BWN-KT [161]	2018	backbone	-	-	65.0
	MOBILENET-Yolo-BWN-KT [161]	2018	backbone	-	-	63.0
	Centernet-34 [72]	2021	backbone, head	-	-	75.8
Quantization	Centernet-50 [72]	2021	backbone, head	-	-	77.1
	EtinyNet† [162]	2022	backbone	0.59	-	56.4
	TinyissimoYOLO† [110]	2023	backbone, neck, head	0.703	-	56.4

† These models represent significant advancements in state-of-the-art, having been deployed on MCUs, demonstrating exceptional efficiency in memory-constrained environments.

with integrated NPUs and CNN accelerators, such as the STM32N6, GAP9, and Kendryte K210, demonstrate substantial efficiency improvements by offloading key processing tasks to specialized neural units. These advancements underscore the increasing importance of hardware-driven optimizations in achieving real-time, energy-efficient OD in TinyML environments [109].

6 Advanced Techniques in Model Optimization for Embedded Systems

In this section, advanced techniques for optimizing OD models specifically designed for resource-constrained embedded systems are examined. These techniques are critical for enhancing model efficiency, reducing computational overhead, and ensuring high performance within the limited resources available in embedded environments. Unlike general reviews on network lightweighting, this section focuses specifically on OD as an illustrative application, highlighting how various components of the OD pipeline, namely the backbone, neck, and head, are optimized in the literature to meet the constraints of embedded AI. By structuring the discussion around these core components, we emphasize how different optimization techniques are applied at each stage of the detection pipeline, demonstrating their impact on both computational complexity and accuracy. Key optimization strategies are explored, broadly categorized into three primary approaches: parameter removal, parameter quantization, and parameter search. Parameter removal techniques, including unstructured, structured, and semi-structured pruning, focus on strategically eliminating redundant parameters to

streamline the model and reduce its computational footprint. Parameter quantization methods, such as quantization-aware training (QAT), post-training quantization (PTQ), and binarized neural networks (BNNs), aim to reduce the precision of model parameters, thereby lowering memory requirements and computational demands. Finally, parameter search approaches, including KD, NAS, and its various methodologies such as reinforcement learning (RL)-based NAS, evolutionary algorithm (EA)-based NAS, and gradient-based NAS are discussed. Table 5 compares some of the state-of-the-art OD models that leverage these optimization techniques, highlighting their performance on both the COCO and PASCAL VOC datasets. The table illustrates the trade-offs between model complexity, parameter size, and detection accuracy (AP50:95 and mAP), showcasing how each optimization strategy impacts model performance across different benchmarks. These advanced techniques are further summarized in Fig. 3, providing a comprehensive overview of the approaches examined in this section.

6.1 Quantization

Quantization is crucial for optimizing NNs for hardware deployment, involves converting high-precision values into lower-bit representations. This process, essential for lightweight DL, is mathematically expressed as:

$$\hat{x} = Q(x) = \text{round}\left(\frac{x}{s}\right) \times s \quad (11)$$

where \hat{x} represents the quantized value of the original value x , and s the scaling factor. Quantization effectively reduces MAC operations and DNN size, thereby accelerating both training and inference phases. Depending on whether quantization errors ($|x - \hat{x}|_p$) are taken into account during the training phase, we can distinguish between QAT and PTQ.

6.1.1 QAT. QAT was initially introduced to mitigate the potential accuracy loss associated with quantization. This technique seamlessly integrates quantization into the training cycle itself, allowing the model to adapt and learn to minimize quantization error, defined as:

$$e = |x - \hat{x}|_p \quad (12)$$

where $|\cdot|_p$ represents the p-norm, quantifying the difference between the original value x and its quantized counterpart \hat{x} . By accounting for quantization effects during both forward and backward passes, QAT enables the model to adjust its parameters and learn representations that are robust to the lower precision representation.

6.1.2 PTQ. As an alternative to QAT, PTQ is often preferred due to its straightforward implementation. Applied after the model training process is completed, PTQ involves quantizing the weights and activations of a pre-trained model. Although easier to implement, PTQ may not achieve the same level of accuracy as QAT, as the model is not retrained to adapt to the quantization process.

6.1.3 PTQ and QAT of OD models. Recent studies have demonstrated the effectiveness of quantization techniques in optimizing OD for various applications. For instance, Jacob *et al.* [60] applied quantization to both weights and activations in OD models, using 8-bit integers to achieve efficient inference. Their evaluation on the COCO dataset showed that the quantized SSD-MobileNet model retained an AP50:90 comparable to its floating-point counterpart, with only a slight decrease in accuracy. Furthermore, the quantized model significantly reduced latency, making it well-suited for real-time OD applications on resource-constrained devices. Similarly, Hinami *et al.* [55] introduced classifier adaptive quantization (CAQ) within a large-scale R-CNN framework to optimize OD at scale, particularly under limited computational resources. By integrating residual vector quantization (RVQ) and inverted indexing, their approach

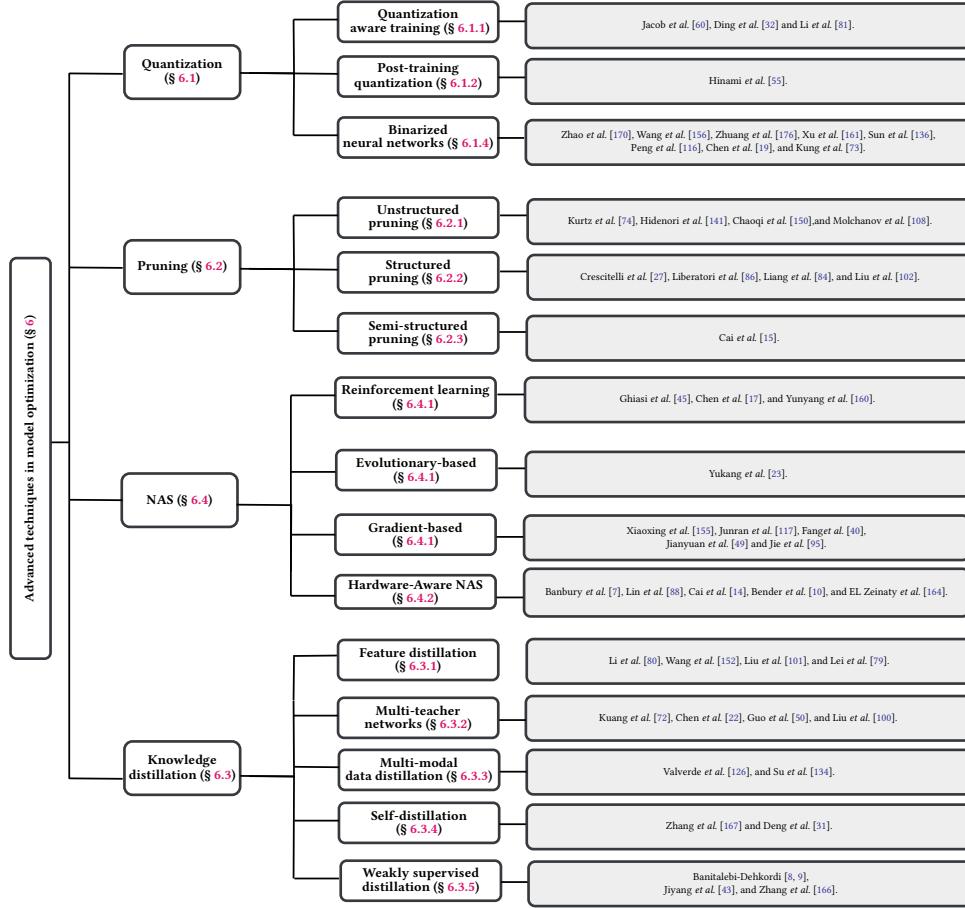


Fig. 3. Taxonomy of model compression methods.

achieved notable speed-ups and memory reductions while maintaining high detection precision, this study underscores the potential of adaptive quantization techniques. Additionally, Ding *et al.* [32] proposed the REQ-YOLO framework, which utilizes heterogeneous weight quantization and the alternative direction method of multipliers (ADMM) for field programmable gate array (FPGA)-based OD. This method exploits block-circulant matrices to improve weight compression and storage efficiency, highlighting how specialized hardware and advanced quantization can be combined to optimize OD performance, reducing both computational load and latency. Moreover, Li *et al.* [81] developed a fully quantized network (FQN) tailored for OD, leveraging low-bit arithmetic for enhanced computational efficiency. Their method quantizes both network weights and activations and was evaluated using a 4-bit RetinaNet detector [91] with a MobileNetV2 backbone [130]. The results showed only a 2.0% AP50:95 loss compared to the full 32-bit floating-point model, showcasing the practicality and effectiveness of low-bit models in constrained environments.

6.1.4 Binarized Neural Networks for Object Detection. BNNs represent an extreme form of quantization, where both weights and activations are constrained to binary values (-1 or 1). This binarization process can be mathematically

expressed using the sign function:

$$B(x) = \text{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \quad (13)$$

To achieve binary activations, the sign function converts floating-point values to their binary counterparts. However, this leads to uniformly zero gradients during training, posing challenges in training BNNs effectively. The straight-through estimator (STE) technique, introduced by Bengio *et al.* [11], is employed to facilitate effective training by approximating the gradient of the sign function.

6.1.5 Binarized OD models. Recent research in BNNs for OD demonstrates their potential in various applications. Peng *et al.* [116] propose a greedy layer-wise training method for BNNs, significantly boosting performance in detection tasks. Wang *et al.* [156] introduce a method called BiDet to enhance detection precision in BNNs through redundancy removal. Another study [136] presents a fast OD algorithm using binary deep CNNs, achieving rapid detection with minimal precision loss. Further, innovative methods like Group-Net [176] and DA-BNN [170] have been proposed to enhance feature representation capacity in BNNs. Further, BNAS [19] introduces binarized neural architecture search to produce highly compressed models suitable for edge computing. Kung *et al.* [73] and Xu *et al.* [161] explore the application of BNNs in infrared human detection and autonomous driving, respectively, showcasing the benefits of BNNs in terms of computational efficiency and memory usage. These studies collectively highlight the advancements and diverse applications of BNNs in OD, underscoring their efficiency and potential in resource-constrained environments.

6.2 Network Pruning

Pruning, in the context of OD models, is a strategic approach for reducing the model's size and computational requirements. This is achieved by strategically removing or "trimming" weight parameters based on predetermined criteria. Consider a deep CNN comprising L_n layers, where convolutional (CONV) layers are typically the most computationally intensive. Each layer within such a network consists of K_n kernels (or filters) and W_n non-zero weights. Consequently, the computational cost incurred during inference is directly proportional to $W_n \times K_n \times L_n$. As DL models grow in complexity, with increasing numbers of layers and parameters, this computational cost escalates significantly.

By implementing parameter pruning, sparsity is introduced into the model, effectively reducing the number of non-zero weights W_n . Similarly, kernel pruning diminishes the number of kernels K_n . These reductions in both parameters and kernels lead to a decreased overall computational burden. Modern computing platforms utilize software compression techniques that efficiently compress input and weight matrices, specifically targeting the zero-valued (pruned) parameters, allowing them to be bypassed during execution. Alternatively, specialized hardware [137] can directly execute these skipping operations, further enhancing efficiency. Existing pruning methodologies can be broadly classified into three categories: unstructured, structured, and semi-structured (or pattern-based) pruning, each offering distinct advantages for model optimization.

6.2.1 Unstructured Pruning. In unstructured pruning, weights are selectively removed from the network to minimize the impact on the loss function while preserving model accuracy. Several schemes have been developed for this purpose, including, weight magnitude pruning, gradient magnitude pruning, synaptic flow pruning, and second-order derivative pruning. The weight magnitude pruning approach zeroes out weights whose absolute values fall below a predefined threshold [74]. Gradient magnitude pruning method targets weights with minimal gradients, as these weights are assumed to have less impact on the final output [108], while synaptic flow pruning attractively eliminates weights

based on a global score that combines information from both weight magnitudes and their contribution to the loss function [141]. Finally, second-order derivative pruning aims to maintain network loss close to its original value by selectively zeroing out weights based on their second-order derivatives [150]. Each of these unstructured pruning techniques offers unique advantages in terms of balancing model size reduction with accuracy preservation.

6.2.2 Structured Pruning. Structured pruning enhances both model sparsity and uniformity by removing entire filters [27] or channels [102], resulting in a more streamlined and efficient model. This uniformity not only reduces the number of MAC operations, offering an advantage over unstructured pruning, but also facilitates compatibility with hardware acceleration technologies like TensorRT [61]. However, it is crucial to acknowledge that this structured approach may potentially compromise model accuracy, as it could inadvertently eliminate significant weights along with redundant ones. Nonetheless, the uniform structure of weight matrices resulting from structured pruning optimizes hardware acceleration, leading to improved memory and bandwidth utilization across various platforms.

6.2.3 Semi-Structured Pruning. Semi-structured pruning, also known as pattern pruning, merges the elements of both structured and unstructured pruning. It uses kernel patterns as masks to selectively preserve weights within a kernel, thereby creating partial sparsity. The efficacy of these patterns is often assessed through metrics like the L2 norm, aiding in the identification of the most efficient masks for inference. However, this pruning method inherently induces less sparsity compared to other types due to the fixed weight reduction within kernels [15]. To enhance sparsity further, it can be combined with connectivity pruning, which fully prunes specific kernels. Although this combined approach is effective, it might overlook redundant weights in 1×1 kernels, as it typically targets larger kernels for more substantial pruning. Despite potential accuracy reductions stemming from the removal of critical weights, the semi-structured nature of kernel pattern pruning facilitates hardware parallelism, thereby enhancing inference speed.

6.2.4 Pruning of OD models. Pruning has proven to be highly effective in optimizing OD models for deployment on resource-constrained devices, as demonstrated in several recent studies. For example, Liberatori *et al.* [86] utilized structured pruning techniques, specifically filter pruning, to significantly enhance the performance of a YOLO model for face mask detection on low-end hardware. By pruning CONV filters with low L1-norm values, they achieved a 50% reduction in the number of parameters, which led to nearly doubling the FPS performance on a raspberry pi-4, while maintaining only a moderate decrease of around 7% in mAP. Similarly, the Edge YOLO framework [84] incorporates structured pruning to streamline YOLO-based models, achieving a parameter reduction of approximately 30% and enhancing computational efficiency without compromising detection accuracy, demonstrating a slight improvement in AP50:95 compared to the full model. Furthermore, DeepCham [78] utilizes pruning at the image level by selectively processing and discarding less relevant image data before feeding it into the OD pipeline. This approach reduces the computational load on edge devices and enhances the adaptability of embedded AI systems in various environmental contexts. By efficiently managing which data is processed, such image-level pruning can significantly benefit embedded AI by minimizing power consumption and improving response times in real-time applications.

6.3 Knowledge Distillation (KD)

KD is a well-established technique in the field of DL, where a smaller, more efficient student model is trained to mimic the behavior of a larger, more complex teacher model. The teacher model's knowledge is transferred to the student model through various loss functions, enabling the student to achieve competitive performance with reduced computational resources. FitNets [127] pioneered the extension of KD by introducing *hint-based* learning, whereby the student model

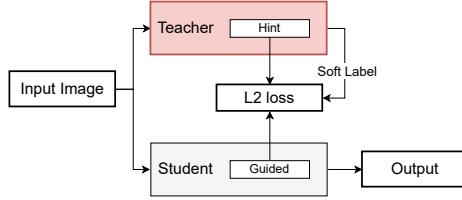


Fig. 4. General hint based KD framework.

is trained not only to mimic the output logits of the teacher but also its intermediate feature representations, often referred to as *hints*. This method has markedly improved the training of student models, particularly in classification tasks, by enabling the student to leverage the richer information embedded within the teacher's internal layers. The fundamental architecture of this approach is depicted in Fig. 4, where the teacher network includes a hint layer, and the student network has a corresponding guided layer. The outputs from these layers, along with the soft labels provided by the teacher network, are combined to form the loss function that optimizes the student model. While this technique had demonstrated success in classification tasks, its application to OD was first explored in the seminal work by Chen *et al.* [18]. In this pioneering approach, the authors extended KD to OD by incorporating three key components into the loss function, as expressed in (14).

$$L = \alpha L_{\text{det}} + \beta L_{\text{dis}} + \gamma L_{\text{hint}}, \quad (14)$$

where L_{det} represents detection-specific losses, which include both classification loss and bounding box regression loss. This component ensures the model's predictions are accurate in both object classification and localization. L_{dis} denotes the distillation loss, where the student model is trained to mimic the soft labels (logits) produced by the teacher model and L_{hint} refers to the hint-based feature distillation loss.

This combination of losses allows the student model to leverage the teacher model's expertise in both recognizing objects and localizing them within an image. The introduction of these components marked a significant advancement in the use of KD for more complex tasks such as OD. To further improve the effectiveness of this framework, a variety of advanced distillation modules, mechanisms, and strategies have been developed. These include feature distillation [79, 80, 101, 152], multi-teacher networks [22, 50, 72, 100], multi-modal data distillation [71, 126, 134], self-distillation [31, 167], and weakly supervised OD networks [8, 9, 43, 166], each offering unique benefits and improvements.

6.3.1 Feature Distillation. Although the aforementioned work extended KD to OD, the approach primarily focused on classification and soft-label distillation, with the hint-based loss playing a supporting role. However, OD requires precise localization in addition to accurate classification, which necessitates a more targeted application of feature distillation. Li *et al.* [80] propose adapting feature distillation specifically to address the localization challenges inherent in OD. Their method, termed "mimicking," involves training the student model to replicate not only the teacher's output or intermediate features in a general sense but, crucially, to focus on the region proposal features that are pivotal for OD. This approach was rigorously tested on the Faster R-CNN [124] OD framework, demonstrating substantial improvements in localization accuracy. Furthermore, the authors suggested that this methodology could be effectively applied to one-stage detectors like SSD [99], broadening its applicability across diverse OD architectures. This adaptation of the KD loss function, focusing on feature distillation for localization, marked a pivotal shift from the earlier FitNets approach. Their work underscored that for OD tasks, where both classification and precise localization are paramount, a more targeted feature distillation strategy could yield significantly enhanced performance. While this formulation serves

as a robust foundation, recent literature has presented various innovative adaptations to further refine the application of feature distillation to OD. For example, Wang *et al.* [152] introduce a cross-head knowledge distillation approach that specifically focuses on aligning predictions between the teacher and student models by sharing intermediate features between their detection heads. This method effectively addresses inconsistencies in predictions, ensuring that the student model captures both high-level semantic information and fine-grained details, thereby improving detection accuracy, particularly for small objects. Similarly, Liu *et al.* [101] extend this concept by proposing an attention-guided distillation technique, ensuring that the student network focuses on the most informative regions of the teacher’s feature maps. This approach optimizes both accuracy and efficiency in complex detection scenarios. In another notable development, Li *et al.* [79] present a hierarchical feature mimicking strategy that leverages multi-level feature distillation, enabling the student model to better understand spatial and semantic characteristics across different scales and contexts. Additionally, the authors introduce a novel approach by distilling knowledge across different heads of OD networks. This cross-head distillation technique allows the student model to more effectively leverage the diverse knowledge encoded in multiple detection heads, thereby enhancing both classification and localization performance.

6.3.2 Multi-Teacher Networks. Multi-teacher networks represent an advancement in KD, incorporating multiple teacher models, each contributing unique knowledge representations. This approach leverages the collective expertise of the teacher ensemble to provide the student network with a richer and more diverse set of features and decision boundaries. By distilling knowledge from these multiple sources, the student network gains a more comprehensive understanding of the OD task, leading to improved generalization across varied data distributions.

Chen *et al.* [22] showcase the application of multi-teacher networks in OD by introducing an enhanced knowledge distillation framework. In their approach, multiple teacher models are first trained to achieve high accuracy on the original dataset. Subsequently, a student model, structurally distinct from the teacher models in its backbone layer but sharing the same output layer, is trained using both soft targets generated by the teacher models and hard targets derived from the original dataset. This strategy enables the student model to benefit from the deep, specialized knowledge encapsulated within the multiple teachers while also ensuring alignment with the ground-truth labels. Furthermore, a weighting mechanism is incorporated, whereby the heatmap representing the size and offset for each object category is adjusted based on the classification accuracy of each teacher model. This refinement further optimizes the distillation process, facilitating more effective knowledge transfer. A similar multi-teacher approach was adopted by Kuang *et al.* [72], who designed a knowledge distillation framework tailored for OD using CenterNet [34] as the base architecture. In this approach, multiple teacher models are used to enhance the student model’s learning by providing soft targets derived from the heatmaps generated by the teachers. These heatmaps are weighted and fused according to each teacher model’s classification accuracy, ensuring a balanced knowledge transfer.

Alternatively, a single teacher model can guide multiple student models concurrently. Liu *et al.* [100] investigate this approach in the context of siamese visual tracking, where a single, well-trained teacher model distills its knowledge into several student models. Each student model is designed to specialize in a particular facet of the tracking task, such as robustness to appearance changes or precise localization. By training these student models in parallel and encouraging them to focus on distinct strengths, the overall tracking performance is substantially enhanced. This method also facilitates the selection of the most suitable student model for deployment, tailored to the specific requirements of the task at hand.

Finally, drawing inspiration from collaborative learning, an approach where students guide each other has also been investigated. In this framework, while a teacher model provides initial guidance, the student models are also designed to

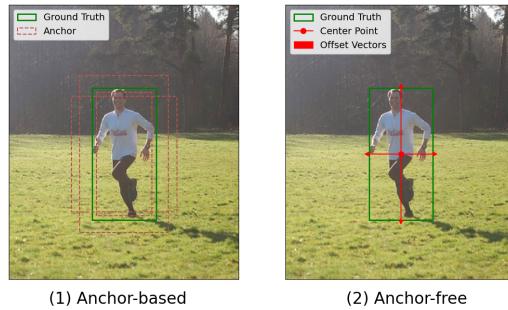


Fig. 5. Comparison between anchor-based and anchor-free detection methods. In both methods, the green box is the ground-truth. In the anchor-based method (1), the red dashed boxes represent predefined anchors that guide the object detection, with no center point shown. In the anchor-free method (2), the red arrows indicate the offset distances from the predicted center point (red dot) to the edges of the bounding box. The bounding box is directly regressed in the anchor-free method, without using predefined anchors.

learn from one another, sharing knowledge to mutually enhance their performance. Specifically, the shared knowledge distillation (shared-KD) method [50] exemplifies this concept by enabling student models to engage in cross-layer distillation within themselves. In this approach, student models not only learn from the teacher but also mimic and refine features across their own layers, effectively bridging the semantic gap between the teacher and student networks. The experimental results presented in the study demonstrated that shared-KD consistently improves detection performance across a diverse range of OD architectures, including two-stage, one-stage, and anchor-free detectors (Fig. 5). Moreover, this method significantly outperforms traditional knowledge distillation techniques, establishing it as a robust and efficient solution for real-time OD tasks.

6.3.3 Multi-Modal Data Distillation Networks. Multi-modal data distillation networks extend the traditional KD framework by incorporating and distilling knowledge from multiple data modalities. In contrast to unimodal approaches, where the student network learns exclusively from a single type of data (e.g., images), multi-modal data distillation leverages additional modalities such as text, depth, or audio, in conjunction with visual data. This integration of diverse data sources empowers the student network to develop a more holistic understanding of the OD task, capturing richer and more nuanced information that may not be fully represented by any single modality in isolation.

For instance, [126] applied multi-modal data distillation to OD and tracking by distilling knowledge from multiple teacher networks, each trained on a distinct modality (e.g., RGB, depth, and thermal images). The student network, distilled from these diverse sources, learns to effectively integrate the complementary strengths of each modality. This enables the student network to perform robust OD even under challenging conditions where single modality data might be unreliable, such as in scenarios with poor lighting or complex backgrounds. In a similar vein, [134] demonstrated a real-time on-road risk detection framework that utilizes multi-modal data distillation to enhance detection accuracy on mobile devices. Their approach leverages both RGB and depth images to train the teacher model, which is then distilled into a smaller, more efficient student model. This multi-modal distillation process not only improves detection performance but also ensures that the resulting model is sufficiently lightweight for deployment on resource-constrained platforms, such as mobility scooters equipped with Intel Neural Compute Stick 2 (NCS-2).

6.3.4 Self-Distillation. Self-distillation is a sophisticated technique in which a NN functions as both the teacher and the student, essentially distilling knowledge from its own internal representations. This process can be implemented either across different layers within the same model or over successive iterations of the model’s training. In the context

of OD, self-distillation serves to refine the model's internal representations and decision-making processes, thereby enhancing its capability to accurately detect and localize objects.

Peizhen *et al.* [167] are the pioneers in applying self-distillation to OD, introducing the label-guided self-distillation (LGD) framework. LGD eliminates the need for a separate, pre-trained teacher model. Instead, the model guides itself using label-encoded information to generate object-wise descriptors that capture both appearance and spatial characteristics. These descriptors are subsequently utilized to model inter-object relationships through a cross-attention mechanism, facilitating the network's refinement of its internal feature representations. The LGD framework also incorporates an intra-object knowledge mapper, which maps the refined features back onto the feature map, aligning them with the spatial and geometric properties of the detected objects. This self-guided distillation process enables the model to continually enhance its detection capabilities by learning from its own predictions, offering a resource-efficient approach particularly advantageous in scenarios with limited computational resources. Building upon this concept, a recent development in the field is the smooth and stepwise self-distillation (SSSD) framework [31]. SSSD further advances self-distillation by addressing certain limitations present in earlier methods, such as those employed in LGD. Notably, SSSD introduces the use of jensen shannon (JS) divergence [37] as a smoother and more robust alternative to the traditional mean squared error (MSE) loss. This proves particularly beneficial in the complex feature space encountered in OD tasks. In OD, where models must contend with diverse and noisy input data (e.g., occlusions, varying object scales, and cluttered backgrounds), JS divergence helps maintain stable training and improves the model's ability to generalize across different detection scenarios. Moreover, SSSD incorporates a stepwise adjustment of the distillation coefficient β in (14), based on the learning rate schedule. This dynamic adjustment is crucial for sustaining effective knowledge distillation throughout the training process. As the learning rate decreases in the later stages of training, the model approaches convergence, and adjusting β ensures that the distillation loss remains balanced with the task-specific loss. This prevents the model from prematurely halting its knowledge transfer from the teacher model, enabling continued refinement and avoiding overfitting in the final stages.

6.3.5 Weakly Supervised Object Detection Networks. Weakly supervised object detection (WSOD) networks are designed to function effectively with limited or incomplete annotations [43], a prevalent scenario in large-scale data collection where obtaining fully labeled datasets is challenging or costly. These networks leverage weak supervision signals, such as image-level labels or partial bounding box annotations, to train OD models capable of generalizing well even in the presence of sparse ground-truth data. The integration of weak supervision into KD frameworks enables the student network to learn from the available limited labeled data while simultaneously benefiting from the distilled knowledge imparted by the teacher network. This synergistic combination proves particularly potent in scenarios where acquiring fully annotated datasets is impractical or infeasible. By utilizing KD to transfer robust knowledge from the teacher to the student, WSOD networks can attain competitive performance, often approaching that of fully supervised models, while significantly reducing the reliance on extensive manual labeling efforts [58].

An illustration of this approach can be found in the work of Banitalebi-Dehkordi [8], who extends the concept of WSOD by leveraging unlabeled data in conjunction with KD techniques to train low-power OD models. In their methodology, a teacher model generates pseudo-labels from unlabeled data, which are subsequently utilized to train the student model in a weakly supervised fashion. This approach effectively decouples the distillation process from the necessity of extensive ground-truth data, making it particularly suitable for scenarios where labeled data is scarce or expensive to acquire. Moreover, this framework allows for the incorporation of multiple teacher models, each potentially specializing in different object classes or utilizing diverse architectures, thereby enriching the training of the student

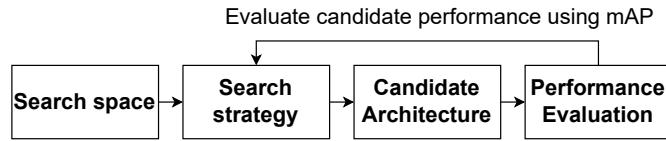


Fig. 6. Neural architecture search framework.

model. The utilization of unlabeled data not only broadens the model's applicability but also enhances its robustness by incorporating a wider array of knowledge sources. Recent advancements in unsupervised domain adaptation for OD have further underscored the significant potential of WSOD frameworks. By leveraging weak supervision in conjunction with domain adaptation techniques, these frameworks enable OD models to effectively generalize across disparate domains, such as those characterized by varying lighting conditions, weather changes, or sensor differences, without requiring annotated data in the target domain [166].

6.4 Neural Architecture Search (NAS)

NAS is a sophisticated process that aims to automate the design and configuration of NN architectures. In the field of OD, these architectures typically consist of three primary components: a backbone, a neck, and a head, as illustrated in Fig. 2. During the iterative process of NAS, the search strategy evaluates candidate architectures for each component (backbone, neck, or head) based on performance metrics such as mAP. This feedback loop refines the search strategy, guiding it towards increasingly promising architectures. This automated approach significantly accelerates the design of efficient and effective OD models. The core challenge in NAS lies in defining the appropriate search space and identifying the optimal multi-objective search strategies to explore this space efficiently. As illustrated in Fig. 6, the general NAS framework involves an iterative process where the search space, which can focus on potential architectures for either the backbone, neck, or head, is explored through various search strategies. These strategies encompass different methodologies, including RL and EA, both of which played a prominent role in early NAS research [17, 23, 45, 160]. While effective, these traditional methods often proved computationally expensive, frequently necessitating extensive computational resources over many graphics processing unit (GPU) days to identify optimal architectures.

To address this efficiency concern, differentiable NAS [40, 49, 95, 117, 155] was proposed, leveraging stochastic gradient descent to guide the search process. This approach diverges from earlier methods by transforming the inherently discrete and non-differentiable search space into a continuous one. This continuous relaxation facilitates the application of gradient-based optimization, rendering the search process more efficient.

Beyond optimizing for accuracy, recent advancements in NAS emphasize multi-objective search, where additional constraints such as computational cost, model size, and energy efficiency are incorporated into the optimization process. This shift is particularly relevant in the context of resource-constrained environments, leading to the development of hardware aware neural architecture search (HNAS). Unlike conventional methods, HNAS explicitly integrates deployment constraints such as latency, power consumption, and memory footprint into the search framework, ensuring that the generated architectures are not only performant but also feasible for real-world deployment. Table 6 presents a comparison of OD models designed using NAS, evaluated on the COCO dataset. The table summarizes key aspects such as the search methods employed within the multi-objective search framework (RL, EA, and gradient-based approaches), the specific architectural components optimized, performance (AP50:95), and the associated search costs. The following sections provide a structured discussion of these search strategies, followed by an in-depth exploration of HNAS, which extends NAS methodologies to account for real-world hardware constraints.

Table 6. Comparison of NAS-based OD models evaluated on the COCO dataset. An asterisk ("*") denotes TPU-days; all other values are in GPU days.

Method	Search	Optim. Parts	AP50:95↑	Search Cost↓
NAS-FCOS [153]	RL	Neck, Head	43.0	28
MobileDets [160]	RL	Backbone	28.5	-
NAS-FPN [45]	RL	Neck	44.2	333*
DetNAS [23]	EA	Backbone	42.0	44
EAutoDet-s [155]	Gradient	Backbone, Neck	40.1	1.4
EAutoDet-m [155]	Gradient	Backbone, Neck	45.2	2.2
EAutoDet-L [155]	Gradient	Backbone, Neck	47.9	4.5
EAutoDet-X [155]	Gradient	Backbone, Neck	49.2	22.0
ResNet-101 (NATS) [117]	Gradient	Backbone	40.4	20.0
ResNeXt-101 (NATS) [117]	Gradient	Backbone	41.6	20.0
FNA++ (RetinaNet) [40]	Gradient	Backbone	34.7	8.5
FNA++ (SSDLite) [40]	Gradient	Backbone	23.9	21.0
Hit-Detector [49]	Gradient	Backbone, Neck, Head	41.4	-
IC-ResNet50 [95]	Gradient	Backbone, Neck, Head	39.2	-

6.4.1 Multi-Objective Search Strategy.

Reinforcement Learning-based NAS. RL-based NAS employs a controller, typically a recurrent neural network (RNN), to sample architecture configurations from a predefined search space [46]. The controller is trained to maximize the expected validation accuracy of the generated architectures, utilizing this accuracy as the reward signal. This process is formalized through a policy gradient method, where the controller's policy $\pi(a|s; \theta)$ generates a probability distribution over possible actions a (i.e., architecture configurations) given the current state s (representing previous architecture decisions). The objective function $J(\theta)$ for the controller, parameterized by θ , defined as the expected reward, is formulated in (15).

$$J(\theta) = \mathbb{E}_{\pi(a|s;\theta)} [R(s, a)], \quad (15)$$

where $R(s, a)$ represents the reward function, which is typically the model's performance on a validation dataset in the context of NAS for OD.

The policy gradient theorem provides a methodology for computing the gradient of $J(\theta)$ as follows:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi(a|s;\theta)} [\nabla_{\theta} \log \pi(a|s; \theta) R(s, a)]. \quad (16)$$

In practice, a RL algorithm such as REINFORCE [138] can be employed to estimate the gradient expressed in (17).

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi(a^{(i)}|s^{(i)}; \theta) R(s^{(i)}, a^{(i)}), \quad (17)$$

where N is the number of architectures sampled in a batch, and $(s^{(i)}, a^{(i)})$ represents the i -th sampled state-action pair.

Nevertheless, (17) can exhibit high variance in the gradient estimation, potentially leading to instability in the learning process and hindering convergence. To mitigate this issue, a baseline $b(s^{(i)})$ is commonly introduced. This baseline, often implemented as the running average of the rewards or an estimated value function, serves to minimize the variance of the gradient by centering the rewards. Although the baseline does not affect the bias of the gradient estimate, it helps promote stability and enhances the efficiency of the training process. Incorporating this baseline subtraction, the gradient estimation equation is modified as follows:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi(a^{(i)} | s^{(i)}; \theta) \left(R(s^{(i)}, a^{(i)}) - b(s^{(i)}) \right), \quad (18)$$

Furthermore, efficient neural architecture search via parameter sharing (ENAS) was developed to address the computational challenges inherent in traditional RL-based NAS. ENAS achieves this efficiency by introducing a technique known as parameter sharing among the candidate architectures sampled during the search process. Specifically, ENAS constructs a single, over-parameterized model referred to as a *supernet*, which encompasses all possible architectures within the predefined search space. Rather than training each sampled architecture independently, ENAS allows these architectures to inherit weights from the supernet. When the controller selects a candidate architecture, it is treated as a subgraph within the supernet, inheriting its corresponding weights rather than starting from random initialization. These shared weights are then updated across multiple architecture evaluations, enabling new architectures to benefit from the training of previously sampled ones.

A notable example of RL-based NAS is demonstrated in the NAS-FCOS framework [153], where a long short-term memory (LSTM)-based controller predicts the complete OD architecture. As shown in Table 6, NAS-FCOS optimizes the neck and head components (FPN structure) and achieves a high AP50:95 of 43.0 with a search cost of 28 GPU days. This progressive search strategy focuses on optimizing the FPN structure and the prediction head separately, leading to significant reductions in computational resources and time. NAS-FCOS further speeds up training by fixing the backbone network and caching its pre-computed outputs, enabling the controller to concentrate solely on optimizing the FPN and prediction head configurations. Additionally, to improve the reward signal's effectiveness during early training, NAS-FCOS replaces the mAP with a negative loss sum as the reward. This approach stabilizes the controller's training and accelerates convergence. Overall, this application showcases the power of RL-based NAS in discovering high-performing OD architectures that are both computationally efficient and accurate.

Another application of the RL-based NAS strategy can be found in the MobileDets framework [160]. This framework extends the principles of RL-based NAS to optimize the backbone architecture specifically for OD applications on mobile devices. As indicated in Table 6, MobileDets achieves AP50:95 of 28.5. While the exact search cost is not explicitly stated, the framework's utilization of the TuNAS algorithm [10], which builds upon the concept of ENAS, combines RL with a parameter sharing approach to efficiently search for architectures that satisfy both accuracy and latency requirements on mobile hardware. In this framework, the controller explores a search space tailored to the computational constraints of mobile devices. The key innovation in MobileDets lies in its latency-aware design, where the reward function is adapted to incorporate latency measurements along with traditional accuracy metrics. By leveraging a supernet that encompasses all potential backbone architectures, MobileDets circumvents the need to fully retrain each sampled architecture from scratch, thereby optimizing search efficiency.

Building upon these principles, the NAS-FPN framework [45] further shows the efficacy of RL-based NAS in refining the neck component of OD models. As demonstrated in Table 6, NAS-FPN achieves a state-of-the-art AP50:95 of 44.2, albeit at a search cost of 333 tensor processing units (TPU) days. In contrast to traditional manually designed FPNs, which rely on fixed cross-scale connections, NAS-FPN automates the discovery of the optimal feature pyramid architecture. In NAS-FPN, the RL-based controller explores a vast search space of potential cross-scale connections, enabling the identification of more effective and scalable FPN structures. The search process culminates in the discovery of an optimal merging cell, which is then recursively applied to construct a deep and flexible FPN. These results underscore the capability of RL-based NAS methodologies to optimize not only the backbone, as demonstrated in MobileDets, but

also critical components such as the neck, ultimately leading to the development of more powerful and efficient OD models.

Evolutionary Algorithm-based-NAS. EAs apply principles inspired by biological evolution, such as selection, mutation, and crossover, to the task of optimizing NN architectures. Within the context of NAS, EAs are employed to evolve a population of candidate architectures over successive generations, with the aim of maximizing a fitness function, typically represented by the model's accuracy on a validation dataset. The evolutionary process starts with an initial population of randomly generated architectures. In each generation, the fitness of every individual within the population, corresponding to a specific network architecture, is evaluated. The fitness function F for OD models can be defined as:

$$F(m) = \text{mAP}(D_{\text{val}}, m) - \lambda C(m), \quad (19)$$

where m represents a model architecture within the population, $\text{mAP}(D_{\text{val}}, m)$ denotes the mAP achieved on the validation dataset D_{val} , and $C(m)$ quantifies the computational cost associated with the model, typically measured in terms of the number of parameters or FLOP. The hyperparameter λ serves as a regularization term, controlling the trade-off between accuracy and computational cost.

The selection process favors selecting architectures with higher fitness scores to serve as parents for the subsequent generation. Crossover and mutation operators are then applied to these chosen architectures to generate offspring, forming the next generation of the population. Crossover involves combining elements from two parent architectures, while mutation introduces random modifications to an architecture's configuration. An example of a mutation operation could be mathematically expressed by (20).

$$m' = \text{mutate}(m, \mu), \quad (20)$$

where m' is the mutated architecture, m is the original architecture, and μ defines the mutation rate which controls the extent of changes applied.

After several generations, the population converges towards architectures that exhibit optimized trade-offs between detection performance and computational efficiency. In the domain of OD, EAs have been leveraged to discover innovative architectures that maintain high accuracy on benchmark datasets while remaining computationally feasible for deployment on embedded systems. A prominent example of applying EAs in OD is the DetNAS framework [23], which employs an EA to search for optimal backbone architectures specifically tailored for OD tasks. As shown in Table 6, DetNAS focuses exclusively on optimizing the backbone component, achieving an AP50:95 of 42.0 with a search cost of 44 GPU days. The search process in DetNAS begins with an initial population of randomly generated architectures that adhere to predefined computational constraints. Each architecture undergoes a two-step evaluation: first, it is evaluated on a small subset of the training set to update batch normalization statistics, and then it undergoes a full evaluation on a validation set to determine its detection performance. Based on these evaluations, the top-performing architectures are selected as parents for the next generation. New architectures are then generated through crossover and mutation operations. Crossover combines elements from two parent architectures, while mutation introduces random changes to the architectural configuration. This evolutionary process iterates over multiple generations, progressively refining the population toward architectures that achieve a balance between high detection accuracy and low computational cost. The results, reflected in Table 6, demonstrate that DetNAS effectively utilizes evolutionary strategies to identify backbone architectures that perform competitively in OD tasks, achieving a notable AP50:95 performance.

Gradient-Based NAS. Gradient-based NAS, specifically differentiable neural architecture search (DNAS), employs a continuous relaxation of the search space to enable efficient gradient-based optimization using methods such as stochastic gradient descent (SGD). This approach starts from earlier RL and EA-based methods by transforming the inherently discrete and non-differentiable search space into a continuous representation. This transformation allows for the application of gradient descent to guide the search process, enhancing efficiency by directly optimizing architecture parameters and weights.

The core of gradient-based NAS involves optimizing a supernet, which is a large NN that encompasses all possible architectures within a search space. Each possible architecture is represented as a subgraph of the supernet. This method allows for the shared use of parameters among all sub-networks, significantly reducing the computational cost compared to evaluating each architecture independently. By training this supernet, gradient-based optimization efficiently learns which architectural configurations perform best, guiding the search towards optimal designs. The optimization problem in this context can be formulated as a bi-level optimization problem:

$$\begin{aligned} & \min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha), \quad \text{with} \\ & w^*(\alpha) = \arg \min_w \mathcal{L}_{train}(w, \alpha), \end{aligned} \quad (21)$$

Here, \mathcal{L}_{train} represents the training loss, while \mathcal{L}_{val} denotes the validation loss. The architecture parameters α and the network weights w undergo a nested optimization process. Initially, for a given set of fixed architecture parameters α , the weights w are optimized to minimize the training loss:

$$w \leftarrow w - \eta \nabla_w \mathcal{L}_{train}(w, \alpha) \quad (22)$$

Subsequently, the architecture parameters α are updated based on the validation loss:

$$\alpha \leftarrow \alpha - \eta' \nabla_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \quad (23)$$

This optimization process enables the supernet to efficiently learn and update the weights and architectural parameters, facilitating the selection of optimal architectures within the search space. Building upon the foundational concept of the supernet, one-shot NAS further enhances this framework by evaluating the entire search space of architectures within a single training cycle [39]. While the supernet itself allows for shared weight training across multiple sub-networks, one-shot NAS extends this capability by enabling the evaluation of all potential architectures without the need for multiple, computationally expensive training iterations. In essence, once the supernet is trained, each architecture can be rapidly assessed by activating its corresponding sub-network within the supernet, leveraging the shared weights. This approach drastically accelerates the evaluation and selection of optimal architectures, circumventing the computationally intensive process of training each candidate network individually. Formally, let \mathcal{A} represent the set of all possible architectures within the supernet, and let W denote the weights of the supernet. The performance of a specific architecture $a \in \mathcal{A}$ is then evaluated using (24).

$$\text{Performance}(a) = \text{Evaluate}(a, W|_a) \quad (24)$$

where $W|_a$ represents the subset of supernet weights allocated to architecture a . By leveraging these shared weights, one-shot NAS provides a practical and scalable method to explore vast architecture spaces efficiently, facilitating the rapid development of high-performing NN architectures.

The adoption of gradient-based NAS has significantly advanced the field of OD, offering more efficient optimization of NN architectures. As shown in Table 6, the EAutoDet framework, proposed by Xiaoxing *et al.* [155], effectively integrates differentiable NAS and one-shot NAS techniques to enhance OD performance. EAutoDet employs a differentiable search strategy that transforms the search space into a continuous domain, enabling efficient optimization via gradient descent. This approach is coupled with a one-shot NAS methodology, where a supernet is constructed to encapsulate all possible architectures. Additionally, EAutoDet introduces kernel reusing and dynamic channel refinement techniques within this supernet, reducing memory requirements and computational costs. As indicated in Table 6, by focusing on optimizing both the backbone and neck components, EAutoDet achieves state-of-the-art results in OD with significantly lower search times and resource consumption compared to previous methods, requiring only 1.4 to 22 GPU days depending on the model variant.

Furthermore, the neural architecture transformation search (NATS) framework [117] offers another compelling example of the effectiveness of gradient-based NAS in OD, as reflected in Table 6. Unlike traditional approaches that design new architectures from scratch, NATS focuses on transforming existing networks, such as ResNet [54] and ResNeXt [159], to better suit the specific demands of OD tasks. It employs a gradient-based search strategy at the channel level, optimizing the dilation rates within convolutional layers, which primarily enhances the backbone's ability to detect objects at multiple scales without increasing computational complexity. By capitalizing on the pre-trained weights of the original networks, NATS efficiently adapts these models for OD, resulting in improved accuracy while maintaining a search cost of around 20 GPU days. The fast network adaptation (FNA++) method [40], as presented in Table 6, harnesses the versatility of gradient-based NAS to optimize OD frameworks. FNA++ builds upon the principles of differentiable NAS by employing a parameter remapping technique, allowing for the efficient adaptation of both the architecture and parameters of a pre-trained seed network, such as MobileNetV2, specifically for the task of OD. This approach, with a particular focus on optimizing the backbone component, eliminates the necessity for extensive retraining, thereby significantly reducing computational overhead. As demonstrated in Table 6, FNA++ achieves remarkable efficiency, requiring a search cost of only 8.5 GPU days for RetinaNet and 21 GPU days for SSDLite, while still delivering competitive AP50:95 scores.

Liu *et al.* [95] further showcase the potential of gradient-based NAS in enhancing OD capabilities through their inception convolution with efficient dilation search (ICEDS). This latter introduces a novel form of dilated convolution, referred to as inception convolution, designed to optimize the effective receptive field (RF) by independently varying dilation patterns along spatial axes, channels, and layers within the network. To efficiently navigate this complex search space, they propose the efficient dilation optimization (EDO) algorithm. EDO pre-trains a supernet encompassing all potential dilation patterns and then selects the optimal configuration for each CONV layer. While specific search costs are not explicitly reported in their work, ICEDS emphasizes reducing computational overhead compared to traditional NAS methods such as DARTS [93].

Finally, the Hit-Detector framework proposed by Guo *et al.* [49] offers a distinctive approach by simultaneously optimizing the backbone, neck, and head components of an object detector in an end-to-end fashion, as highlighted in Table 6. This hierarchical trinity architecture search framework ensures consistency across all components of the detection pipeline, addressing the challenge of inter-component alignment that other NAS methods might neglect. Hit-Detector introduces a multi-level search space, differentiated into sub-search spaces for each key component, and utilizes gradient-based optimization to efficiently navigate the design space. This holistic approach mitigates potential mismatches and suboptimal performance that can arise from independent component optimization, thereby enhancing

the overall coherence and effectiveness of the network design. Although precise search costs are not specified, the AP50:95 performance achieved by Hit-Detector underscores its efficacy in OD tasks.

6.4.2 Hardware-Aware NAS. Recent advancements in NAS have led to the emergence of hardware aware (HW) optimization techniques, where models are designed not only to maximize accuracy but also to meet stringent hardware constraints. Unlike conventional NAS methods, which focus primarily on improving task performance, HNAS integrates deployment constraints, such as inference latency, power consumption, memory footprint, and computational complexity, directly into the search process. This paradigm shift is particularly critical for resource-constrained environments, such as edge devices, mobile processors, and MCUs, where efficiency is just as important as accuracy.

HNAS methods typically incorporate hardware constraints at various stages of the search pipeline. In differentiable approaches, constraints such as latency and peak memory usage are integrated into the optimization objective, enabling architectures to be selected based on both performance and efficiency metrics. For example, Micronets [7] extends differentiable search techniques to ensure model feasibility on low-power devices by constraining activation sparsity and computational complexity. Similarly, MCUNet [88] adopts a two-stage search strategy: first, architectures are designed for optimal task performance, and then they are pruned and adapted to fit within MCU memory limitations. In RL-based frameworks, hardware awareness is incorporated through reward functions that penalize excessive latency or computational cost, as demonstrated in methods such as ProxylessNAS [14] and TuNAS [10], which dynamically profile candidate architectures on real hardware during the search process.

Recent work by EL Zeinaty et al. [164] introduced a framework that leverages large language models (LLMs) for NAS, incorporating a vision transformer (ViT)-based KD strategy and explainability. This framework optimizes accuracy, efficiency, and memory usage in three phases: (1) the search phase, where LLMs guide the exploration of architectures using Pareto optimization; (2) the full training phase, where the best candidate is fine-tuned; and (3) the KD phase, where a pre-trained ViT model distills logits into the student model. The framework reduces search time from 12.5 days to 1.5 days (worst-case 3.5 days), demonstrating efficiency in hardware-aware architecture design. Deployed on an STM32H7 MCU with a 320 KB SRAM constraint, the models show the potential of HNAS for embedded systems. This work advances HNAS for TinyML, paving the way for future research.

7 Optimizing Object Detection for TinyML

In the preceding section, various optimization techniques for OD, each tailored to specific computing environments, were explored. As depicted in Fig. 7, these environments can be broadly categorized into cloud/edge AI, mobile AI, and tiny AI, each characterized by distinct levels of computing power and memory resources. Cloud/edge and mobile platforms benefit from relatively abundant memory and storage capacities, with cloud/edge AI typically exceeding 16GB of memory and boasting several terabytes of storage, while mobile AI platforms commonly offer around 4GB of memory and upwards of 64GB of storage. In stark contrast, TinyML operates under far more stringent constraints, typically offering between 256KB and 512KB of memory, with storage capacities in the vicinity of 1MB. TinyML applications often run on MCUs, which are designed for low-power, low-cost scenarios [89]. Thus, optimizing OD models for TinyML requires not only reducing computational complexity but also ensuring that the models can fit within the restricted memory and processing capacities of MCU platforms. Given these extreme limitations, specialized optimization strategies are crucial for enabling TinyML applications. Incorporating the advanced optimization techniques discussed earlier, OD models can be effectively tailored for deployment in TinyML environments. Traditional models like YOLO, originally designed for real-time detection on GPU-class devices, require significant modifications to operate effectively

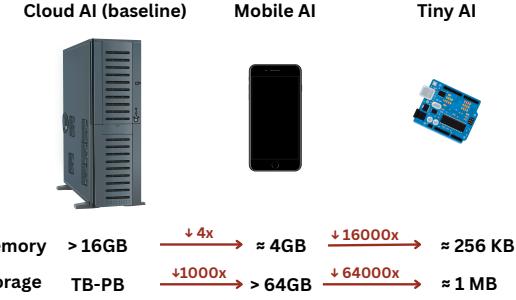


Fig. 7. Comparison of computational resource constraints across different AI platforms: cloud AI, mobile AI, and tiny AI.

Table 7. Categorized Summary of Optimization Techniques Used in MCU-Optimized Object Detection Models

Optimization Strategy	Model	Key Features
Neural Architecture Search (NAS)	MCUNetV1-YOLOV2 [88]	NAS with memory scheduling for MCUs; optimized memory layout reduces peak SRAM usage by 3.4x.
	MCUNetV2-YOLOV3 [87]	Use patch-based execution to reduce memory bottlenecks; in-place convolutions overwrite input activations, saving 1.6x memory.
Quantization-Based Optimization	TinyissimoYOLO [110]	Use fully quantized (8-bit integer) models to reduce size by 4x while maintaining accuracy.
	EtinyNet-SSD [162]	Adaptive per-layer quantization and depthwise convolutions to reduce computation.
Hardware-Aware Scaling (HAS)	XiNet-YOLOV7 [4]	Scale models based on RAM, flash, and compute resources; adapt to MCU memory and energy constraints.

within these constrained settings. Over the years, YOLO has evolved through various iterations, offering scaled-down versions such as 'nano', 'small', 'medium', and 'large' to cater to different computational needs. However, despite these scaled-down versions, the computational demands and memory requirements of YOLO models generally remain prohibitive for TinyML applications [118]. While substantial progress has been made in embedded mobile ML with models like MobileNetV1, ShuffleNet, MobileDenseNet, and Lite-YOLOv3, deploying DL on MCUs for TinyML presents even greater challenges. In TinyML applications, SRAM significantly restricts the size of activations (both read and write), and flash memory limits the model size, which is typically read-only. Additionally, MCUs typically operate at clock speeds ranging from 50 MHz to 500 MHz, which is considerably slower than processors found in typical laptops. Despite these constraints, model pruning has emerged as a crucial optimization strategy for reducing the complexity of OD models while maintaining accuracy. The DTMM framework [52] introduces a filter pruning technique that enables structured yet fine-grained compression, significantly reducing inference latency without compromising performance, making it particularly well-suited for TinyML OD. Additionally, depth pruning with auxiliary networks [30] offers another promising approach, where entire layers are pruned instead of individual filters, and auxiliary networks compensate for any loss in accuracy. These methods enhance computational efficiency, allowing MCUs to run more sophisticated OD models with minimal performance degradation. Beyond computational constraints, TinyML OD models also face challenges related to limited training data, which can lead to overfitting and poor generalization. To address this issue, *dataset distillation* [1] has been proposed as a technique to enable low-memory, on-device training. By synthesizing highly compact yet informative datasets, dataset distillation allows models to be efficiently trained directly on MCUs, reducing reliance on large-scale datasets and cloud-based retraining. This technique is especially beneficial for IoT-based OD, where connectivity constraints make frequent cloud updates impractical. Innovations like TinyTracker [12] and the faster objects more objects (FOMO) architecture [59] exemplify cutting-edge approaches being developed to extend the capabilities of TinyML in OD. TinyTracker utilizes AI "in-sensor" technology, enabling efficient processing directly at the sensor level, thus reducing the need for data transmission and significantly lowering

power consumption. Similarly, FOMO, introduced by Edge Impulse, is designed to detect and track multiple objects in real-time on highly resource-constrained devices, achieving remarkable efficiency through optimized NN architectures and lightweight inference techniques. While TinyTracker and FOMO showcase the ongoing innovation in TinyML, other models have been benchmarked to formally demonstrate their effectiveness. Table 7 provides a comprehensive overview of the optimization techniques used in MCU-optimized OD models, categorizing key strategies such as NAS, quantization, memory-aware scheduling, and hardware-aware scaling. These techniques serve as the foundation for adapting OD architectures to TinyML, effectively addressing the trade-offs between computational efficiency, memory usage, and detection performance under severe resource constraints. As shown in Table 8, various MCU-optimized OD models have been evaluated on the PASCAL VOC benchmark. The table provides a comparative analysis of these models, highlighting their optimization techniques, computational complexity (in terms of MMACs), memory usage, and performance (mAP). The results demonstrate a wide range of trade-offs between model complexity, resource consumption, and detection accuracy. These trade-offs are further illustrated in Fig. 8, which depicts the relationship between MMACs, mAP and the number of parameters for each model. In the following sections, we analyze these solutions in detail, highlighting how they address the challenges inherent in deploying OD in TinyML applications, while maintaining efficient, real-time performance under stringent power and memory constraints.

MCUNet-YOLO. The development of MCUNetV1-YOLOv2 [88] is grounded in the MCUNetV1 framework, which integrates TinyNAS and TinyEngine to optimize DL models for deployment on resource-constrained MCUs. TinyNAS, a two-stage NAS method, begins by refining a specialized factorized hierarchical search space designed to meet the stringent resource constraints of mobile and embedded platforms [140]. This search space structures the NN into configurable blocks, allowing for various options such as different CONV types (regular, depthwise separable, mobile inverted bottleneck), kernel sizes (3×3 , 5×5), squeeze-and-excitation ratios, skip connections, filter sizes, and varying numbers of layers. TinyNAS optimizes this search space specifically for TinyML applications, adjusting parameters such as input resolution $R = \{48, 64, 80, \dots, 192, 208, 224\}$ and width multiplier $W = \{0.2, 0.3, 0.4, \dots, 1.0\}$ to cover a wide spectrum of resource constraints. This approach results in $12 \times 9 = 108$ possible search space configurations, $S = W \times R$. Each search space configuration contains 3.3×10^{25} possible sub-networks. To efficiently identify the best models within this vast search space, TinyNAS eschews exhaustive searches by employing a one-shot neural architecture search strategy. In this approach, a supernet encompassing all possible sub-networks is trained using weight sharing, enabling the rapid estimation of each sub-network's performance. Subsequently, an evolutionary search algorithm is applied to select the optimal model that strikes the best balance between accuracy and resource constraints. A key component of MCUNetV1 process is TinyEngine, a memory-efficient inference library co-designed with TinyNAS. TinyEngine is utilized during the evaluation of each sampled network to optimize memory scheduling and measure optimal memory usage. Unlike traditional inference libraries that optimize memory scheduling layer by layer, TinyEngine optimizes memory usage based on the entire network topology, achieving up to a $3.4 \times$ reduction in memory consumption and a $1.7\text{-}3.3 \times$ increase in inference speed. This tight integration between TinyNAS and TinyEngine allows for the exploration of larger search spaces and the fitting of more complex models into the limited memory of MCUs, ensuring efficient and high-performance execution. The MCUNetV1 framework has demonstrated state-of-the-art performance on ImageNet, achieving a top-1 accuracy of 70.7% on off-the-shelf MCUs. Building upon these promising classification results, the MCUNet model was then employed as a backbone for a YOLOv2 detector to assess its effectiveness in OD tasks. Through this integration, MCUNetV1-YOLOv2 achieved a notable mAP score of 51.4% on the PASCAL VOC dataset.

Table 8. Comparative analysis of MCU-optimized OD models using 8-bit precision for weights and activations on the VOC benchmark.

Model	Venue (Year)	Optimization	Input Size	Parameters↓	MCU	MMACs↓	Peak SRAM↓	mAP↑
MCUNetV1-YOLOV2 [88]	NeurIPS (2020)	NAS	224x224	1.20M	STM32H743	168	466kB	51.4%
MCUNetV2-YOLOV3-M4 [87]	NeurIPS (2021)	NAS	224x224	0.47M	STM32F412	172	247kB	64.6%
MCUNetV2-YOLOV3-H7 [87]	NeurIPS (2021)	NAS	224x224	0.67M	STM32H743	343	438kB	68.3%
EtinyNet-SSD [162]	AAAI (2022)	Quantization (ASQ)	256x256	0.59M	STM32H743	164	395kB	56.4%
XiNet-YOLOV7-S [4]	ICCV (2023)	Model Design (HAS)	256x256	-	STM32H743	80	53kB	54.0%
XiNet-YOLOV7-M [4]	ICCV (2023)	Model Design (HAS)	384x384	-	STM32H743	220	138kB	67.0%
XiNet-YOLOV7-L [4]	ICCV (2023)	Model Design (HAS)	416x416	-	STM32H743	789	511kB	74.9%
TinyissimoYOLO [110]	IEEE Access (2024)	Quantization (QAT)	112x112	0.70M	GAP9	55	-	56.4%

Note: Peak SRAM indicates the maximum on-chip SRAM usage during inference, expressed in kilobytes.

Building upon the foundation established by the MCUNetV1 framework, MCUNetV2 [87] introduces several key innovations to further optimize OD on MCUs. A notable enhancement is the implementation of in-place depth-wise convolution, which substantially reduces peak memory usage by a factor of 1.6. This technique achieves efficiency by overwriting input activations with output activations during computation, thereby effectively managing the constrained memory resources typical of MCUs. Moreover, the MCUNetV2 framework addresses the challenge of imbalanced memory distribution across network layers with its innovative patch-based inference strategy. Unlike traditional layer-by-layer execution, which can lead to memory bottlenecks in the early stages of the network, MCUNetV2 processes memory-intensive layers in smaller patches. This approach considerably lowers memory requirements, enabling high-resolution processing to run efficiently on memory-limited devices, a crucial factor for effective OD.

To mitigate the computational overhead associated with patch-based inference, a method for redistributing the receptive field (RF) throughout the network is proposed. By strategically decreasing the RF in the initial layers to reduce the size of input patches and the frequency of computations, and subsequently expanding the RF in later stages, the framework maintains high performance, particularly for large-scale OD tasks. The co-design methodology of MCUNetV2 seamlessly integrates NAS with advanced inference scheduling, optimizing both the architecture design and the execution strategy concurrently. This joint optimization approach allows for greater adaptability to the stringent memory and latency constraints of MCUs, resulting in more efficient and robust OD capabilities for embedded applications. Similar to its predecessor, MCUNetV1, the new MCUNetV2 framework achieves even better results on ImageNet, attaining a top-1 accuracy of 71.8%. For OD tasks, the MCUNetV2 model was utilized as the backbone for the YOLOv3 detector. The resulting MCUNetV2-YOLOv3 configurations (M4 and H7) achieved impressive mAP scores of 64.6% and 68.3%, highlighting its exceptional performance in resource-constrained environments. **TinyissimoYOLO**. TinyissimoYOLO [110] is a highly efficient and fully quantized multi-OD network, designed specifically for MCU platforms with severe resource constraints. The network architecture, initially based on YOLOv1, has been carefully adapted to deliver real-time inference performance while operating within the stringent memory and computational limits typical of edge AI systems.

TinyissimoYOLO achieves this through a combination of network simplifications and optimizations. The model supports flexible input resolutions, with the ability to adjust the number of detection classes and first-layer kernel sizes to match specific deployment requirements. This modular approach allows TinyissimoYOLO to trade off between detection accuracy and computational complexity, offering a scalable solution for various MCU architectures. Specifically, the network employs 8-bit quantization throughout, significantly reducing the memory footprint and model size by a factor of four when compared to traditional 32-bit models. For instance, TinyissimoYOLO's parameter count ranges from 441K to 887K parameters and from 32 to 57 MMACs, compared to the much larger YOLOv1 with 20 GMACs and

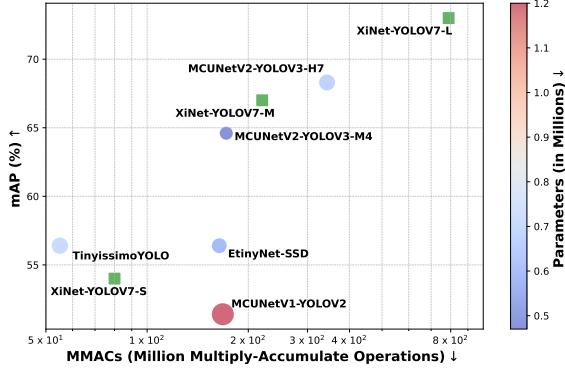


Fig. 8. Comparison of MCU-optimized OD models in terms of MMACs, mAP, and parameters. The X-axis shows the MMACs (Million Multiply-Accumulate Operations) in log scale, representing the computational cost of each model. The Y-axis represents the mAP on the PASCAL VOC benchmark, indicating the accuracy of the models. The color of the bubbles corresponds to the number of parameters (in millions), with the color bar on the right indicating the scale. Larger bubbles represent models with more parameters, showing the trade-off between computational complexity and detection performance across different models. Green squares represent XiNet models, for which the parameter count was not directly specified in the paper; hence, a fixed square size was used for these models.

45M parameters. This compact design is achieved without sacrificing accuracy, making it ideal for deployment on resource-constrained MCUs.

QAT using the QuantLab framework ¹ is a key component of TinyissimoYOLO’s optimization pipeline. This process ensures that 8-bit models retain accuracy close to their full-precision counterparts while also enabling efficient integer-only deployment. The QuantLab framework automates the conversion of full-precision models into fake-quantized and fully deployable integer models, making it easier to deploy TinyissimoYOLO on various hardware platforms, including RISC-V and ARM-based MCUs.

In experimental results, TinyissimoYOLO demonstrated its flexibility by achieving a mAP score of 56.4% on the PASCAL VOC dataset, while operating with only 55 MMACs. This makes TinyissimoYOLO well-suited for real-time, ultra-low-power OD on MCUs, such as the GAP9 multi-core RISC-V processor and the MAX78000 with an AI accelerator. Moreover, the network supports dynamic scaling for different MCU configurations, allowing developers to choose between low-latency or high-energy efficiency modes depending on the application.

EtinyNet-SSD. EtinyNet-SSD, as detailed in [162], is meticulously designed for efficient OD on MCUs. It incorporates two key innovations: the depthwise linear block (DLB) and adaptive scale quantization (ASQ). The DLB seamlessly integrates depthwise separable convolutions with linear layers, substantially reducing the model’s computational complexity while maintaining accuracy. This design ensures the network remains both lightweight and efficient, rendering it particularly suitable for deployment on MCUs. The backbone of EtinyNet is then leveraged in conjunction with the SSD architecture to achieve superior performance in OD tasks. ASQ further enhances EtinyNet-SSD by dynamically adjusting quantization scales for both weights and activations based on the input data. This technique optimizes the precision of quantized parameters, thereby minimizing memory and computational overhead. Through the synergistic integration of these strategies, EtinyNet-SSD achieves a balance between efficiency and performance, allowing it to operate effectively within the resource constraints of MCUs. Notably, by employing EtinyNet as the backbone,

¹<https://github.com/pulp-platform/quantlab>

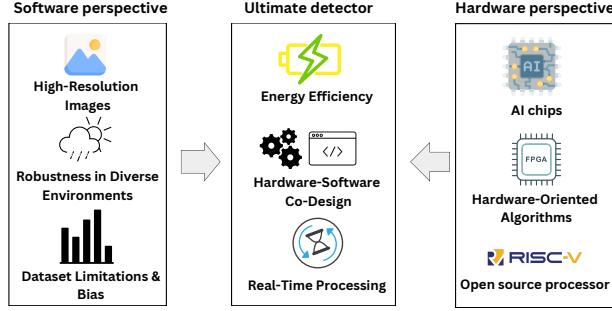


Fig. 9. Key challenges and considerations in developing the ultimate OD system, highlighting the software perspective (handling high-resolution images, robustness, and dataset limitations), hardware perspective (utilizing AI chips, FPGAs, and RISC-V architecture), and the need for energy-efficient, real-time hardware-software co-design.

EtinyNet-SSD achieves a remarkable mAP of 56.4% on the PASCAL VOC dataset, underscoring the effectiveness of this model.

XiNet-YOLOV7. XiNet-YOLOV7, as introduced in [4], presents a novel approach to optimizing NNs for OD on resource-constrained devices. The architecture builds upon the efficient operators identified through extensive analysis of standard, depthwise, and pointwise convolutions, ensuring maximum performance and energy efficiency. XiNet replaces depthwise convolutions with standard convolutions to enhance arithmetic intensity and reduce memory accesses, resulting in significant random-access memory (RAM) and parameter savings. A critical innovation in XiNet is the use of hardware aware scaling (HAS) [114], which enables independent scaling of RAM, flash memory, and operations. This allows the DNN architecture to be easily adapted to various hardware platforms, including MCUs and embedded devices. XiNet-YOLOV7 uses a hyper-parameter tuning approach involving α , β , and γ , which scale the network's width, shape, and compression to balance between computational complexity, energy consumption, and performance. The use of standard convolutions, combined with efficient attention mechanisms and skip connections, results in a more hardware-efficient model without compromising on detection accuracy. The XiNet-YOLOV7 series "S", "M", and "L" are scaled variants optimized for different operational requirements, from low-energy consumption (XiNet-YOLOV7-S) to higher accuracy (XiNet-YOLOV7-L), each adapted for deployment on platforms with limited computational power. Using HAS, the authors demonstrated the effectiveness of their method, as shown in Fig. 8. These variants were deployed on an STM32H743 MCU and achieved state-of-the-art performance, with XiNet-YOLOV7-"S", "M", and "L" achieving mAP scores of 54%, 67%, and 74.9% respectively, underscoring their suitability for real-time OD in TinyML while significantly reducing energy and memory usage.

8 Open Challenges and Future Works

In recent years, OD has seen significant advancements, with state-of-the-art models achieving impressive accuracy on large-scale datasets. However, deploying these models on embedded systems and TinyML platforms remains an ongoing challenge due to stringent memory, computational, and energy constraints. The emergence of modern techniques, including transformer-based architectures, generative AI, vision-language models (VLMs), and hardware-aware model optimizations, has opened new research avenues to address these limitations. Despite these innovations, ensuring

real-time performance, efficiency, and adaptability of OD models in highly constrained environments still requires further exploration.

In this section, we outline the open challenges associated with deploying OD on embedded systems and TinyML platforms and propose potential research directions to address these challenges. These directions are categorized from two primary perspectives: software (managing high-resolution images, ensuring robustness, and overcoming dataset limitations) and hardware (advancements in AI chips, hardware-oriented algorithms, and open-source processors), as illustrated in Fig. 9. Additionally, the increasing role of NAS, KD, and lightweight model compression techniques highlights the need for co-optimizing detection models with their hardware deployment. These techniques are particularly relevant given the shift towards multimodal and self-supervised approaches in OD, where models can leverage textual and contextual cues for more efficient and robust detection. We believe that the ideal detector will emerge from the synergy of both perspectives, addressing key challenges such as energy efficiency, real-time processing, and the integration of hardware-software co-design.

1) Managing High-Resolution Images. The challenge of balancing image resolution with computational feasibility remains a significant hurdle in embedded OD systems. Using lower-resolution images can lead to difficulties in detecting smaller or distant objects [20, 178]. Future research should explore adaptive resolution strategies that dynamically adjust image resolution based on the context and requirements of the OD task, allowing for efficient processing without compromising detection accuracy. Recent advances, such as efficient small object detection (ESOD) [96], demonstrate that small-object detection can be achieved by identifying and isolating high-interest regions at the feature level, rather than processing the full high-resolution image. This plug-and-play framework leverages adaptive slicing and sparse prediction heads to reduce computation and memory usage without degrading performance. Following this principle, future TinyML-oriented methods could explore lightweight, context-aware patch-based processing to selectively activate only relevant spatial regions, making high-resolution processing feasible even on constrained MCUs.

2) Energy Efficiency. Developing energy-efficient OD algorithms is essential for low-cost and battery-operated devices commonly used in IoT applications. Traditional DNN models tend to be power-intensive, making them less suitable for resource-constrained environments. Promising innovations, such as Spiking YOLO, leverage spiking neural networks (SNNs) to significantly reduce power consumption while maintaining competitive performance [121]. Future research should focus on optimizing SNN-based OD models and exploring hybrid architectures that integrate conventional and spiking neurons for enhanced energy efficiency. Implementing these models on neuromorphic hardware, such as the SpiNNaker chip [119], further highlights the potential for real-time OD applications in resource-constrained environments due to their low power consumption and high efficiency.

3) Robustness in Diverse Environments. Ensuring reliable OD across various environmental conditions, such as changing lighting, weather, and occlusions, remains a major challenge [97]. Future work should investigate the development of adaptive OD models that can self-tune parameters in real-time, enhancing their robustness in diverse and dynamic environments. Techniques such as transfer learning and domain adaptation could also be explored to improve model performance across different scenarios. In addition, leveraging advanced data augmentation strategies such as photometric transformations, geometric distortions, and synthetic data generation can help models generalize better to unseen environments by simulating challenging real-world conditions during training [48].

4) Real-Time Processing. Achieving low-latency OD is critical for applications such as autonomous navigation and real-time surveillance. The use of SNNs, which mimic the human brain's processing to achieve faster response times, shows potential [69]. Future research could explore optimizing these networks for real-time OD tasks, as well as integrating hardware acceleration techniques to further reduce processing delays.

5) Dataset Limitations and Bias. The lack of extensive and diverse datasets specifically designed for embedded systems can result in biases in OD models, limiting their overall effectiveness [163]. Future research should focus on developing synthetic data generation techniques and augmentation methods that can enhance the diversity and quality of training datasets. Additionally, employing unsupervised and semi-supervised learning approaches could reduce the dependence on labeled data. Initiatives such as the Wake Vision dataset [5] represent a good start in this direction, offering a large-scale, high-quality dataset specifically tailored for TinyML applications, setting a foundation for more robust and inclusive OD models.

6) Hardware-Software Co-Design. The integration of hardware and software plays a pivotal role in optimizing OD performance on embedded systems. System-level co-design strategies are essential for maximizing processing efficiency and minimizing power consumption [68]. Leveraging LLMs for architecture search, similar to the approach taken by GENIUS [172], can minimize the computational cost of searching for optimal architectures tailored for specific hardware platforms. By reducing the exploration space and discovering more efficient models quickly, LLM-driven approaches could significantly enhance the co-design process. In addition, hardware-software co-design frameworks, such as custom computational function (CFU) Playground [120], offer an efficient way to tailor AI accelerators for TinyML applications. CFU Playground enables rapid prototyping of FPGA-based AI accelerators by allowing hardware designers to seamlessly integrate CFUs into the processor pipeline, facilitating the co-optimization of hardware and software, ensuring efficient execution of OD models on resource-constrained embedded platforms.

7) AI Accelerator Chips. The adoption of specialized AI accelerator chips, including NPUs and application specific integrated circuits (ASICs), especially those integrated within modern MCUs such as the ARM Cortex-M55 STM32N6 with a built-in NPU, has significantly enhanced ODs efficiency by providing dedicated hardware for high-speed inference in energy-constrained environments. Unlike general-purpose GPUs and TPUs, which are unsuitable for MCU-class deployments, these embedded NPU solutions offer hardware-level acceleration tailored for low-latency, low-power inference. The GnetDet model, optimized for a CNNs accelerator chip, has demonstrated high frames per second (FPS) on standard computing platforms [135]. As TinyML applications demand increasingly efficient inference, future research should focus on optimizing OD models for various AI accelerators while evaluating the impact of hardware configurations, memory hierarchies, and I/O interfaces on inference speed and accuracy. Beyond conventional AI accelerator chips, emerging architectures such as neuromorphic and analog computing have gained traction for energy-efficient machine learning in resource-constrained environments [121]. Neuromorphic chips, inspired by biological neural systems, enable event-driven processing through SNNs, significantly reducing power consumption compared to traditional DNN accelerators. Similarly, analog computing leverages in-memory computation [173] to perform matrix operations with minimal energy overhead, making it particularly suitable for ultra-low-power TinyML applications. Integrating these architectures into OD pipelines could enable real-time, low-latency inference on edge devices with stringent power and memory constraints.

8) Hardware-Aware Co-Design and RISC-V Opportunities. Achieving real-time, ultra-efficient OD on embedded platforms requires tight coupling between algorithm design and hardware capabilities. Rather than adapting general-purpose algorithms to constrained devices, recent trends emphasize hardware-oriented models developed with specific architectural constraints in mind [42]. In particular, multi-objective HNAS techniques are gaining traction by jointly optimizing neural architectures for both computational efficiency and hardware cost [132]. This co-design philosophy is further enabled by the rise of open-source instruction set architectures such as RISC-V [115], which offer modularity, flexibility, and low-cost customization for embedded AI. Recent efforts like Flex-RV [113] showcase how lightweight RISC-V-based microprocessors can integrate machine learning accelerators directly into their pipeline, paving the

way for task-specific TinyML solutions. These developments suggest a promising direction where open hardware ecosystems and co-designed algorithms together drive scalable, energy-efficient OD at the edge.

9) Designing Transformer-Based OD for TinyML. Transformer-based models have achieved state-of-the-art performance in various computer vision tasks, surpassing traditional CNNs[33]. This trend has extended to OD, with the recent YOLOv12 architecture incorporating attention mechanisms to enhance feature extraction. YOLOv12's Nano variant achieves superior results with fewer parameters, showcasing the potential of attention-based models for resource-constrained environments [145]. However, deploying transformer-based object detectors on MCU-class devices presents challenges due to their computational complexity and memory requirements. Unlike CNNs, where computations are localized, transformers operate with self-attention mechanisms that scale quadratically with input size, making them inefficient for platforms with limited SRAM and energy budgets. Emerging lightweight transformer architectures, such as MobileViT [105], TinyViT [158], and EdgeFormer [44], have shown improvements for edge devices, but their applicability to TinyML remains largely unexplored, as their model footprints and computational demands often exceed the capabilities of typical MCUs. To address these limitations, future work should focus on optimizing transformers for TinyML. One promising direction involves KD to transfer the representations learned by large-scale transformer detectors into compact models tailored for MCU deployment. Through KD, a compact student model can retain the expressiveness of transformers while operating efficiently under stringent memory constraints. In addition, recent work such as MCUFormer [85] demonstrates the feasibility of transformer-based models on embedded platforms by employing techniques such as low-rank decomposition, token overwriting, and quantized attention mechanisms. MCUFormer achieves 73.6% accuracy on ImageNet while running on a Cortex-M7 MCU with only 320KB of available memory, underscoring the potential for transformer-based OD on TinyML. Further research should adapt such designs for OD pipelines, ensuring alignment between model architectures and hardware capabilities. Exploring hybrid transformer-CNN models and applying transformers to remote sensing tasks may offer additional insights into optimizing these models for TinyML. To enable efficient deployment of transformer-based OD models, new approaches to self-attention, such as Linformer [154] and Performer [25], should be explored to reduce memory overhead while maintaining global feature modeling. As an alternative approach, NAS presents an opportunity to identify optimal transformer configurations tailored to resource constraints, dynamically adjusting architectural components to fit within MCU limitations. Furthermore, specialized operator libraries like CMSIS-NN and TensorFlow Lite Micro could accelerate inference while maintaining power efficiency. The intersection of transformer methods, KD, hardware-aware model search, and efficient inference scheduling is a critical research frontier for achieving high-performance, energy-efficient OD on embedded platforms.

10) Expanding Capabilities: Multimodal Learning and Generative AI in OD. In addition to architectural streamlining for transformers, the next frontier in TinyML OD lies in enhancing object detectors with richer representational capabilities through generative and multimodal AI. Generative AI techniques, such as diffusion models [175] and generative adversarial networks (GANs) [65], offer powerful tools to improve training data quality, mitigating data scarcity and enhancing the generalization of small-scale detectors. These approaches are especially beneficial for low-shot learning, where limited annotated samples are available for embedded systems. Furthermore, multimodal KD presents a promising strategy for transferring representations from transformer-based models to more compact TinyML-friendly detectors. By integrating cross-domain knowledge from VLMs [165] and large-scale transformers[33], multimodal KD allows TinyML models to retain the contextual reasoning of more complex architectures while remaining computationally efficient. This approach extends beyond standard distillation, which focuses mainly on reducing model size, and incorporates a broader knowledge transfer that improves the model's expressive power. Another

important direction involves developing compression techniques specifically for transformer-based OD models in TinyML. While traditional compression techniques have shown success with edge devices, they must be adapted to TinyML environments, where even "lightweight" models can exceed the available SRAM and computation budgets. NAS could help identify optimized transformer architectures that balance model expressiveness and hardware feasibility. Emerging zero-shot VLM models, such as SpatialLM [143], provide capabilities to extend TinyML OD to open-set scenarios, allowing models to detect novel object categories without retraining. This could be a game-changer for TinyML OD, enabling more flexible and scalable detection systems in dynamic environments. To truly capitalize on these advancements, future research must focus on integrating generative AI, multimodal learning, and transformer compression into the TinyML domain. This includes rethinking transformer compression, developing hardware-aware architecture design, and enabling efficient cross-domain knowledge transfer to achieve real-time, energy-efficient OD on ultra-low-power embedded platforms.

9 Conclusion

In this paper, a comprehensive survey of model compression techniques such as quantization, pruning, KD, and NAS, tailored to enhance OD performance in embedded systems, was conducted. These techniques were analyzed in the context of their application to consumer electronics, IoT, and edge computing, highlighting their effectiveness in optimizing model efficiency while maintaining accuracy. Key metrics for assessing optimization success were also reviewed, and comparative insights into the practical impacts of these methods were provided. Additionally, existing challenges in deploying OD models in resource-constrained environments were outlined, with an emphasis on the need for further advancements in energy efficiency and real-time processing. The findings of this survey suggest promising avenues for future research, particularly in developing more energy-efficient algorithms and exploring integrated hardware-software solutions to enhance OD capabilities in embedded systems.

References

- [1] Andrea Giovanni Accettola and Massimo Merenda. 2023. Dataset distillation as an enabling technique for on-device training in TinyML for IoT: an RFID use case. In *2023 8th International Conference on Smart and Sustainable Technologies (SpliTech)*. 1–4.
- [2] Shay Aharon, Louis-Dupont, Ofri Masad, Kate Yurkova, Lotem Fridman, Lkdci, Eugene Khvedchenya, Ran Rubin, Natan Bagrov, Borys Tymchenko, Tomer Keren, Alexander Zhilko, and Eran-Deci. 2021. Super-Gradients.
- [3] Daghsh K Alqahtani, Muhammad Aamir Cheema, and Adel N Toosi. 2024. Benchmarking deep learning models for object detection on edge computing devices. In *International Conference on Service-Oriented Computing*. Springer, 142–150.
- [4] Alberto Ancilotto, Francesco Paissan, and Elisabetta Farella. 2023. XiNet: Efficient Neural Networks for tinyML. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 16968–16977.
- [5] Colby Banbury, Emil Njor, Matthew Stewart, Pete Warden, Manjunath Kudlur, Nat Jeffries, Xenofon Fafoutis, and Vijay Janapa Reddi. 2024. Wake Vision: A Large-scale, Diverse Dataset and Benchmark Suite for TinyML Person Detection. *arXiv e-prints*, Article arXiv:2405.00892 (May 2024), arXiv:2405.00892 pages. arXiv:[2405.00892](#) [cs.CV]
- [6] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597* (2021).
- [7] Colby R. Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N. Whatmough. 2020. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. *CoRR* abs/2010.11267 (2020). arXiv:[2010.11267](#)
- [8] Amin Banitalebi-Dehkordi. 2021. Knowledge Distillation for Low-Power Object Detection: A Simple Technique and Its Extensions for Training Compact Models Using Unlabeled Data. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. 769–778.
- [9] Amin Banitalebi-Dehkordi. 2021. Revisiting Knowledge Distillation for Object Detection. *CoRR* abs/2105.10633 (2021). arXiv:[2105.10633](#)
- [10] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc Le. 2020. Can weight sharing outperform random architecture search? An investigation with TuNAS. *CoRR* abs/2008.06120 (2020). arXiv:[2008.06120](#)
- [11] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR* abs/1308.3432 (2013). arXiv:[1308.3432](#)

- [12] Pietro Bonazzi, Thomas Rüegg, Sizhen Bian, Yawei Li, and Michele Magno. 2023. TinyTracker: Ultra-Fast and Ultra-Low-Power Edge Vision In-Sensor for Gaze Estimation. In *2023 IEEE SENSORS*. 1–4.
- [13] Han Cai, Chuang Gan, and Song Han. 2019. Once for All: Train One Network and Specialize it for Efficient Deployment. *CoRR* abs/1908.09791 (2019). arXiv:1908.09791
- [14] Han Cai, Ligeng Zhu, and Song Han. 2018. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *CoRR* abs/1812.00332 (2018). arXiv:1812.00332
- [15] Yuxuan Cai, Hongjia Li, Geng Yuan, Wei Niu, Yanyu Li, Xulong Tang, Bin Ren, and Yanzhi Wang. 2020. YOLObile: Real-Time Object Detection on Mobile Devices via Compression-Compilation Co-Design. *ArXiv* abs/2009.05697 (2020).
- [16] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-End Object Detection with Transformers. *CoRR* abs/2005.12872 (2020). arXiv:2005.12872
- [17] Bo Chen, Golnaz Ghiasi, Hanxiao Liu, Tsung-Yi Lin, Dmitry Kalenichenko, Hartwig Adam, and Quoc V. Le. 2019. MnasFPN: Learning Latency-aware Pyramid Architecture for Object Detection on Mobile Devices. *CoRR* abs/1912.01106 (2019). arXiv:1912.01106
- [18] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Mammoohan Chandraker. 2017. Learning Efficient Object Detection Models with Knowledge Distillation. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc.
- [19] Hanlin Chen, Li'an Zhuo, Baochang Zhang, Xiawu Zheng, Jianzhuang Liu, Rongrong Ji, David S. Doermann, and Guodong Guo. 2020. Binarized Neural Architecture Search for Efficient Object Recognition. *CoRR* abs/2009.04247 (2020). arXiv:2009.04247
- [20] Shaoyu Chen, Tianheng Cheng, Jiemin Fang, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. 2023. TinyDet: Accurate Small Object Detection in Lightweight Generic Detectors. arXiv:2304.03428 [cs.CV]
- [21] Shoufa Chen, Peize Sun, Yibing Song, and Ping Luo. 2022. DiffusionDet: Diffusion Model for Object Detection. *arXiv e-prints*, Article arXiv:2211.09788 (Nov. 2022), arXiv:2211.09788 pages. arXiv:2211.09788 [cs.CV]
- [22] Xingjian Chen, Jianbo Su, and Jun Zhang. 2019. A Two-Teacher Framework for Knowledge Distillation. In *Advances in Neural Networks – ISNN 2019*, Huchuan Lu, Huajin Tang, and Zhanshan Wang (Eds.). Springer International Publishing, Cham, 58–66.
- [23] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. 2019. DetNAS: Neural Architecture Search on Object Detection. *CoRR* abs/1903.10979 (2019). arXiv:1903.10979
- [24] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. 2024. Yolo-world: Real-time open-vocabulary object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16901–16911.
- [25] Krzysztof Choromanski, Valerii Likhoshesterov, David Dolan, Xingyu Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy J. Colwell, and Adrian Weller. 2020. Rethinking Attention with Performers. *CoRR* abs/2009.14794 (2020). arXiv:2009.14794
- [26] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. 2019. Visual Wake Words Dataset. *CoRR* abs/1906.05721 (2019). arXiv:1906.05721
- [27] Viviana Crescitetli, Seiji Miura, Goichi Ono, and Naohiro Kohmu. 2021. Edge devices object detection by filter pruning. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 1–7.
- [28] Li Cuimeei, Qi Zhiliang, Jia Nan, and Wu Jianhua. 2017. Human face detection algorithm via Haar cascade classifier combined with three additional classifiers. In *2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*. 483–487.
- [29] N. Dalal and B. Triggs. 2005. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. 886–893 vol. 1.
- [30] Josen Daniel De Leon and Rowel Atienza. 2022. Depth pruning with auxiliary networks for tinyml. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3963–3967.
- [31] Jieren Deng, Xin Zhou, Hao Tian, Zhihong Pan, and Derek Aguiar. 2023. Smooth and Stepwise Self-Distillation for Object Detection. *arXiv e-prints*, Article arXiv:2303.05015 (March 2023), arXiv:2303.05015 pages. arXiv:2303.05015 [cs.CV]
- [32] Caiwen Ding, Shuo Wang, Ning Liu, Kaidi Xu, Yanzhi Wang, and Yun Liang. 2019. REQ-YOLO: A Resource-Aware, Efficient Quantization Framework for Object Detection on FPGAs. *CoRR* abs/1909.13396 (2019). arXiv:1909.13396
- [33] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR* abs/2010.11929 (2020). arXiv:2010.11929
- [34] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. 2019. CenterNet: Keypoint Triplets for Object Detection. *CoRR* abs/1904.08189 (2019). arXiv:1904.08189
- [35] Christophe El Zeinaty, Glenn Herrou, Wassim Hamidouche, and Daniel Menard. 2024. Dicetrack: Lightweight Dice Classification on Resource-Constrained Platforms with Optimized Deep Learning Models. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 51–55.
- [36] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *J. Mach. Learn. Res.* 20, 1 (jan 2019), 1997–2017.
- [37] Erik Englesson and Hossein Azizpour. 2021. Generalized Jensen-Shannon Divergence Loss for Learning with Noisy Labels. *CoRR* abs/2105.04522 (2021). arXiv:2105.04522

- [38] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. [n. d.]. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. <http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html>.
- [39] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. 2019. Densely Connected Search Space for More Flexible Neural Architecture Search. *CoRR* abs/1906.09607 (2019). arXiv:[1906.09607](#)
- [40] Jiemin Fang, Yuzhu Sun, Qian Zhang, Kangjian Peng, Yuan Li, Wenyu Liu, and Xinggang Wang. 2020. FNA++: Fast Network Adaptation via Parameter Remapping and Architecture Search. *CoRR* abs/2006.12986 (2020). arXiv:[2006.12986](#)
- [41] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. 2010. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 9 (2010), 1627–1645.
- [42] Xin Feng, Youni Jiang, Xuejiao Yang, Ming Du, and Xin Li. 2019. Computer vision algorithms and hardware implementations: A survey. *Integration* 69 (2019), 309–320.
- [43] Jiyang Gao, Jiang Wang, Shengyang Dai, Li-Jia Li, and Ram Nevatia. 2018. NOTE-RCNN: NOise Tolerant Ensemble RCNN for Semi-Supervised Object Detection. *CoRR* abs/1812.00124 (2018). arXiv:[1812.00124](#)
- [44] Tao Ge, Si-Qing Chen, and Furu Wei. 2022. EdgeFormer: A parameter-efficient transformer for on-device Seq2Seq generation. *arXiv preprint arXiv:2202.07959* (2022).
- [45] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. 2019. NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection. *CoRR* abs/1904.07392 (2019). arXiv:[1904.07392](#)
- [46] C. Lee Giles, Gary M. Kuhn, and Ronald J. Williams. 1994. Dynamic recurrent neural networks: Theory and applications. *IEEE Transactions on Neural Networks* 5, 2 (1994), 153–156.
- [47] Ross Girshick, Pedro Felzenszwalb, and David McAllester. 2011. Object Detection with Grammar Models. In *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger (Eds.), Vol. 24. Curran Associates, Inc.
- [48] Cunhan Guo and Heyan Huang. 2025. Enhancing camouflaged object detection through contrastive learning and data augmentation techniques. *Engineering Applications of Artificial Intelligence* 141 (2025), 109703.
- [49] Jianyuan Guo, Kai Han, Yunhe Wang, Chao Zhang, Zhaohui Yang, Han Wu, Xinghao Chen, and Chang Xu. 2020. Hit-Detector: Hierarchical Trinity Architecture Search for Object Detection. *CoRR* abs/2003.11818 (2020). arXiv:[2003.11818](#)
- [50] Zhen Guo, Pengzhou Zhang, and Peng Liang. 2024. Shared Knowledge Distillation Network for Object Detection. *Electronics* 13, 8 (2024).
- [51] Mohammad Hajizadeh, Mohammad Sabokrou, and Adel Rahmani. 2023. MobileDenseNet: A new approach to object detection on mobile devices. *Expert Systems with Applications* 215 (2023), 119348.
- [52] Lixiang Han, Zhen Xiao, and Zhenjiang Li. 2024. Dtmm: Deploying tinyml models on extremely weak iot devices with pruning. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 1999–2008.
- [53] Jianwei Hao, Piyush Subedi, Lakshminish Ramaswamy, and In Kee Kim. 2023. Reaching for the Sky: Maximizing Deep Learning Inference Throughput on Edge Devices with AI Multi-Tenancy. *ACM Transactions on Internet Technology* 23, 1, Article 2 (Feb. 2023), 33 pages.
- [54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:[1512.03385](#)
- [55] Ryota Hinami and Shin'ichi Satoh. 2016. Large-Scale R-CNN with Classifier Adaptive Quantization. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 9907)*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer, 403–419.
- [56] Geoffrey Hinton, Jeff Dean, and Oriol Vinyals. 2014. Distilling the Knowledge in a Neural Network. 1–9.
- [57] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017). arXiv:[1704.04861](#)
- [58] Zeyi Huang, Yang Zou, Vijayakumar Bhagavatula, and Dong Huang. 2020. Comprehensive Attention Self-Distillation for Weakly-Supervised Object Detection. *CoRR* abs/2010.12023 (2020). arXiv:[2010.12023](#)
- [59] Edge Impulse. 2022. FOMO: Object Detection for Constrained Devices. Accessed: 02-01-2024.
- [60] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *CoRR* abs/1712.05877 (2017). arXiv:[1712.05877](#)
- [61] Eunjin Jeong, Jangryul Kim, and Soonhoi Ha. 2022. TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards. *ACM Trans. Embed. Comput. Syst.* 21, 5, Article 51 (oct 2022), 26 pages.
- [62] SouYoung Jin, Aruni RoyChowdhury, Huaiyu Jiang, Ashish Singh, Aditya Prasad, Deep Chakraborty, and Erik G. Learned-Miller. 2018. Unsupervised Hard Example Mining from Videos for Improved Object Detection. *CoRR* abs/1808.04285 (2018). arXiv:[1808.04285](#)
- [63] Glenn Jocher et al. 2022. *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*.
- [64] Glenn Jocher and Jing Qiu. 2024. *Ultralytics YOLO11*.
- [65] Do-Yoon Jung, Yeon-Jae Oh, and Nam-Ho Kim. 2024. A study on GAN-Based Car body part defect detection process and Comparative Analysis of YOLO v7 and YOLO v8 object detection performance. *Electronics* 13, 13 (2024), 2598.
- [66] Pilsung Kang and Athip Somtham. 2022. An Evaluation of Modern Accelerator-Based Edge Devices for Object Detection Applications. *Mathematics* 10, 22 (2022).
- [67] Mandeep Kaur and Rajni Aron. 2021. A systematic study of load balancing approaches in the fog computing environment. *The Journal of Supercomputing* 77, 8 (Feb. 2021), 9202–9247.

- [68] Kyungho Kim, Sung-Joon Jang, Jonghee Park, Eunchong Lee, and Sang-Seol Lee. 2023. Lightweight and Energy-Efficient Deep Learning Accelerator for Real-Time Object Detection on Edge Devices. *Sensors* 23, 3 (2023).
- [69] Seijoong Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. 2019. Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection. *arXiv e-prints*, Article arXiv:1903.06530 (March 2019), arXiv:1903.06530 pages. arXiv:1903.06530 [cs.CV]
- [70] Yuma Koizumi, Shoichiro Saito, Hisashi Uematsu, Noboru Harada, and Keisuke Imoto. 2019. ToyADMOS: A Dataset of Miniature-Machine Operating Sounds for Anomalous Sound Detection. *arXiv e-prints*, Article arXiv:1908.03299 (Aug. 2019), arXiv:1908.03299 pages. arXiv:1908.03299 [eess.AS]
- [71] Srinivas S. S. Kruthiventi, Pratyush Sahay, and Rajesh Biswal. 2017. Low-light pedestrian detection from RGB images using multi-modal knowledge distillation. In *2017 IEEE International Conference on Image Processing (ICIP)*. 4207–4211.
- [72] Hongbo Kuang and Ziwei Liu. 2021. Research on Object Detection Network Based on Knowledge Distillation. In *2021 4th International Conference on Intelligent Autonomous Systems (ICoAS)*. 8–12.
- [73] Jaeha Kung, David Zhang, Gooitzen Wal, Sek Chai, and Saibal Mukhopadhyay. 2018. Efficient Object Detection Using Embedded Binarized Neural Networks. *J. Signal Process. Syst.* 90, 6 (jun 2018), 877–890.
- [74] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. 2020. Inducing and Exploiting Activation Sparsity for Fast Inference on Deep Neural Networks. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 5533–5543.
- [75] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. 2018. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. *CoRR* abs/1811.00982 (2018). arXiv:1811.00982
- [76] Hei Law and Jia Deng. 2018. CornerNet: Detecting Objects as Paired Keypoints. *CoRR* abs/1808.01244 (2018). arXiv:1808.01244
- [77] Yann LeCun, John Denker, and Sara Solla. 1989. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*, D. Touretzky (Ed.), Vol. 2. Morgan-Kaufmann.
- [78] Dawei Li, Theodoros Salonidis, Nirmit V. Desai, and Mooi Choo Chuah. 2016. DeepCham: Collaborative Edge-Mediated Adaptive Deep Learning for Mobile Object Recognition. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. 64–76.
- [79] Lei Li, Alexander Linger, Mario Millhäuser, Vagia Tsiminaki, Yuanyou Li, and Dengxin Dai. 2024. Object-centric Cross-modal Feature Distillation for Event-based Object Detection. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 15440–15447.
- [80] Quanquan Li, Shengying Jin, and Junjie Yan. 2017. Mimicking Very Efficient Network for Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [81] Rundong Li, Yan Wang, Feng Liang, Hongwei Qin, Junjie Yan, and Rui Fan. 2019. Fully Quantized Network for Object Detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2805–2814.
- [82] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. 2018. Tiny-DSOD: Lightweight Object Detection for Resource-Restricted Usages. *CoRR* abs/1807.11013 (2018). arXiv:1807.11013
- [83] Ming Liang and Xiaolin Hu. 2015. Recurrent convolutional neural network for object recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3367–3375.
- [84] Siyuan Liang, Hao Wu, Li Zhen, Qiaozhi Hua, Sahil Garg, Georges Kaddoum, Mohammad Mehedi Hassan, and Keping Yu. 2022. Edge YOLO: Real-Time Intelligent Object Detection System Based on Edge-Cloud Cooperation in Autonomous Vehicles. *IEEE Transactions on Intelligent Transportation Systems* 23, 12 (2022), 25345–25360.
- [85] Yinan Liang, Ziwei Wang, Xiuwei Xu, Yansong Tang, Jie Zhou, and Jiwen Lu. 2023. Mcuformer: Deploying vision transformers on microcontrollers with limited memory. *Advances in Neural Information Processing Systems* 36 (2023), 8501–8512.
- [86] Benedetta Liberatori, Ciro Antonio Mami, Giovanni Santacatterina, Marco Zullich, and Felice Andrea Pellegrino. 2022. YOLO-Based Face Mask Detection on Low-End Devices Using Pruning and Quantization. In *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*. 900–905.
- [87] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. 2021. MCUNetV2: memory-efficient patch-based inference for tiny deep learning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 180, 13 pages.
- [88] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: tiny deep learning on IoT devices. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (Vancouver, BC, Canada) (NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 982, 12 pages.
- [89] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, and Song Han. 2024. Tiny Machine Learning: Progress and Futures. *arXiv e-prints*, Article arXiv:2403.19076 (March 2024), arXiv:2403.19076 pages. arXiv:2403.19076 [cs.LG]
- [90] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. 2016. Feature Pyramid Networks for Object Detection. *CoRR* abs/1612.03144 (2016). arXiv:1612.03144
- [91] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. *CoRR* abs/1708.02002 (2017). arXiv:1708.02002
- [92] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312 (2014). arXiv:1405.0312
- [93] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. *CoRR* abs/1806.09055 (2018). arXiv:1806.09055

- [94] Hou-I Liu, Marco Galindo, Hongxia Xie, Lai-Kuan Wong, Hong-Han Shuai, Yung-Hui Li, and Wen-Huang Cheng. 2024. Lightweight Deep Learning for Resource-Constrained Environments: A Survey. *arXiv e-prints*, Article arXiv:2404.07236 (April 2024), arXiv:2404.07236 pages. arXiv:2404.07236 [cs.CV]
- [95] Jie Liu, Chuming Li, Feng Liang, Chen Lin, Ming Sun, Junjie Yan, Wanli Ouyang, and Dong Xu. 2020. Inception Convolution with Efficient Dilation Search. *CoRR* abs/2012.13587 (2020). arXiv:2012.13587
- [96] Kai Liu, Zhihang Fu, Sheng Jin, Ze Chen, Fan Zhou, Rongxin Jiang, Yaowu Chen, and Jieping Ye. 2024. ESOD: Efficient Small Object Detection on High-Resolution Images. *IEEE Transactions on Image Processing* (2024).
- [97] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. 2018. Deep Learning for Generic Object Detection: A Survey. *CoRR* abs/1809.02165 (2018). arXiv:1809.02165
- [98] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. 2018. Path Aggregation Network for Instance Segmentation. *CoRR* abs/1803.01534 (2018). arXiv:1803.01534
- [99] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. 2015. SSD: Single Shot MultiBox Detector. *CoRR* abs/1512.02325 (2015). arXiv:1512.02325
- [100] Yuanpei Liu, Xingping Dong, Wenguan Wang, and Jianbing Shen. 2019. Teacher-Students Knowledge Distillation for Siamese Trackers. *ArXiv* abs/1907.10586 (2019).
- [101] Zhuoming Liu, Xuefeng Hu, and Ram Nevatia. 2024. Efficient Feature Distillation for Zero-Shot Annotation Object Detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 893–902.
- [102] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning Efficient Convolutional Networks Through Network Slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
- [103] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. 2018. Bi-Real Net: Enhancing the Performance of 1-bit CNNs With Improved Representational Capability and Advanced Training Algorithm. *CoRR* abs/1808.00278 (2018). arXiv:1808.00278
- [104] Siliang Ma and Yong Xu. 2023. MPDIoU: A Loss for Efficient and Accurate Bounding Box Regression. *arXiv e-prints*, Article arXiv:2307.07662 (July 2023), arXiv:2307.07662 pages. arXiv:2307.07662 [cs.CV]
- [105] Sachin Mehta and Mohammad Rastegari. 2021. MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer. *CoRR* abs/2110.02178 (2021). arXiv:2110.02178
- [106] T. Mita, T. Kaneko, and O. Hori. 2005. Joint Haar-like features for face detection. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, Vol. 2*. 1619–1626 Vol. 2.
- [107] Payal Mittal. 2024. A comprehensive survey of deep learning-based lightweight object detection models for edge devices. *Artificial Intelligence Review* 57 (08 2024).
- [108] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance Estimation for Neural Network Pruning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11256–11264.
- [109] Julian Moosmann, Pietro Bonazzi, Yawei Li, Sizhen Bian, Philipp Mayer, Luca Benini, and Michele Magno. 2023. Ultra-efficient on-device object detection on ai-integrated smart glasses with tinyissimoyolo. *arXiv preprint arXiv:2311.01057* (2023).
- [110] Julian Moosmann, Hanna Müller, Nicky Zimmerman, Georg Rutishauser, Luca Benini, and Michele Magno. 2024. Flexible and Fully Quantized Lightweight TinyissimoYOLO for Ultra-Low-Power Edge Systems. *IEEE Access* 12 (2024), 75093–75107.
- [111] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. 2021. A White Paper on Neural Network Quantization. *CoRR* abs/2106.08295 (2021). arXiv:2106.08295
- [112] Quang-Huy Nguyen, Jin Peng Zhou, Zhenzhen Liu, Khanh-Huyen Bui, Kilian Q Weinberger, and Dung D Le. 2024. Zero-shot Object-Level OOD Detection with Context-Aware Inpainting. *arXiv preprint arXiv:2402.03292* (2024).
- [113] Emre Ozer, Jedrzej Kufel, Shvetank Prakash, Alireza Raisiardali, Olof Kindgren, Ronald Wong, Nelson Ng, Damien Jausseran, Feras Alkhalil, David Kong, Gage Hills, Richard Price, and Vijay Janapa Reddi. 2024. Bendable non-silicon RISC-V microprocessor. *Nature* (25 Sep 2024).
- [114] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. 2021. PhiNets: a scalable backbone for low-power AI at the edge. *CoRR* abs/2110.00337 (2021). arXiv:2110.00337
- [115] David A. Patterson and John L. Hennessy. 2017. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [116] Hanyu Peng and Shifeng Chen. 2019. BDNN: Binary convolution neural networks for fast object detection. *Pattern Recognition Letters* 125 (2019), 91–97.
- [117] Junran Peng, Ming Sun, Zhaoxiang Zhang, Tieniu Tan, and Junjie Yan. 2019. Efficient Neural Architecture Transformation Searchin Channel-Level for Object Detection. *CoRR* abs/1909.02293 (2019). arXiv:1909.02293
- [118] Hoang-The Pham, Minh-Anh Nguyen, and Chi-Chia Sun. 2019. AIoT Solution Survey and Comparison in Machine Learning on Low-cost Microcontroller. In *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*. 1–2.
- [119] Luis A. Plana, David Clark, Simon Davidson, Steve Furber, Jim Garside, Eustace Painkras, Jeffrey Pepper, Steve Temple, and John Bainbridge. 2011. SpiNNaker: Design and Implementation of a GALS Multicore System-on-Chip. *J. Emerg. Technol. Comput. Syst.* 7, 4, Article 17 (dec 2011), 18 pages.
- [120] Shvetank Prakash, Tim Callahan, Joseph Bushagour, Colby Banbury, Alan V. Green, Pete Warden, Tim Ansell, and Vijay Janapa Reddi. 2023. CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (TinyML) Acceleration on FPGAs. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 157–167.

- [121] Jinye Qu, Zeyu Gao, Tielin Zhang, Yanfeng Lu, Huajin Tang, and Hong Qiao. 2023. Spiking Neural Network for Ultra-low-latency and High-accurate Object Detection. *arXiv:2306.12010 [cs.CV]*
- [122] Rakandhiya D. Rachmanto, Zaki Sukma, Ahmad N. L. Nabhaan, Arief Setyanto, Ting Jiang, and In Kee Kim. 2024. Characterizing Deep Learning Model Compression with Post-Training Quantization on Accelerated Edge Devices. In *2024 IEEE International Conference on Edge Computing and Communications (EDGE)*. 110–120.
- [123] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. 2015. You Only Look Once: Unified, Real-Time Object Detection. *CoRR abs/1506.02640* (2015). *arXiv:1506.02640*
- [124] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR abs/1506.01497* (2015). *arXiv:1506.01497*
- [125] Tianhe Ren, Qing Jiang, Shilong Liu, Zhaoyang Zeng, Wenlong Liu, Han Gao, Hongjie Huang, Zhengyu Ma, Xiaoke Jiang, Yihao Chen, et al. 2024. Grounding dino 1.5: Advance the "edge" of open-set object detection. *arXiv preprint arXiv:2405.10300* (2024).
- [126] Francisco Rivera Valverde, Juana Valeria Hurtado, and Abhimav Valada. 2021. There is More than Meets the Eye: Self-Supervised Multi-Object Detection and Tracking with Sound by Distilling Multimodal Knowledge. *arXiv e-prints*, Article *arXiv:2103.01353* (March 2021), arXiv:2103.01353 pages. *arXiv:2103.01353 [cs.CV]*
- [127] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. FitNets: Hints for Thin Deep Nets. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).
- [128] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. 2014. ImageNet Large Scale Visual Recognition Challenge. *CoRR abs/1409.0575* (2014). *arXiv:1409.0575*
- [129] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *CoRR abs/1801.04381* (2018). *arXiv:1801.04381*
- [130] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *CoRR abs/1801.04381* (2018). *arXiv:1801.04381*
- [131] Arief Setyanto, Theopilus Bayu Sasongko, Muhammad Ainul Fikri, and In Kee Kim. 2024. Near-Edge Computing Aware Object Detection: A Review. *IEEE Access* 12 (2024), 2989–3011.
- [132] Nilotpal Sinha, Peyman Rostami, Abd El Rahman Shabayek, Anis Kacem, and Djamil Aouada. 2024. Multi-Objective Hardware Aware Neural Architecture Search using Hardware Cost Diversity. *arXiv e-prints*, Article *arXiv:2404.12403* (April 2024), arXiv:2404.12403 pages. *arXiv:2404.12403 [cs.LG]*
- [133] Yang Song and Stefano Ermon. 2019. Generative Modeling by Estimating Gradients of the Data Distribution. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc.
- [134] Kai Su, Chowdhury MD Intsar, Qiangfu Zhao, and Yoichi Tomioka. 2020. Knowledge Distillation for Real-time On-Road Risk Detection. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. 110–117.
- [135] Baohua Sun, Tao Zhang, Jiapeng Su, and Hao Sha. 2021. GnetDet: Object Detection Optimized on a 224mW CNN Accelerator Chip at the Speed of 106FPS. *CoRR abs/2103.15756* (2021). *arXiv:2103.15756*
- [136] Siyang Sun, Yingjie Yin, Xingang Wang, De Xu, Wenqi Wu, and Qingyi Gu. 2018. Fast object detection based on binary deep convolution neural networks. *CAAI Transactions on Intelligence Technology* 3, 4 (2018), 191–197.
- [137] Febin Sunny, Mahdi Nikdast, and Sudeep Pasricha. 2022. SONIC: A Sparse Neural Network Inference Accelerator with Silicon Photonics for Energy-Efficient Deep Learning. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 214–219.
- [138] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller (Eds.), Vol. 12. MIT Press.
- [139] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2020. How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful. *IEEE Solid-State Circuits Magazine* 12, 3 (2020), 28–41.
- [140] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. 2018. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *CoRR abs/1807.11626* (2018). *arXiv:1807.11626*
- [141] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. 2020. Pruning neural networks without any data by iteratively conserving synaptic flow. *CoRR abs/2006.05467* (2020). *arXiv:2006.05467*
- [142] Qiankun Tang, Jie Li, Zhiping Shi, and Yu Hu. 2020. Lightdet: A Lightweight and Accurate Object Detection Network. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2243–2247.
- [143] ManyCore Research Team. 2025. SpatialLM: Large Language Model for Spatial Understanding. <https://github.com/manycore-research/SpatialLM>.
- [144] Juan Terven and Diana Cordova-Esparza. 2023. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *arXiv e-prints*, Article *arXiv:2304.00501* (April 2023), arXiv:2304.00501 pages. *arXiv:2304.00501 [cs.CV]*
- [145] Yunjie Tian, Qixiang Ye, and David Doermann. 2025. Yolov12: Attention-centric real-time object detectors. *arXiv preprint arXiv:2502.12524* (2025).
- [146] Antonio Torralba. 2003. Contextual Priming for Object Detection. *International Journal of Computer Vision* 53, 2 (2003), 169–191. Publisher: Kluwer Academic Publishers ISBN: 1573-1405.
- [147] P. Viola and M. Jones. 2001. 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*, Vol. 1. I–I.

- [148] Cinzia Viroli and Geoffrey J. McLachlan. 2017. Deep Gaussian Mixture Models. *arXiv e-prints*, Article arXiv:1711.06929 (Nov. 2017), arXiv:1711.06929 pages. arXiv:[1711.06929](#) [stat.ML]
- [149] Ao Wang, Lihao Liu, Hui Chen, Zijia Lin, Jungong Han, and Guiguang Ding. 2025. YOLOE: Real-Time Seeing Anything. *arXiv preprint arXiv:2503.07465* (2025).
- [150] Chaoqi Wang, Guodong Zhang, and Roger Baker Grosse. 2020. Picking Winning Tickets Before Training by Preserving Gradient Flow. *CoRR abs/2002.07376* (2020). arXiv:[2002.07376](#)
- [151] Ching-Hao Wang, Kang-Yang Huang, Yi Yao, Jun-Cheng Chen, Hong-Han Shuai, and Wen-Huang Cheng. 2024. Lightweight Deep Learning: An Overview. *IEEE Consumer Electronics Magazine* 13, 4 (2024), 51–64.
- [152] Jiaobao Wang, Yuming Chen, Zhaohui Zheng, Xiang Li, Ming-Ming Cheng, and Qibin Hou. 2024. CrossKD: Cross-Head Knowledge Distillation for Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 16520–16530.
- [153] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, and Chunhua Shen. 2019. NAS-FCOS: Fast Neural Architecture Search for Object Detection. *CoRR abs/1906.04423* (2019). arXiv:[1906.04423](#)
- [154] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-Attention with Linear Complexity. *CoRR abs/2006.04768* (2020). arXiv:[2006.04768](#)
- [155] Xiaoxing Wang, Jiale Lin, Junchi Yan, Juanping Zhao, and Xiaokang Yang. 2022. EAutoDet: Efficient Architecture Search for Object Detection. *arXiv e-prints*, Article arXiv:2203.10747 (March 2022), arXiv:2203.10747 pages. arXiv:[2203.10747](#) [cs.CV]
- [156] Ziwei Wang, Ziyi Wu, Jiwen Lu, and Jie Zhou. 2020. BiDet: An Efficient Binarized Object Detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [157] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR abs/1804.03209* (2018). arXiv:[1804.03209](#)
- [158] Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. 2022. Tinyvit: Fast pretraining distillation for small vision transformers. In *European conference on computer vision*. Springer, 68–85.
- [159] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2016. Aggregated Residual Transformations for Deep Neural Networks. *CoRR abs/1611.05431* (2016). arXiv:[1611.05431](#)
- [160] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Vikas Singh, and Bo Chen. 2020. MobileDets: Searching for Object Detection Architectures for Mobile Accelerators. *CoRR abs/2004.14525* (2020). arXiv:[2004.14525](#)
- [161] Jiaolong Xu, Peng Wang, Heng Yang, and Antonio M. López. 2018. Training a Binary Weight Object Detector by Knowledge Transfer for Autonomous Driving. *CoRR abs/1804.06332* (2018). arXiv:[1804.06332](#)
- [162] Kunran Xu, Yishi Li, Huawei Zhang, Rui Lai, and Lin Gu. 2022. EtinyNet: Extremely Tiny Network for TinyML. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 4 (Jun. 2022), 4628–4636.
- [163] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoonah Naveed Asghar, and Brian Lee. 2021. A Survey of Modern Deep Learning based Object Detection Models. *CoRR abs/2104.11892* (2021). arXiv:[2104.11892](#)
- [164] Christophe El Zeinat, Wassim Hamidouche, Glenn Herrou, Daniel Menard, and Merouane Debbah. 2025. Can LLMs Revolutionize the Design of Explainable and Efficient TinyML Models? arXiv:[2504.09685](#) [cs.LG]
- [165] Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. 2024. Vision-language models for vision tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).
- [166] Jiabin Zhang, Hu Su, Wei Zou, Xinyi Gong, Zhengtao Zhang, and Fei Shen. 2021. CADN: A weakly supervised learning-based category-aware object detection network for surface defect detection. *Pattern Recognition* 109 (2021), 107571.
- [167] Peizhen Zhang, Zijian Kang, Tong Yang, Xiangyu Zhang, Nanning Zheng, and Jian Sun. 2021. LGD: Label-guided Self-distillation for Object Detection. *CoRR abs/2109.11496* (2021). arXiv:[2109.11496](#)
- [168] Xingzhou Zhang, Yifan Wang, and Weisong Shi. 2019. pCAMP: Performance Comparison of Machine Learning Packages on the Edges. *CoRR abs/1906.01878* (2019). arXiv:[1906.01878](#)
- [169] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *CoRR abs/1707.01083* (2017). arXiv:[1707.01083](#)
- [170] Junhe Zhao, Sheng Xu, Runqi Wang, Baochang Zhang, Guodong Guo, David Doermann, and Dianmin Sun. 2022. Data-adaptive binary neural networks for efficient object detection and recognition. *Pattern Recognition Letters* 153 (2022), 239–245.
- [171] Yian Zhao, Wenyu Lv, Shangliang Xu, Jimman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. 2023. DETRs Beat YOLOs on Real-time Object Detection. *arXiv e-prints*, Article arXiv:2304.08069 (April 2023), arXiv:2304.08069 pages. arXiv:[2304.08069](#) [cs.CV]
- [172] Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. 2023. Can GPT-4 Perform Neural Architecture Search? *arXiv e-prints*, Article arXiv:2304.10970 (April 2023), arXiv:2304.10970 pages. arXiv:[2304.10970](#) [cs.LG]
- [173] Chuteng Zhou, Fernando Garcia-Redondo, Julian Büchel, Irem Boybat, Xavier Timoneda Comas, S. R. Nandakumar, Shidhartha Das, Abu Sebastian, Manuel Le Gallo, and Paul N. Whatmough. 2021. AnalogNets: ML-HW Co-Design of Noise-robust TinyML Models and Always-On Analog Compute-in-Memory Accelerator. *CoRR abs/2111.06503* (2021). arXiv:[2111.06503](#)
- [174] Yipeng Zhou, Huaming Qian, and Peng Ding. 2023. Lite-YOLOv3: a real-time object detector based on multi-scale slice depthwise convolution and lightweight attention mechanism. *J. Real Time Image Process.* 20, 6 (2023), 123.
- [175] Jingyuan Zhu, Shiyu Li, Yuxuan Andy Liu, Jian Yuan, Ping Huang, Jiulong Shan, and Huimin Ma. 2024. Odgen: Domain-specific object detection data generation with diffusion models. *Advances in Neural Information Processing Systems* 37 (2024), 63599–63633.

- [176] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian D. Reid. 2019. Structured Binary Neural Networks for Image Recognition. *CoRR* abs/1909.09934 (2019). arXiv:[1909.09934](https://arxiv.org/abs/1909.09934)
- [177] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. 2023. Object Detection in 20 Years: A Survey. *Proc. IEEE* 111, 3 (2023), 257–276.
- [178] F. Özge Ünel, Burak O. Özkalayci, and Cevahir Çığla. 2019. The Power of Tiling for Small Object Detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 582–591.