IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

A Survey on Real-Time Object Detection on FPGAs

SEYED HANI HOZHABR[1], and ROBERTO GIORGI[1,2]
[1]Department of Information Engineering and Mathematics, University of Siena, Siena, Italy
[2]Consorzio Interuniversitario Nazionale per l'Informatica, Rome, Italy

Corresponding author: R. Giorgi (e-mail: giorgi@unisi.it).

The European Commission partially supported this work under the EDGE-ME, AXIOM H2020 (id. 645496), TERAFLUX (id. 249013), and HiPEAC (id. 101069836) projects. This work is partly funded by the European Union - NextGenerationEU - via the PNRR M4C2-Inv1.4 Italian Research Center on High-Performance Computing, Big-Data and Quantum Computing, cascade funding project EDGE-ME, MUR-ID: CN0000013. The University of Siena, Campera-ES, and the Tuscany Region also partially supported this work under the HIPERAIHL project.

ABSTRACT This paper focuses on real-time object detection systems, analyzing existing Field-Programmable Gate Arrays (FPGAs) implementations that aim to achieve the best efficiency, performance, and accuracy at the same time. These three metrics are typically crucial for domains such as autonomous driving, and robotics. Fortunately, recent advancements in object detection models, particularly based on Convolutional Neural Networks (CNNs), have significantly improved object detection accuracy and speed. When these models are combined with FPGAs, it is possible to achieve even more energy efficiency and more easily satisfy real-time constraints. FPGAs can deliver low latency and high throughput by leveraging true parallelism making them suitable platforms for developing real-time object detection systems. This paper reviews existing literature on FPGA-based real-time object detection, discussing commonly used algorithms, acceleration techniques, and optimization strategies. Evaluation metrics and typical datasets for assessing real-time systems are also examined. We have compared the performance of these implementations by using pixel throughput as a fair metric across different systems while processing video streams or images. Insights into state-of-the-art works, comparative analysis, challenges, and future research directions are provided to guide researchers interested in leveraging FPGA devices for real-time object detection applications.

INDEX TERMS Real-time object detection, FPGA, Convolutional Neural Networks (CNNs), hardware accelerator.

I. INTRODUCTION

REAL-TIME object detection is a computer vision task that aims at identifying and localizing objects in images or video sequences with acceptable inference speed, frame rate, and accuracy, satisfying the requirement of the application of interest. Autonomous driving [1], [2], robotics [3], [4], and healthcare monitoring [5], [6] are some examples in which real-time object detection can play a vital role.

Similar to classical real-time theory, we can define hard and soft constraints for real-time object detection. In hard real-time scenarios, the system fails if a specific deadline is not met, whereas in soft real-time situations, missing a deadline is undesirable but can be tolerated [7].

Achieving high energy efficiency and, at the same time, performance satisfying hard constraints typically leads to the use of FPGA platforms [8], [9].

a: The role of Machine Learning

Advancements in machine learning, particularly in deep learning [10], have led to the development of highly accurate and optimized object detection algorithms. Contemporary real-time object detectors commonly leverage Convolutional Neural Network (CNN) architectures to achieve an acceptable balance between accuracy and speed.

Recently, Transformer-based object detectors [11], [12] have gained considerable attention owing to their simplified, end-to-end architectures and impressive performance. However, their high computational demands limit their practical application [13], especially on resource-constrained platforms.

b: The role of FPGAs

FPGAs are widely regarded as suitable platforms for implementing object detection systems as stand-alone target de-

vices or, in heterogeneous systems, as accelerators alongside a processor [14]. They have gained remarkable attention for implementing real-time object detection systems due to their ability to offer low and deterministic latency, along with high throughput—critical factors for real-time systems [15].

Moreover, FPGAs can directly receive images from external imaging sources and process them without the need for processor intervention. This results in improved system performance and ensures lower and more predictable end-to-end latency. This is particularly significant as real-world object detection systems commonly rely on input image data from one or multiple imaging sources [16], [17].

Furthermore, the introduction of one-stage CNN-based object detection models [18], [19] in 2016, characterized by simpler architectures and higher speed compared to their two-stage counterparts [20]–[22], has facilitated the development of real-time systems, particularly on resource-constrained devices like FPGAs. Over time, with the introduction of more advanced one-stage object detection models [23], [24], these architectures have managed to strike a balance between accuracy and speed. Figure 1 illustrates the increasing contribution of "real-time" systems in the context of implementing object detection on FPGAs over the last decades. This figure highlights that the introduction of these detection models has marked a significant turning point in attention to this topic.

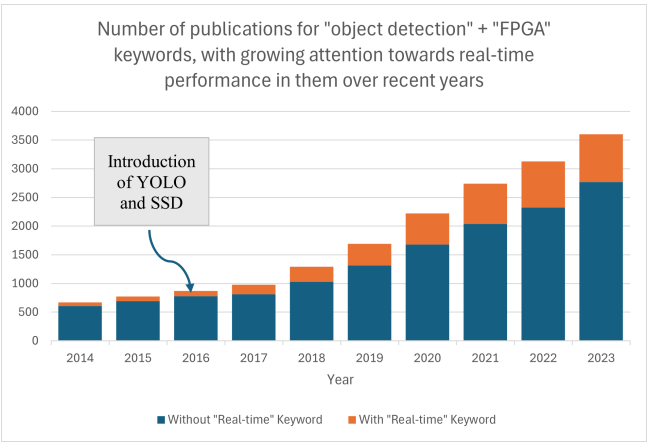


FIGURE 1. Increasing number of publications related to real-time object detection and FPGA over the years, especially after the introduction of one-stage object detection models like YOLO [18] and SSD [19], based on data extracted from Google Scholar.

c: Other related surveys

This review examines implementations of real-time object detection on FPGAs, while other recent surveys partially address the topic as a whole as illustrated in Table 1. The survey literature is quite large, so we divided the contributions in three main categories discussed below: i) surveys on general object detection; ii) surveys on object detection on resource-constrained platforms; iii) surveys on FPGA-based object detection. In those categories, we discuss below and highlight in Table 1 the surveys of the last three years (2022, 2023, 2024), assuming that they already include comprehensively previous works and we explain how we extended such reviews.

Surveys on General Object Detection

Numerous research works provide reviews of object detection algorithms, covering traditional and deep learning-based methods, commonly used datasets, typical performance metrics, and more [26], [28], [30], [32], [34]–[39].

The work by Amjoud et al. [26] provides an overview of the current state of object detection based on deep learning, covering two-stage anchor-based detectors, one-stage anchor-based detectors, anchor-free detectors, and transformer-based detectors. It also evaluates existing models across major object detection datasets such as Pascal VOC [40] and MS-COCO [41].

J. Kaur et al. [32] review key techniques, datasets, and tools for object detection, with a focus on the advancements and challenges within the field. It covers various detection methods, noting how deep neural networks have improved performance, while challenges like small object detection, video analysis, and dataset limitations persist. The paper also explores future directions, such as combining detection models. They highlighted key research opportunities, such as designing efficient networks and leveraging multi-source information to enhance robustness in detection tasks.

The paper by R. Kaur et al. [28] reviews the evolution of object detection, focusing on deep convolutional neural networks and their applications across various fields. It compares deep learning methods with other object detection techniques and discusses key frameworks, architectures, datasets, and evaluation metrics.

Zou et al. [30] provide a comprehensive review of popular detectors, key technologies, speedup methods, datasets, and metrics spanning 20 years of object detection history, along with discussing promising future directions.

Surveys on Object Detection on Resource-Constrained Platforms

Implementing these algorithms on resource-constrained devices is a practical and interesting topic. Extensive research has been conducted in this domain, covering various aspects of this issue [27], [29], [31], [33], [42]–[44]. From these works, several important design aspects emerge that need to be taken care of. They include the design of appropriate algorithms, techniques to simplify architectures, energy efficiency (e.g., optimizing memory access, a primary source of power consumption), edge-specific metrics, such as cost, model size versus resource availability, and common metrics like accuracy, latency, and throughput.

Huang et al. [31], examine the challenges and advancements in enabling real-time object detection on resource-constrained edge devices. It explores methods of data processing on such devices, the development of efficient deep learning models, and the importance of input size reduction in neural networks for edge scenarios. The authors analyze existing approaches, comparing traditional machine learning

TABLE 1. Comparison of recent related review studies with the present work, focusing on the main topics discussed.

Year	Ref.	Main Surveyed Topics (Corresponding to ours)				Highlights
		Object Detection (OD) Methods	Real-time Performance	FPGA Implementation	HW Acceleration/ Optimization Techniques	
2024	**Ours**	**YES**	**YES**	**YES**	**YES**	**Focuses mainly on suitable OD models, HW acceleration methods, and optimization strategies to achieve real-time performance on FPGAs**
2024	Mittal [25]	YES	YES	NO	NO	Focuses on the design and evaluation of lightweight object detection models based on deep learning, specifically tailored for edge devices
2023	Amjoud et al. [26]	YES	NO	NO	NO	Provides an overview of the current state of deep learning-based OD models
2023	Kamath et al. [27]	YES	NO	NO	NO	Provides a systematic review on deep learning-based OD on resource-constrained devices, emphasizing lightweight models
2023	R. Kaur et al. [28]	YES	NO	NO	NO	Reviews the evolution of OD, emphasizing DNN-based models while discussing key frameworks, architectures, datasets, and evaluation metrics
2023	Setyanto et al. [29]	YES	YES	NO	NO	Reviews OD methods and simplification strategies suitable for edge computing, focusing on compression techniques
2023	Zou et al. [30]	YES	YES	NO	NO	Reviews the development of OD with a discussion on detection speedup helpful for real-time performance
2022	Huang et al. [31]	YES	YES	NO	NO	Discusses CNN-based OD models suitable for implementation on Raspberry Pi while achieving real-time performance
2022	J. Kaur et al. [32]	YES	YES	NO	NO	Discusses various OD techniques, applications, and related tools and datasets, with a focus on existing challenges
2022	Zaidi et al. [33]	YES	YES	NO	NO	Reviews OD methods, focusing on lightweight models suitable for real-time performance on edge devices
2022	Zeng et al. [14]	YES	NO	YES	YES	Reviews some software and hardware optimization methods for the implementation of OD models on FPGAs

methods and deep learning techniques for object positioning and classification, and evaluate their suitability for edge devices like Raspberry Pi. The paper also highlights trade-offs between computational efficiency, accuracy, and energy consumption in these contexts.

Zaidi et al. [33] review recent developments in deep learning for object detection, covering benchmark datasets, evaluation metrics, and lightweight models for edge devices. It emphasizes the progress in creating faster, more accurate detectors to achieve real-time performance suitable for embedded applications.

The work by Kamath et al. [27] explores recent trends in deep learning-based object detection for resource-constrained devices. The study identifies key research areas, techniques, devices, and applications in this domain, based on a systematic literature review of 167 studies. It highlights the need for lightweight models that perform well on constrained devices, with a focus on the transportation industry.

Setyanto et al. [29] discuss various approaches for deploying object detection on near-edge devices, such as autonomous vehicles. They emphasize the challenges posed by limited computational resources and the need for efficient model compression techniques, such as network pruning and quantization, to maintain accuracy while reducing computational demand.

Mittal's review [25] provides an exploration of deep learning-based lightweight object detection models designed for edge devices, addressing the increasing need for accurate, efficient, and low-latency detection systems. The study presents a taxonomy of lightweight detection algorithms, delves into key backbone architectures, and evaluates their performance on widely used datasets such as MS COCO [41] and Pascal VOC [45]. Additionally, it highlights the challenges and opportunities in the field, outlines real-world applications, compares state-of-the-art models, and offers insights into optimization strategies to enhance the performance of object detection models on edge platforms.

FIGURE 2. Organization of the paper.

Surveys on FPGA-based Object Detection

Despite the proven benefits of FPGAs in developing object detection systems [15], especially when real-time performance is required [46]–[48], relatively little attention has been devoted to reviewing the related accomplished works.

Zeng et al. [14] partially address this gap by examining and comparing existing works in this domain. They also provide a summary of some software and hardware optimization techniques for the implementation of FPGA-based accelerators for object detection. However, in this survey, there is no emphasis on the real-time related aspects, which are connected more closely and examined in our work, besides analyzing also more recent works.

d: Our contribution

However, to the best of our knowledge, this work marks the first dedicated review toward the implementation and optimization of "real-time object detection on FPGAs". As illustrated in Figure 1, the contribution of real-time systems in this field is on the rise, highlighting the growing importance of focusing on this topic. Our discussion covers appropriate object detection algorithms, acceleration techniques, and optimization strategies for soft and hard real-time systems. Furthermore, we conduct a comprehensive examination and comparison of state-of-the-art works in this domain from various perspectives. Additionally, we delve into the existing challenges and propose potential solutions for achieving real-time object detection on FPGAs.

e: Organization of this survey

As illustrated in Figure 2, the rest of the paper is organized as follows. Section II provides background information about object detection, the hard as well as soft real-time concepts. It also discusses evaluation metrics and datasets commonly used in assessing real-time object detection systems. Section III provides an overview of FPGAs, exploring fundamental concepts, advantages, and common design approaches. It further delves into how FPGAs are applied in computer vision, particularly in object detection, and highlights specific platforms that enable accelerated processing for these tasks. Then, Section IV delves into the implementation of real-time object detection systems using FPGAs, covering common architectures, acceleration techniques, and optimization strategies to achieve real-time performance. A review of state-of-the-art related works and their comparative analysis are also provided in that section. Section VI addresses existing challenges in achieving real-time object detection systems on FPGAs, alongside potential directions for future research. The paper concludes in section VII.

II. FUNDAMENTALS AND BACKGROUND

A. OBJECT DETECTION OVERVIEW

Object detection is a computer vision task in which one or several objects in an image or a video frame can be identified and localized [30]. In other words, object detection algorithms perform two primary tasks:

- Classification: by assigning a label to every detected object based on the obtained probability that the object belongs to a certain category.
- Localization: by providing bounding boxes (bounding box detection) or highlighting significant landmarks (landmark detection) indicating the locations of detected objects within the input image.

Nowadays, object detection is widely applied in various real-world applications, such as video surveillance [49], [50], autonomous driving [1], [51], and healthcare monitoring [5], [52] to name a few. Also, object detection forms the foundation for numerous other computer vision tasks, such as object tracking [53], instance segmentation [54]–[56], and image captioning [57], [58].

The progress in object detection can be divided into two major periods: before and after the development of Deep Neural Networks *(DNNs)* [30]. Figure 3 depicts a classification of object detection techniques. This figure also illustrates which categories of models are more popular for real-time systems and which are better suited for FPGA implementation based on algorithm complexity and model size. The study specifically focuses on models that exhibit both characteristics,

represented by the overlapping green area, which are related to CNN-based one-stage detectors.

We discuss the works that do not rely on Deep Neural Networks (here referred to as *Non-DNN-based Methods* and those that rely on DNNs (referred to here as *DNN-based Methods*).

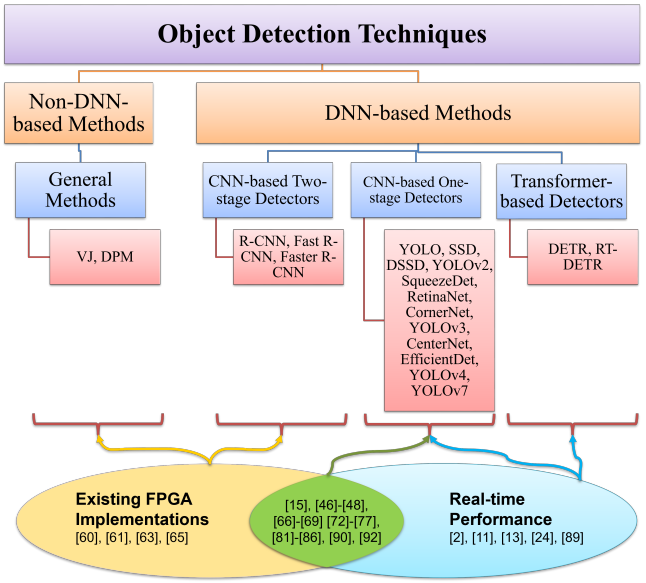


FIGURE 3. Classification of object detection techniques: Non-DNN-based and DNN-based detectors, along with instances of models for each category. It also shows which models may show real-time performance and which are more suitable for FPGA implementation. The focus of this study is the overlapping area (green zone.)

In the following, the object detection models listed in Table 2 are discussed in detail. We provide references to the original work that introduced the model, along with its main features, best-achieved performance, and, if available, references to subsequent FPGA-based implementations. For detailed descriptions of the performance metrics used in the table and the following parts, please refer to Section II-C.

The table shows that CNN-based one-stage object detection models achieve a good balance between accuracy and speed, making them excellent choices for developing real-time object detection systems.

On the other hand, the introduction of DETR [11] marks the beginning of a new era in object detection models in which Transformers are deployed. These models can capture global dependencies and contextual understanding in input images, improving object detection accuracy [93]. However, their large size and high memory requirements often render them unsuitable for scenarios such as implementing real-time object detection systems on resource-constrained devices like FPGAs [94].

1) Non-DNN-based Detectors

Early works on object detection tasks were performed using handcrafted features. Viola and Jones introduced the first real-time detector, called *VJ detector*, for human faces in the early 2000s [59]. Using sliding windows over the image to find out which windows contain a face and deploying some techniques to speed up the algorithm, this detector was about 15 times faster than other detectors at that time with comparable accuracy. However, in addition to the long training time, the VJ detector was limited to performing binary classification.

In 2005, Dalal and Triggs proposed a more accurate detector, which was capable of detecting more object classes, based on the Histogram of Oriented Gradients *(HOG)* [95]. HOG serves as a feature descriptor, widely used for extracting features from input images in computer vision, especially in object detection applications [95], [96]. Similar to methods such as Scale-Invariant Feature Transform *(SIFT)* [97], which focuses on the structure of objects, HOG counts intensity gradient orientation occurrences in localized regions of an image. In contrast to its equivalent descriptors [97], [98], HOG deals with a uniformly spaced array of cells forming a dense grid and takes advantage of overlapping local contrast normalization to enhance detection performance.

An extension of the HOG detector [95] is the Deformable Part-Based Model (*DPM*), proposed by Felzenszwalb et al. [62] in 2008. DPM is one of the well-known non-DNN object detectors, as it won several Pascal VOC detection challenges [40]. Following the ‘‘divide and conquer’’ approach, DPM tries to decompose objects into parts during training and performs ensemble-based inference on these components. It means that the problem of detecting an object can be decomposed into detecting constituted parts of that object. Later, several improvements were made by Girshick on the original DPM detector to enhance the accuracy as well as the speed of detection [64], [99]–[101].

Efforts have been made to develop object detection systems on FPGAs using these types of models [60], [61], [63]. However, in addition to offering low speed (up to 11.75 FPS, to the best of our knowledge), since these detectors rely on fixed, handcrafted features, they often fail to deliver high accuracy and robustness, particularly when detecting diverse objects in complex backgrounds. As a result, they are not well-suited to achieve real-time performance for FPGA-based object detection systems.

2) DNN-based Detectors

Object detection has witnessed a remarkable breakthrough after introducing deep neural networks [102]. The ability to learn robust and high-level representations of images without any need for handcrafted features (such as SIFT [97] and HOG [95]), which had limited accuracy of non-DNN-based object detectors, motivated many researchers to leverage DNNs in vision tasks [103].

DNN-based object detectors can be divided into two categories: CNN-based and Transformer-based detectors [104]. The architectures of CNN-based detectors have progressed from ‘‘two-stage detectors’’ to ‘‘one-stage detectors’’.

Object detectors incorporate a *backbone* network as a feature extractor to derive features from input images. VGG [105], GoogleNet [106], EfficientNet [107], and DenseNet [108] are some CNN architectures that can be adopted as

TABLE 2. A summary of the discussed models from all categories, including Non-DNN-based Detectors (NDD), DNN-based Two-stage Detectors (DTD), DNN-based One-stage Detectors (DOD), and Transformer-based Detectors (TBD).

Detection Model	Year	Detector Category	Backbone Network	#Param (M)	Required GFLOPs	Best Reported Performance				Dataset	Hardware	FPGA Implementation	Highlights
						Accuracy	Latency (ms)	FPS	Size				
VJ [59]	2001	NDD	N/A	N/A	N/A	90% (detection rate)	N/A	15	384*288	MIT	CPU (Intel Pentium III)	**Yes [60], [61]**	Fast, simple, good for face detection
DPM [62]	2008	NDD	N/A	N/A	N/A	34% AP	N/A	N/A	N/A	VOC	CPU	**Yes [63]**	Objects are represented as a collection of parts, Effective for Complex Objects
R-CNN [64]	2014	DTD	AlexNet	N/A	N/A	58.5% mAP	47000	N/A	227*227	VOC	GPU (NVIDIA Titan Black)	N/A	Uses CNN to extract features from regions proposed by region proposal part, improved accuracy compared to traditional methods, multi-stage pipeline, slow
Fast R-CNN [21]	2015	DTD	AlexNet	N/A	N/A	66.9% mAP	2000	N/A	1000*600	VOC	GPU (NVIDIA K40)	N/A	Faster than R-CNN by using a shared convolutional layer to process the entire image., multi-stage pipeline
Faster R-CNN [22]	2015	DTD	VGG-16	N/A	N/A	69.9% mAP	200	5	1000*600	VOC	GPU (NVIDIA K40)	**Yes [65]**	Faster than Fast R-CNN by using a better selective search algorithm, end-to-end network
YOLO [18]	2016	DOD	Modified GoogleNet	N/A	40.19	63.4% mAP	25	45	448*448	VOC	GPU (Geforce GTX Titan X)	**Yes [66]**	Introduced the concept of dividing the image into a grid and predicting bounding boxes, real-time performance with moderate accuracy
SSD [19]	2016	DOD	ResNet-101	N/A	N/A	74.3% mAP	N/A	46	300*300	VOC	GPU (NVIDIA Titan X)	**Yes [46], [47], [67]–[69]**	Real-time object detector, Utilized a set of default bounding boxes at different aspect ratios and scales
DSSD [70]	2017	DOD	ResNet-101	N/A	N/A	81.5% mAP	N/A	6.4	513*513	VOC	GPU (NVIDIA Titan X)	N/A	An extension of SSD, Improved accuracy by adopting a deconvolution layer to extract more semantic information, Slower than SSD
YOLOv2 [71]	2017	DOD	DarkNet-19	193	34.90	76.8% mAP	N/A	67	416*416	VOC	GPU (Geforce GTX Titan X)	**Yes [72]–[74]**	Detection of a large number of object categories, Addressed some limitations of YOLO, such as localization accuracy and small object detection
YOLOv2-tiny [71]	2017	DOD	DarkNet-19	60.5	6.97	57.1% mAP	N/A	207	416*416	VOC	GPU (Geforce GTX Titan X)	**Yes [15], [48], [75]–[77]**	Lightweight and faster alternative to YOLOv2, optimized for real-time performance on resource-constrained devices
SqueezeDet [2]	2017	DOD	SqueezeNet	26.8	77.2	80.4% mAP	N/A	32.1	1242*375	KITTI	GPU (Geforce GTX Titan X)	N/A	Designed for efficient object detection on embedded systems by reducing model size and complexity
RetinaNet [78]	2017	DOD	ResNet-101-FPN	N/A	N/A	39.1% mAP	90	5	600*600	COCO	GPU(NVIDIA M40)	N/A	Addressed the imbalance between foreground and background classes with a focal loss function
CornerNet [79]	2018	DOD	Hourglass	N/A	N/A	42.2% mAP	224	N/A	511*511	COCO	GPU (Titan X)	N/A	Detected objects as paired keypoint locations (top-left and the bottom-right corners)
YOLOv3 [80]	2018	DOD	Darknet-53	237	65.86	55.3% mAP	29	35	416*416	COCO	GPU (Geforce GTX Titan X)	**Yes [81], [82]**	Introduced multi-scale detection and feature pyramid network to improve accuracy
YOLOv3-tiny [80]	2018	DOD	Darknet-53	33.8	5.56	33.1% mAP	4.5	220	416*416	COCO	GPU (Geforce GTX Titan X)	**Yes [83]–[86]**	Efficient and faster variant of YOLOv3, offering a good balance between speed and accuracy on embedded devices
CenterNet [87]	2019	DOD	Hourglass	N/A	N/A	47% mAP	340	N/A	511*511	COCO	GPU (NVIDIA Tesla P100)	**Yes [88]**	Predicting the center point and regressing to the object size and orientation
EfficientDet [89]	2020	DOD	EfficientNet	6.6	6.1	39.6% AP	20	50	640*640	COCO	GPU (Titan V)	N/A	Utilized a compound scaling method to efficiently scale the network, balanced model efficiency and accuracy.
YOLOv4 [23]	2020	DOD	CSPDarknet-53	64.4	43.5	41.2 % mAP	N/A	96	416*416	COCO	GPU(Tesla V100)	**Yes [90]**	Introduced significant architectural improvements over YOLOv3
YOLOv4-tiny [91]	2020	DOD	CSPDarknet-53	6.1	6.9	21.7 % mAP	N/A	371	416*416	COCO	GPU(Tesla V100)	**Yes [85]**	Optimized for real-time detection with high speed, providing improved accuracy over previous tiny models
DETR [11]	2020	TBD	ResNet-50	41	86	42% mAP	N/A	28	800*1200	COCO	GPU (V100)	N/A	Transformer encoder-decoder, streamlining the detection pipeline, issues with processing small objects, too large
YOLOv7 [24]	2022	DOD	E-ELAN	36.9	51.2	51.2% mAP	N/A	161	640*640	COCO	GPU(Tesla V100)	N/A	Outperforms all YOLO versions in speed and detection accuracy
YOLOv7-tiny [24]	2022	DOD	E-ELAN	6.2	13.8	35.2% mAP	N/A	161	416*416	COCO	GPU(Tesla V100)	**Yes [92]**	Enhanced speed and accuracy for real-time detection, featuring optimized architecture for edge devices
RT-DETR [13]	2023	TBD	ResNet-50	42	136	53.1% mAP	8.8	108	1024*1024	COCO	GPU (T4)	N/A	Good speed-accuracy trade-off, eliminates NMS, Supports flexible speed tuning without retraining, still too large

backbones networks of object detectors. Recently, it has been proven that in addition to CNNs, Transformers can also be deployed as backbone networks in object detectors [109], [110].

2.a: Two-stage Detectors

The Region-based Convolutional Neural Network *(RCNN)*, introduced by Girshick et al. [20], [64], marked the initial breakthrough in utilizing DNNs for object detection. Based on this promising model, a new branch of object detectors emerged called "Two-Stage Detectors." As the name implies, these detectors are made up of two major stages:

1) Region proposal: to propose a set of candidate regions or bounding boxes likely to contain objects of interest [111].
2) Classification and refinement: to perform classification for determining the presence of objects within each proposed region. If any object is detected in this stage, the corresponding bounding box coordinates are refined for more accurate localization.

Several two-stage detectors have been proposed [21], [22], [56], [112], [113], but reviewing them is beyond the scope of this survey. The existing implementations typically require more computational resources than one-stage detectors, particularly for the region proposal task, making them less suitable for real-time object detection. Furthermore, considering their complex architectures with a vast array of parameters, deploying these detectors on resource-constrained platforms such as FPGAs is a challenging task [29]. In an attempt to implement these types of detectors on FPGAs, An et al. [65] implemented the Faster R-CNN algorithm [22] on an Arria-10 GX FPGA board. However, the results indicated a high latency of 153.6 ms, which is typically considered unsuitable for real-time performance.

2.b: One-stage Detectors

Poor speed and complex architectures of two-stage detectors motivated researchers to develop detectors capable of performing object classification and localization in one single-stage, called one-stage detectors. Although they usually suffer from poor accuracy, especially in detecting small objects, compared to two-stage detectors [30], one-stage detectors feature simpler architectures, higher speed, and adaptability to various scales [29]. These attributes make them suitable for many real-world vision tasks, especially when it comes to leveraging resource-constrained devices.

YOLO family: In contrast to two-stage detectors, which treat object detection as a classification task, Redmon et al. [18] approached it as a regression problem by proposing the first version of You Only Look Once *(YOLO)* architecture in 2016.

In YOLO, the entire image is passed through a CNN, where objects are classified and localized directly. Initially, input images are divided into several cells, each responsible for predicting multiple bounding boxes along with their corresponding confidence scores. Every bounding box is defined by its coordinates, (x, y) representing the center of the box and (w, h) denoting its width and height, respectively. For each bounding box, YOLO predicts class probabilities for every category of objects. In addition, the confidence score indicates how confident the model is that a bounding box contains an object of interest. Then, during the post-processing, redundant and low-confidence bounding boxes are removed, keeping only the most confident and non-overlapping predictions. YOLO provides a set of bounding boxes, each containing a class label and a confidence score, as output.

Li et al. [66] showed the feasibility of deploying YOLO on FPGAs for developing real-time object detection systems. They achieved about 44x inference speedup on a ZYNQ7035 compared to an Intel Core i5-6200U CPU, demonstrating promising results for real-time performance.

While it can detect multiple objects at a high speed, making it suitable for real-time vision applications, YOLO suffers from poor localization accuracy, especially for small objects [71]. So far, many different versions and variants of YOLO have been developed, each improving performance and efficiency. Among all, YOLOv2 [71], YOLOv3 [80], YOLOv4 [23], and YOLOv7 [24] are briefly discussed in the following. Although several diverse and newer versions of YOLO have been introduced at the time of writing this paper [114], [115], it was not feasible to review all of them due to the extensive volume of content. Therefore, we have restricted our discussion to the versions introduced by the main YOLO research group that have been considered for implementation on FPGAs thus far.

Unlike its predecessor, which utilized GoogleNet [106] as its backbone network, YOLOv2 adopts a less complex and lighter architecture called DarkNet-19, with 19 convolutional and 5 max-pooling layers [116]. Leveraging the Batch Normalization technique [117], which accelerates the optimization process, and anchor idea[1] [22], the generalization capability was improved in this version. These enhancements made the model more powerful in predicting objects of varying scales and shapes. Overall, YOLOv2 significantly enhanced the performance of YOLO, achieving advancements in speed (by at least 30%), accuracy (approximately 22% on the Pascal VOC dataset [40]), flexibility, and its capability to handle a wider range of object categories [71].

Numerous successful efforts have deployed YOLOv2 on FPGAs to develop real-time object detection systems [15], [48], [72]–[77] , demonstrating its significant capability in this area. For instance, Nakahara et al. [72] deployed a simplified YOLOv2 model on a ZCU102 FPGA board, achieving a throughput of 35.71 FPS, a promising processing speed in various fields such as video surveillance. Other recent works are reviewed in this study.

To further improve detection accuracy and speed, YOLOv3 [80] adopts Darknet-53 [118], a deeper (53 convolutional layers) and more complex architecture than Darknet-19 [71],

[1] Anchor boxes are predefined boxes with various aspect ratios and scales. They are utilized to align the predicted bounding boxes with the actual objects present in the image.

[116]. Also, unlike YOLOv2 [71], in which all anchor boxes have the same size, YOLOv3 uses anchor boxes in different scales and aspect ratios. Overall, YOLOv3 shows better detection accuracy, especially for small objects, compared to its ancestors.

Like YOLOv2, YOLOv3 is also popular in developing real-time object detection systems using FPGAs [81]–[85]. Wang et al. [81], demonstrated that deploying this model on FPGAs could lead to higher throughput by 6.5x and lower energy consumption by 5x compared to running the same model on a GeForce GTX1080 GPU. In another effort, Yu et al. [83] showed that a tiny YOLOv3 model could be deployed to develop object detection systems with low latency, a crucial parameter for achieving real-time performance, even on low-end FPGAs.

In YOLOv4 [23], many techniques are integrated to improve both accuracy and speed. By combining some methods, such as Weighted-Residual-Connections *(WRC)*, Cross-Stage-Partial-connections *(CSP)*, and Cross mini-Batch Normalization *(CmBN)*, YOLOv4 achieve improved results in detection accuracy (up to 43.5% mAP on MS COCO [41]) at a speed of 30 frames per second *(FPS)* and higher on a GPU-based platform (see Table 2). Using WRC, it becomes feasible to amplify the influence of crucial features acquired from preceding layers onto the current layer. This enhancement results in improved performance compared to using simple residual connections. CSP aims to lower computational complexity by splitting the feature map of the base layer into two sections and then merging them after further computations are executed on only one branch. CmBN, a variation of standard Batch Normalization [117], normalizes the model's activations based on the assumption that each batch consists of four mini-batches. This method gathers statistics only between mini-batches within a single batch, making YOLOv4 more efficient in both training and testing.

YOLOv4 is a practical choice for real-time FPGA-based object detection systems [85], [90], thanks to its efficient design and optimal balance between accuracy and speed. This study offers an in-depth analysis of the results obtained from recent advancements in this field.

YOLOv7 [24] outperforms all previous YOLO versions in speed and detection accuracy. The remarkable results are attained through optimizing model architecture, employing advanced training strategies, and utilizing efficient feature extraction techniques. These innovations enhance accuracy without significantly increasing inference costs. Additionally, this model can use parameters and computation more effectively by leveraging some proposed techniques such as compound scaling. The introduced compound scaling aims to allow developers to customize key model attributes, such as width, depth, and resolution, based on the desired application requirements and the characteristics of the target computing device while maintaining the initial model properties.

YOLOv7 is another viable option for developing real-time object detection systems on FPGAs [92], offering state-of-the-art accuracy, an optimized design for faster inference, and advanced techniques like compound model scaling that ensure compatibility with FPGA resource constraints.

SSD: Early YOLO models suffered from poor accuracy compared to two-stage detectors. The Single Shot Multibox Detector *(SSD)*, proposed by Liu et al. [19], was the first one-stage detector that achieved accuracy comparable to two-stage detectors while operating at real-time speeds on GPUs.

The backbone network in SSD is based on VGG-16 [105] with a few customizations. In particular, some fully connected layers are replaced with convolutional ones, and more multi-scale convolutional layers are added at the end of the network to improve detection performance.

In SSD, the predictions are made based on features extracted at different convolutional layers. This strategy helps the model detect objects of different scales and sizes more efficiently since every layer can provide different semantic information. In addition, a fast Non-Maximum Suppression *(NMS)* technique [119] is deployed at every stage to remove redundant bounding boxes. This leads to reduced computation compared to using NMS only at the final stage. (NMS works by selecting the bounding box with the highest confidence score and suppressing all other boxes with significant overlap, ensuring only the most accurate detections are retained.)

SSD's lightweight architecture, single-stage detection pipeline enabling fast inference, scalability across various input resolutions, and compatibility with FPGA-friendly optimizations such as quantization and pruning make it a widely adopted model for FPGA-based real-time object detection. As discussed later in this study, the favorable results achieved in recent works [46], [47], [67]–[69], strongly support its popularity in this domain.

Later, Deconvolutional SSD *(DSSD)* [70] was introduced to improve the accuracy of SSD in detecting small objects by adopting a deconvolution layer [2] to extract more semantic information. However, this improved accuracy comes at the cost of slower inference speed than SSD.

CenterNet: One problem with keypoint-based detectors is that they generate many incorrect object bounding boxes, mainly because they do not examine the cropped regions [87]. Duan et al. [87] proposed a new keypoint-based object detection named *CenterNet* to address this issue.

In this detector, a third keypoint, i.e., the center of each object, is also predicted, leading to detection accuracy improvement up to 4.9% on the MS COCO dataset [41]. In addition, two customized pooling layers are introduced to provide more precise and recognizable information about the top-left and bottom-right corners, as well as the center of each object.

solovyev et al. [88] achieved promising results (about 19 FPS) by deploying CenterNet on a Cyclone V FPGA, demonstrating the potential of this model towards developing real-time object detection on FPGAs.

[2]Unlike convolution layers, where the input image is downsized through convolution with a kernel, deconvolution layers perform the reverse process.

In the following, other one-stage detectors listed in Table 2 are briefly discussed. While these models demonstrate potential for deployment in real-time object detection systems based on their characteristics and performance, as summarized in the table, to the best of our knowledge, no FPGA implementations of them have been reported in recent publications.

SqueezeDet: SqueezeDet [2] is a lightweight, fully CNN-based object detector proposed by the developer of SqueezeNet DNN architecture [120] in 2017. Initially designed for autonomous vehicles, SqueezeDet aimed to achieve reasonable accuracy at real-time speed on resource-constrained devices.

In addition, this model addresses some other concerns related to such devices, such as model size and power efficiency. It also adopted SqueezeNet as its backbone network, with the architecture consisting of a single forward-pass neural network, making it an effectively lightweight and fast single-stage detector

RetinaNet: Recognizing the significant class imbalance as a primary factor leading to lower accuracy in one-stage detectors compared to two-stage ones, Lin et al. [78] introduced a new loss function, named ''Focal Loss'', to tackle this issue. Focal Loss can address the class imbalance bottleneck by reducing the loss contribution from well-classified examples, enabling the model to prioritize and focus on difficult-to-classify instances.

Based on the proposed loss function, they introduced Retina-Net which concentrates more on misclassified examples during the training phase, resulting in a remarkable accuracy improvement. In addition, this model takes advantage of ResNet [121] (ResNet-50 and ResNet-101) followed by a Feature Pyramid Network *(FPN)* [122] as its backbone network. Deploying FPN helps Retina-Net detect objects of different scales and dimensions more accurately (40.8% AP on COCO [41]).

CornerNet: The main idea of CornerNet [79] is finding two corners of each object, as *keypoints*, instead of dealing with anchor boxes to perform object detection. A single CNN is utilized to predict two separate heatmaps and one embedding vector for each predicted corner, covering both the top-left and bottom-right corners. Heatmaps are binary masks that indicate the locations of a class's corners, and embeddings are numerical vectors that group the corners associated with each object.

Law and Deng [79] also introduced a novel pooling layer, called *Corner Pooling*, to enable the proposed model to localize corners more accurately. Compared to its contemporary one-stage detectors, CornerNet shows a competitive accuracy of 42.2% AP on the MS COCO dataset [41].

EfficientDet: In 2020, Tan et al. [89] introduced several optimization methods to improve the efficiency of existing object detectors. These methods include a novel FPN, called Weighted Bi-directional Feature Pyramid Network (BiFPN), and a compound scaling method. While BiFPN effectively enables the model to perform multi-scale feature fusion, the compound scaling method can simultaneously and consistently scale the depth, width, and resolution of different parts of the model. By deploying EfficientNet [107] as the backbone network and integrating these optimization techniques, EfficientDet [89] was introduced.

This model can achieve remarkable accuracy (55.1% AP on MS COCO [41]) with significantly fewer parameters (up to 9x fewer) and reduced computation (up to 42x fewer FLOPs) compared to previous state-of-the-art object detectors. These advancements make EfficientDet an appealing choice for resource-constrained platforms.

Overall, CNN-based one-stage object detectors, particularly the YOLO family and SSD models, play a crucial role in real-time object detection on FPGAs due to their efficient architectures and high-speed processing. Unlike two-stage detectors that separate object proposal and classification stages, one-stage detectors integrate these tasks, enabling faster inference times—a key requirement for real-time applications. The combination of their streamlined architectures and FPGA capabilities results in high throughput, low latency, and energy-efficient designs, making them highly attractive for real-time object detection systems. Deploying these models on FPGAs supports applications like autonomous vehicles, robotics, and surveillance systems, where speed and accuracy are paramount.

2.c: Transformer-based detectors

In addition to the above-mentioned deep learning-based objection models, i.e., CNN-based one-stage and two-stage detectors, Transformer-based object detectors have recently attracted remarkable attention in the computer vision field, especially for object detection tasks [123]. However, despite demonstrating excellent performance results, as shown in Table 2, transformer-based object detectors are not well-suited for FPGA implementations due to their high computational complexity and memory demands. To the best of our knowledge, no FPGA-based real-time object detection implementations of these models have been reported.

From an architectural perspective, these detectors can be considered a subset of one-stage detectors. However, due to the distinct model structure, we review this category of detectors separately.

Transformers are a type of deep-learning model originally proposed for performing sequence-to-sequence tasks in Natural Language Processing *(NLP)* [124]. They are taking advantage of the self-attention mechanism, enabling them to identify and weigh the significance of different parts of the input data. As a result, Transformers can discover long-term dependencies and complex patterns within a sequence.

The recent achievements of transformers in NLP have prompted researchers to investigate their capabilities in the field of computer vision as well [125]. In this context, some studies have explored the integration of Transformers with CNN-based architectures [11], [126], whereas some endeavors have focused on constructing the entire model solely using Transformers and without incorporating any CNN network

for feature extracting [110], [127], [128]. In the following, some of these efforts are briefly reviewed.

DETR: In 2020, Carion et al. [11] proposed the first end-to-end object detection model that leverages Transformers, named DEtection TRansformer *(DETR)*. They introduced a novel approach for object detection, treating it as a direct set prediction task. Also, the proposed method simplifies detection by eliminating the need for hand-designed parts such as non-maximum suppression or anchor generation.

DETR utilizes a transformer encoder-decoder architecture. By reasoning about object relations and global image context, DETR outputs the final set of predictions in parallel using a fixed small set of learned object queries.

It achieves accuracy and runtime performance comparable to Faster RCNN [129] on the MS COCO dataset [41]. However, the practical application of DETRs is constrained by their high computational cost [13]. DETRs also show poor performance in detecting small objects [130]. Numerous modifications have been introduced thus far to tackle the challenges associated with DETR. [12], [131], [132].

RT-DETR: In 2023, Zhao et al. tried to reduce the computation complexity of DETR [11] to make it suitable for real-time object detection [13]. Based on those efforts, they proposed the first real-time end-to-end object detector, called Real-Time DEtection TRansformer *(RT-DETR)*.

They improved the encoder part of DETR to be able to process multi-scale features. In addition, an Intersection over Union *(IoU)*-aware query selection component was proposed to enhance the initialization of object query.

RT-DETR outperforms other real-time detectors and end-to-end detectors of comparable size, excelling in both speed and accuracy (up to 54.3% AP on COCO [41]) and 108 FPS) to achieve state-of-the-art performance. Furthermore, the proposed detector allows for adaptable adjustment of inference speed by utilizing different decoder layers without any need for retraining. This feature is extremely useful for the practical implementation of real-time object detection systems.

2.d: Object Detectors based on Vision Transformers

ViT: Prior to the introduction of Vision Transformer *(ViT)* by Dosovitskiy et al. [110] in 2020, attention mechanisms were primarily utilized for vision tasks in conjunction with CNNs. In computer vision, ''attention'' refers to examining the relationships between pairs of input image tokens or patches. This mechanism allows the model to prioritize relevant input features, aiding in capturing more informative representations of the input image.

Although some researchers tried to replace convolutional architectures with Transformers [127], [128], those models were not efficient enough for deploying on existing hardware accelerators. That was mainly because of utilizing customized attention patterns. However, ViT demonstrated that a pure Transformer could effectively perform image processing by treating it as sequences of patches without relying on CNNs [110]. The straightforward and scalable approach in ViT proves remarkably effective, particularly when combined with pre-training on extensive datasets.

As a result, Vision Transformer not only meets but often surpasses the performance of state-of-the-art CNNs [110]. This breakthrough opened the door to more widespread exploration of Transformer-based models for various computer vision tasks.

In an effort, for instance, developing a Transformer-based object detector by combining ViT and DETR [11] showed promising results [133].

Swin Transformer: Swin Transformer [109], introduced by Liu et al. in 2021, is a vision Transformer, basically designed as a backbone network for computer vision tasks. To address the challenges of applying Transformers to computer vision tasks—such as the varied scale of visual entities and higher image resolutions compared to texts—the authors introduced a hierarchical Transformer using **S**hifted **win**dows.

Swin Transformer enhances efficiency by confining self-attention to non-overlapping local windows while enabling cross-window connections. Its hierarchical design approach accommodates various scales and maintains linear computational complexity relative to image size. Swin Transformer demonstrates its versatility across image classification, semantic segmentation, and object detection tasks, achieving state-of-the-art performance on the MS COCO dataset. [41].

The results show the potential of Transformer-based models as effective vision backbones, especially with the hierarchical design and the *shifted window* approach, which also benefits all-MLP (multi-layer perceptron) architectures.

B. SOFT AND HARD REAL-TIME CONSTRAINTS

Real-time systems are those in which the accuracy of performance is determined by both the logical results of computation and the timeliness of producing those results [134], [135]. Today, they play a key role in many applications, such as robotics, manufacturing, healthcare, and autonomous driving systems.

Real-time systems ensure that tasks are executed within a precise and predictable time frame, making these systems safe, predictable, and reliable [7]. The predictability of real-time systems can be measured by evaluating the existing latency and its variation between iterations, also known as jitter [7].

Another important feature of real-time systems is their ability to manage real-time and non-real-time tasks to prevent system failure [136]. For example, in a self-driving system equipped with an object detection system, the system must be able to prioritize sending a warning signal to the central control system when detecting an obstacle rather than displaying it on the screen.

Real-time systems can be classified into two main categories as follows:

- Soft real-time systems: They are characterized by possible performance degradation rather than complete failure when response time constraints are not met [137].

- Hard real-time systems: These are systems in which failure is inevitable if response-time constraints are not met.

More precisely, in hard real-time systems, missing a deadline may have dangerous consequences, such as human injury, equipment damage, or even death [136]. This makes it crucial to ensure deterministic and predictable behavior. In autonomous vehicles, for instance, timely processing of input data is crucial for collision avoidance and safe navigation. These systems must process data from various sensors and analyze received images in real time to detect pedestrians, other vehicles, and obstacles in the path. This example highlights the significance of a hard real-time system for executing object detection, which is crucial for ensuring the safety of both passengers and pedestrians.

Designing real-time systems across different domains involves addressing various considerations [138]–[141]. As highlighted earlier, contemporary approaches heavily rely on CNN models in the context of real-time object detection systems, which is our focus in this study. A crucial challenge in implementing these systems, particularly on embedded platforms, is achieving a delicate trade-off between accuracy and speed to satisfy real-time requirements [13], [142], [143].

To that end, a range of techniques can be employed, including algorithmic and hardware implementation strategies, such as quantization and model pruning [27], [144], [145]. These approaches aim to optimize computational efficiency while preserving adequate accuracy for timely and reliable object detection. These methods are investigated in more detail in this study.

C. EVALUATION METRICS AND DATASETS

1) Common Datasets

Datasets are crucial in developing object detection models, including training and evaluation phases. During the training process, models learn to recognize patterns and features associated with different objects based on many example data provided by a dataset. Utilizing a diverse, extensive, and well-structured dataset for training can significantly improve the model's capability to detect unseen objects during inference [38].

Datasets are also used to evaluate the performance of an object detection model and compare it with others [103].

Below, we briefly discuss the commonly used datasets in the field of object detection. Table 3 summarizes the key characteristics of these datasets.

a: ImageNet

ImageNet [102] is a large-scale dataset commonly used in object classification and detection models. It has also been considered the main benchmark in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [146] for object detection and image classification. It contains over 14 million images annotated in one of two ways: image-level or object-level. The former indicates whether or not an object class exists in the image, while the latter provides a bounding box and class label for every object instance in the image.

b: MS COCO

Microsoft Common Objects in Context *(MS COCO)* dataset, introduced in 2014 by Lin et al. [41], is a large visual dataset widely used in computer vision tasks, such as object detection and image segmentation. It contains 2.5 million labeled instances in 328K images of 91 different object categories, including everyday objects and humans. It is primarily prepared for the detection and segmentation of objects that appear in their natural surroundings, containing more classes and instances compared to PASCAL VOC [45]. After years, this dataset remains one of the most popular choices in the field of computer vision, with its usage continuing to grow. In 2023 alone, it was referenced in over 2200 articles [147].

c: Pascal VOC

The Pascal Visual Object Classes *(VOC)* dataset [40], [45] is widely used in object detection, semantic segmentation, and image classification tasks. Two more popular versions of this dataset are VOC 2007 [40] and VOC 2012 [45]. The former contains 20 object classes, including persons, animals, vehicles, and some indoor stuff, in 9,963 images with 24,640 annotated objects. The latter's number of categories has remained unchanged, while it contains more data: over 11K images with 27,450 Region of Interest *(ROI)* annotated objects.

2) Performance Metrics for Real-time Object Detection

Various parameters, including accuracy, processing throughput, model complexity, and inference time, can be employed to measure the efficiency and performance of object detection models and systems. Table 4 shows some commonly used metrics for evaluating object detection models. Considering this study's primary focus, this section only discusses metrics that are more relevant to assessing real-time object detection systems, particularly those that are more important when FPGA implementation is concerned. To better understand other useful metrics in this context, such as Intersection over Union *(IoU)* and Average Precision *(AP)*, it may be helpful to review Appendix A.

One popular evaluation metric in the context of object detection is mean Average Precision *(mAP)*. Several object detection algorithms, including Faster R-CNN [22] and YOLO [18], employ mAP for assessing their models. It serves as a standard metric in various benchmark challenges like Pascal VOC [40], [45]. mAP evaluates detection accuracy across multiple classes and indicates the average of AP (see Appendix A) across all classes. Equation 1 shows how mAP is obtained for N classes of objects, where the AP_k is the AP of class k.

$$mAP := \frac{1}{N}\sum_{k=1}^{N}(AP_k) \tag{1}$$

TABLE 3. A summary of the commonly-used datasets in the context of object detection

Dataset	# of classes	# of images	# of annotated objects	Examples of object types	Tasks
ImageNet [102]	20K	14M	1M (with Bbox)	animals, everyday objects, natural scenes, plants, people, food, abstract concepts	image classification, object detection
MS COCO [41]	91	328K	2.5M	animals, person, everyday objects, vehicles, food, sport equipment	object detection, image segmentation
Pascal VOC 07 [40]	20	9963	26640	person, animals, vehicles, and indoor objects	object detection, semantic segmentation, image classification
Pascal VOC 12 [45]	20	11K	27450	person, animals, vehicles, and indoor objects	object detection, semantic segmentation, image classification

TABLE 4. A summary of commonly used metrics for evaluating object detection models.

Metric	Measure of
Intersection over Union (IoU)	Localization accuracy
mean Average Precision (mAP)	Detection (localization and classification) accuracy
Frames Per Second (FPS)	Processing throughput
Number of required FLoating-point OPerations (FLOPs)	Computational complexity
Number of trainable parameters	Computational complexity and memory footprint
Latency	Inference time

Another key metric for evaluating object detection systems is the number of frames the model can process, which is, in fact, a measure of the speed and efficiency [148]. This factor, called frame rate, is measured as Frames Per Second *(FPS)* and is directly related to the term ''real-time'' in real-time object detection systems. When discussing ''real-time'' object detection models, it is crucial to first provide a clear and precise definition of what they are.

According to the discussion provided in section II-B, the word ''real-time'' has a specific definition and does not mean simply being ''fast''. However, an inaccurate definition of real-time object detection systems and models is sometimes used. This definition may focus only on speed or processing throughput and consider meeting a certain frame rate (e.g., 30 FPS) sufficient to classify a system as real-time. However, it should be evaluated according to specific criteria depending on the requirements of the application. For example, a system that detects objects at 30 FPS for a self-driving car traveling at speeds of 110 km/h or higher cannot meet the criteria for a hard real-time system. This is because, at such a speed, the distance to objects changes by approximately 1 meter (for stationary objects) in each frame interval. Therefore, although speed, or processing throughput, is an important factor in assessing object detection systems, it should not be considered the only criterion for categorizing a system as real-time or non-real-time.

Although the frame rate metric can provide valuable insight into real-time object detection system performance, it alone may not offer a complete picture, particularly when comparing systems that process images of varying resolutions. To address this limitation, we introduce pixel throughput, as defined in Equation 2, to evaluate the performance while processing video streams or images. Pixel throughput is calculated as the product of the achieved frame rate (FPS) and the number of pixels per frame, capturing both the temporal and spatial aspects of system performance.

$$\langle Pixel\ Throughput \rangle = \langle Frame\ Rate \rangle \times \langle Pixel\ Per\ Frame \rangle \quad (2)$$

This metric, measured in Mega Pixels Per Second *(MPPS)*, allows for a more fair comparison between works that evaluate implementations that use different image sizes.

There are some other key parameters that can be used to evaluate object detection systems, which are particularly important when it comes to achieving real-time performance on resource-constrained devices. They include computational complexity, number of parameters, and latency [148]–[150].

The computational complexity of an object detection model can be represented by the number of FLoating-point OPerations *(FLOPs)* required for inference. Although higher FLOPs can result in higher accuracy, it directly affects the number of required computational resources and the model's speed [148].

The number of trainable parameters of a model can also show the model's complexity as well as memory footprint [148]. Memory footprint means how much memory we need to store all the parameters of a model. This metric can directly affect the power consumption and processing performance of the system. In fact, when the model parameters exceed the capacity of available on-chip memory of the target device, resorting to off-chip memory becomes unavoidable, a common scenario in real-world object detection systems. As the rate

of memory access increases, system speed decreases while power consumption rises. [149].

Generally speaking, latency refers to the duration it takes for a system to generate output after input is received [149]. It quantifies the delay in the response of a digital system. Typically measured in milliseconds *(ms)* [13], [48], A greater latency value indicates a slower system performance. Likewise, in an object detection system, latency indicates the time required for the model to analyze an image and provide information about the identified objects. Especially in real-time object detection systems, it is a crucial metric in system evaluation and a critical parameter for improvement if needed [48], [149].

III. FPGA BASICS AND APPLICATIONS IN OBJECT DETECTION

A. FPGA OVERVIEW

1) Basics

Field-Programmable Gate Arrays *(FPGAs)* are popular devices for implementing digital hardware circuits [151]. Advances in process technology have significantly boosted the logic capacity of FPGAs, making them more attractive for larger and more computationally intensive designs [152].

An FPGA consists of a set of programmable logic blocks, memory blocks, and specialized arithmetic units such as Digital Signal Processing (DSP) blocks. These components are interconnected by programmable interconnects, allowing for the development of highly flexible digital circuits. In addition, programmable input/output (I/O) blocks facilitate external connectivity [151]. Figure 4 shows the basic architecture of an FPGA.

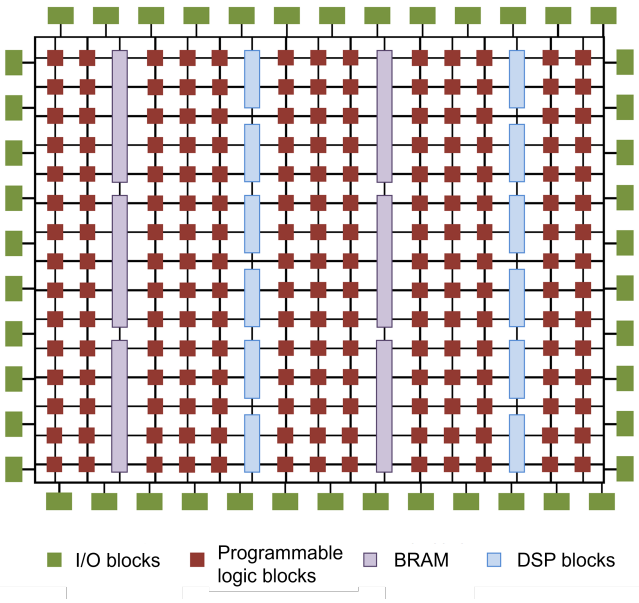


FIGURE 4. An overview of basic FPGA architecture, including programmable logic blocks, Block RAMs *(BRAMs)*, DSP blocks, input/output blocks *(I/O)*, and programmable interconnects (shown by black lines). (Figure adapted from [153]).

Some key factors in selecting an FPGA device as a target device for a specific application typically include the number of available I/O blocks and programmable logic blocks, the number and capabilities of fixed-function logic blocks (such as multipliers), and the total size of on-chip memory resources. Programmable logic blocks, which form the fundamental building blocks of FPGAs, directly impact the design's flexibility. Although there is no rigid standard for the architecture of these blocks, they typically include Flip-Flops *(FFs)*, Look-Up Tables *(LUTs)*, arithmetic carry logic, and multiplexers. LUTs are small, programmable memory blocks that can store truth tables of logic functions.

Another key factor in choosing FPGAs for a specific application is the available memory resources. There are two main types of on-chip memory within an FPGA device:

- Distributed RAM: In addition to forming various logic functions, LUTs can store data sets and act as "distributed" memory cells throughout the FPGA, as their name suggests. A k-input LUT can store 2^k bits.
- Block RAM *(BRAM)*: BRAMs are built by dedicated SRAM memory blocks. They are typically used to store large amounts of data inside an FPGA. Additionally, supplementary peripheral circuitry enhances BRAM's reconfigurability for diverse applications and facilitates its connection to the programmable routing within the FPGA.

2) Why FPGAs?

The hardware reconfigurability of FPGAs, combined with their capacity for optimization based on specific needs, makes them suitable for a wide range of applications, including digital signal processing [154], medical imaging [155], and communication encoding [156].

Thanks to their reprogrammable and versatile characteristics, FPGAs enable developers to customize and adapt systems to meet specific application requirements [157]. Also, unlike processors, which typically support one or multiple predefined data types, FPGAs offer the flexibility to accommodate various custom data types with different sizes within a single design. This capability allows FPGA designers to adopt mixed-precision designs tailored to the specific requirements of the target application.

Moreover, FPGAs provide true parallelism, making them well-suited for implementing sophisticated and compute-intensive algorithms while achieving satisfactory latency and throughput [158]. In addition, regarding power efficiency, FPGAs are considered appropriate platforms for achieving high performance per watt [159], [160].

Furthermore, FPGAs can simultaneously receive input data from diverse and multiple sources, each potentially working with different data types and communication standards [161]. It is an important feature, particularly in numerous real-world applications like real-time object detection, where connectivity with other sensors and cameras is often necessary [162].

An essential aspect to consider is the comparison of FPGAs with other hardware platforms. Table 5 provides

TABLE 5. Comparing FPGAs with some possible alternatives in different criteria.

Criteria	FPGAs	ASICs	GPUs	CPUs
General Performance	Good for parallel processing (more powerful than GPUs in some cases, like irregular data accesses and multiple-stream management), Limited computational and memory resources, totally customizable architecture	Superior performance in memory usage, energy efficiency, latency, and throughput, Highly optimized pipelines for efficient data processing, fixed architecture	Good for parallel processing, high power consumption, fixed architecture	Good for sequential tasks, fixed architecture
Cost of the hardware	Higher initial cost but Longer life cycle compared to GPUs, cost savings due to additional capabilities integration	Cost-effective only in mass production, high initial development cost	Wide-spectrum of costs	Typically cost-effective for general-purpose computing tasks
Flexibility/ Programmability	Reprogrammable for different functionalities, Adaptable to different tasks, More flexible compared to ASICs	Not reconfigurable after manufactured, suitable only for specific tasks	Not reconfigurable at the hardware level, offer software-level flexibility, less adaptable compared to FPGAs	Not reconfigurable at the hardware level, offer software-level flexibility
Power	Can achieve reasonable energy efficiency	Superior energy efficiency	Less energy efficient	Varied power efficiency depending on workload and design
I/O Interface	Can receive input data from multiple and diverse sources	Fixed I/O types after production	Typically communicates with the host through PCIe interface, Less versatile in input sources compared to FPGAs	Typically versatile in terms of input and output sources
Development Tools	Less complex and faster development compared to ASICs, Lacks strong tools for development of AI-based systems, Developing high-performance architectures is not straightforward	Requires more complex development workflows, Extended development time	User-friendly environments with extensive libraries, Faster and more straightforward code development compared to FPGAs and ASICs	Extensive and mature development ecosystem
Latency	Can deliver deterministic and low latency, Can directly connect with peripheral hardware components, which improve latency	Can offer the lowest latency than other platforms	Exhibits longer latency due to communication with the host through PCIe interface	Typically can offer lower latency than GPUs due to a single memory system
Suitability for developing phases of CNN-based object detection	Primarily for the inference phase due to their limited computational and memory resources	Deployed for both training and inference phases	Primarily leveraged for the training phase	Used for both training and inference phases, Generally slower compared to GPUs

a comprehensive comparison of FPGAs with Application-Specific Integrated Circuits (ASICs), Graphics Processing Units (GPUs), and Central Processing Units (CPUs) across various criteria. In summary, FPGAs can offer strong parallel processing capabilities, flexibility, and low latency, making them suitable for, e.g., inference phases of CNN-based object detection but with higher initial costs. ASICs, on the other hand, excel in performance, energy efficiency, and latency but lack flexibility and have high development costs, making them ideal only for mass production and specific tasks. GPUs are effective for parallel processing and training phases of AI but consume more power and exhibit longer latency. CPUs are versatile, cost-effective for general-purpose tasks, and have a mature development ecosystem but are generally less efficient for parallel processing and AI training compared to GPUs and ASICs.

3) FPGA Design Approaches

The operation of all FPGA blocks and the configuration of the programmable interconnects are typically managed by a so-called ''bitstream'', which is provided to internal dedicated *(SRAM)* cells [163]. This operation is done either once at boot time or even during the run time (partial reconfiguration).

The following outlines three primary FPGA design approaches, allowing developers to select one or a combination of them to realize their designs:

1) Hardware Description Language *(HDL)* design: In this approach, designers describe the intended functionality using an HDL, such as Verilog or VHDL. Then, the HDL design is compiled through an intricate Computer-Aided Design (CAD) flow, generating a ''bitstream'' file [163]. This bitstream file, also referred to as the configuration file, is used to program the FPGA.
2) High-level design: High-level languages like C/C++, OpenCL, and SyCL can also be employed for FPGA design. In this approach, the high-level design can be translated into its corresponding HDL using available tools such as Vivado HLS [164], HLS compiler [165], and HDL Coder [166].
3) Model-based design *(MBD)*: This approach can be considered a form of high-level design. Instead of using programming languages, high-level models or blocks are employed to design and simulate the systems to be implemented. In this approach, once the system's behavior is verified, the design is translated into HDL code. Some of the most commonly used tools in MBD include Simulink [166], LabVIEW [167], and Model Composer [168].

Adopting each of these approaches can depend on various factors, including the designer's preference and the target application. For example, software engineers who are more accustomed to high-level languages might prefer the second approach. Additionally, for some applications, such as aerospace and automotive, due to the availability of ready-made models and the ease of behavioral simulation of developing systems, adopting MBD is very popular [169].

Each of these approaches has its own advantages and disadvantages, which are beyond the scope of this study. However, it is noteworthy that in all cases, having in-depth knowledge of hardware implementation is essential for designing an optimal system.

Additionally, helpful tools, frameworks, hardware libraries, and reusable reconfigurable components (''IP Cores'') are continuously being introduced to facilitate the design of FPGA-based systems in various fields. One example of a very commonly used IP core for developing FPGA-based object detection systems is the Deep Learning Processor Unit (DPU) [170]. This programmable engine enables designers to implement many object detection models on FPGAs without requiring low-level design. In this regard, designers can use tools, such as the Vitis AI tool to select, prepare, optimize, and evaluate a detection model and compile the instructions used in the deployed DPU.

B. APPLICATIONS OF FPGAS IN COMPUTER VISION-OBJECT DETECTION

Computer Vision *(CV)* tasks typically involve techniques for capturing, processing, interpreting, and comprehending digital images or videos [171]. One of the most demanding CV tasks in this field, which is also our focus in this work, is Object Detection *(OD)*.

A typical solution is to use efficient designs like FPGA-based systems to achieve real-time performance and meet latency constraints. In the following, we analyze the architecture of such systems.

In such systems, input data is received using one or more imaging sensors [172]. These sensors can be categorized into two types based on how their output is generated: frame-based and event-based [173]. The former can generate output, i.e., images, at specific intervals or frame rates, while in the latter, each pixel can independently respond to local alterations in light intensity that exceeds a given threshold [173]. Although the predominant focus in OD research relies on frame-based sensors, there is increasing attention to the development of algorithms for event-based computer vision [174] [173] [175]. Depending on the desired application, OD algorithms extract useful and meaningful information from the received image(s) to enable the system to make decisions or just to visualize the manipulated input data.

Figure 5 illustrates a typical and abstract block diagram of a computer vision system, where raw visual data is captured from cameras or sensors in the *Data Acquisition unit*. The data is then pre-processed and prepared (in the *Data Pre-processing unit*) for the next module, i.e., *Feature Extraction unit*. Data normalization, data enhancement, and color space conversion are some tasks that may be executed during data pre-processing [176], [177]. Depending on the application of interest, relevant patterns and features are then extracted to be analyzed and processed by the main OD algorithm in the next module, i.e., *Data Processing and Analysis unit*. The results

can be interpreted and used in the subsequent unit to make a decision or trigger an appropriate action (*Decision Making and Control*). For instance, if an object is identified, it might trigger a robotic arm to pick it up. Finally, the results should be prepared to be transmitted to another system or utilized locally (*Output Preparation unit*).

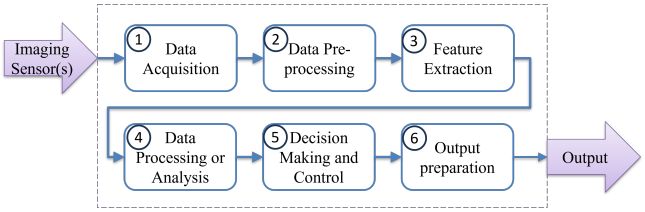


FIGURE 5. Typical structure of a computer vision system.

FPGAs or other co-processors can be integrated into the workflow shown in Figure 5 to accelerate the processing, resulting in an arrangement similar to Figure 6.

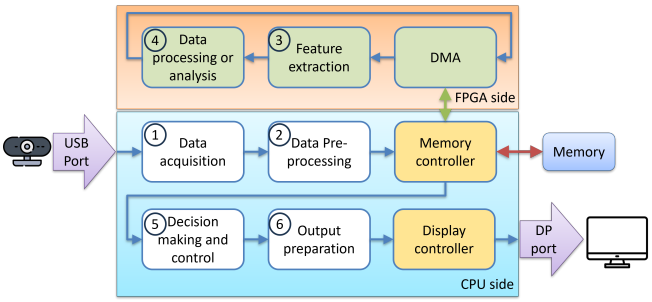


FIGURE 6. An example of FPGA co-processing architecture: images are captured by the CPU, and FPGA is used to accelerate the task. In comparison with the typical structure of a CV system shown in Figure 5, here, two tasks, i.e., the tasks performed in blocks 3 and 4, are done on the FPGA side. The white blocks represent the tasks run in the software. The green blocks indicate hardware units implemented on FPGA, and the light yellow blocks denote hardware units within the system.

Object detection includes a broad range of applications and tasks, such as image classification, medical image analysis, and facial recognition, to name a few [178]–[180]. As discussed in section II-A, deep-learning approaches, specifically using deep Convolutional Neural Networks, are increasingly used for accomplishing these tasks. However, they require high computational complexity and power consumption [31]. With all these in mind, as FPGAs can offer high performance and determinism as well as low latency systems, they are one of the popular platforms frequently adopted in OD systems [181], [182]. In addition, since FPGAs offer true parallelism, they are suitable for the implementation of inherently parallel, sophisticated, and compute-intensive algorithms in a way that the required latency and throughput needed in many OD tasks like real-time object detection are satisfied [183]–[185].

Depending on the type of deployed OD algorithm, the system specifications, and the intended application requirements, three main use cases for FPGAs in OD systems may be considered.

1) In the first use case, FPGAs can work as accelerators or co-processors alongside the main processor [181], [186], as shown in figure 6. In this case, some parts of an OD system capable of being accelerated or optimized on hardware, specifically the feature extraction and the processing modules, can be offloaded to an FPGA to make the system more efficient. The input data/image is considered to be acquired through the CPU in this use case. Therefore, this architecture is commonly used when devices like "GigE Vision feed the system" [187] and USB3 cameras as their acquisition logic can optimally be executed in CPUs [186], [188]. The data can be exchanged between the FPGA and the host memory via Direct Memory Access (DMA).
2) In the second scenario, similar to the first one, the FPGA and CPU collaborate as a co-designed, heterogeneous OD system. However, in this case, the acquisition logic is implemented on the FPGA side. This architecture is particularly adopted in OD systems equipped with cameras, such as "MIPI cameras" [189], for which acquisition logic is easily implemented on FPGAs [190]. In this scenario, the data pre-processing unit is typically implemented in the FPGA, resulting in reduced data transfer and improved overall performance [190].
3) Another use case is an OD system based only on an FPGA [86]. In a more complex scenario, a cluster of FPGAs can be considered. This architecture is more suitable for applications in which communication with connected devices through FPGAs is easier, and hardware implementation of the entire system units can result in better or at least similar to their software counterparts, providing a system with low latency and significantly high performance, in term of both power and throughput [86].

C. FPGA PLATFORMS FOR ACCELERATING OBJECT DETECTION

Although there are several FPGA suppliers worldwide, the FPGA market is dominated by AMD (formerly Xilinx) and Intel (formerly Altera) [185]. These two companies provide a broad range of FPGA devices for applications ranging from aerospace to data centers [195], [196].

Over the past decade, following significant breakthroughs in AI and its applications across various fields, such as object detection, there has been a growing trend to enhance the performance of AI-based systems using hardware accelerators.

In response, FPGA manufacturing companies have introduced a wide range of FPGA devices to facilitate the design and deployment of FPGA-based AI systems [195], [196]. Table 6 lists commonly used FPGA boards in recent works for the implementation of real-time object detection models, detailing their available resources. They continuously introduce high-performance FPGAs enhanced with AI capabilities and tools tailored for this new era. Key characteristics of these FPGA devices include better performance per watt, higher

TABLE 6. A list of some FPGA boards used in recent works for implementing Real-time object detection models, sorted by number of LUTs/ALMs.

Board	FPGA Part	Number of Available Resources				Reference
		LUTs/ALMs[a]	FFs	DSPs	Memory Blocks (Size)	
ZC702	ZYNQ XC7Z020-1CLG484	53,200	106,400	220	140 (36Kb)	[90], [191]
Pynq-Z1	ZYNQ XC7Z020-1CLG400C	53,200	106,400	220	140 (36Kb)	[192]
Ultra96	Zynq UltraScale+ MPSoC ZU3EG A484	70,560	141,120	360	216 (36Kb)	[69], [193]
ZC706	ZYNQ XC7Z045 FFG900 - 2 SoC	218,600	437,200	900	545 (36Kb)	[47], [48], [68]
ZCU104	Zynq UltraScale+ MPSoC ZU7EV	230,400	460,800	1,728	312 (36Kb)	[194]
KU040	XCKU040-1FBVA676	242,400	484,800	1,920	600 (36Kb)	[85]
ZCU102	Zynq UltraScale+ XCZU9EG-2FFVB1156	274,080	548,160	2,520	912 (36Kb)	[82]
VC707	Virtex 7 XC7VX485T-2FFG1761C	303,600	607,200	2,800	2,060 (36Kb)	[15]
Arria-10	Arria 10 GX1150	427,200	1,708,800	1,518	2,713 (20Kb)	[46], [73], [74], [76]
DE10-PRO	Stratix 10 GX2800	933,120	3,732,480	5,760	11,721 (20Kb)	[67]

[a] For Arria 10 and Stratix 10 FPGA parts, the number of Adaptive Logic Modules (ALMs) is reported. Each ALM block has an eight-input adaptable LUT, two adders, and four FFs.

memory bandwidth to alleviate bottlenecks in memory-bound AI-based object detection models, and dedicated units for executing compute-bound AI models more efficiently [170], [195], [196].

In addition, these FPGA manufacturers offer various tools and frameworks to bridge the gap between AI model development and FPGA design flow. By utilizing these frameworks, designers can seamlessly develop object detection models using popular libraries like TensorFlow and PyTorch, evaluate the preliminary models, optimize them for the target FPGA device, and compile the final models for integration into the FPGA design [195], [196].

IV. FPGA-BASED DESIGNS

This section begins by examining two primary hardware architectures—single computation engine and streaming architectures—commonly considered for FPGA-based object detection systems. It then explores various acceleration techniques, categorized into model-related and implementation-related approaches, that can enhance performance across key metrics, including throughput, accuracy, and power consumption.

Following this, the section assesses the impact of each acceleration technique on system performance, particularly within real-time object detection contexts.

A. FPGA ARCHITECTURE DESIGN APPROACHES

The hardware architectures employed in FPGA-based object detection systems can be categorized into two main types, mirroring the classification outlined in [197]: *single computation engine* and *streaming architecture* (see Table 7).

1) Single Computation Engine Architecture

The single computation engine design approach, also known as the *one-size-fits-all*, prioritizes flexibility over customization [197]. It takes advantage of a single computation engine, often in the form of a systolic array of Processing Elements *(PEs)* [202], to perform all layers of an object detection system sequentially [47], [197].

Figure 7 shows an example of adopting a single computation engine accelerator in which one operation unit is configured and deployed for performing all layers one after the other in time (not in space). The data is transferred between the FPGA and host memory via the DMA unit. There is also a ''control unit'' on the FPGA side, responsible for managing all executions. This unit receives instructions from the host side, where the central operational controller exists.

Single computation engine-based architectures temporally share common computation resources across different layers [48]. In such a scenario, the unit is time-shared, but we save resources compared to a pipelined design [198].

The computation engine is configurable to accommodate the specific characteristics of each layer. This adaptive approach reduces resource utilization and enables the implementation of any model, provided it utilizes layers supported by the engine [199], [200].

This structure offers the advantage of easy adaptation to various object detection models [144]. Nevertheless, this flexibility comes at the cost of reduced efficiency and varying and inconsistent performance when employing different models [203]. In a related experiment, Guo et al. [199] implemented YOLO [18] and Face Alignment models on Angel-Eye, a CNN accelerator featuring a single computation engine architecture, using Zynq XC7Z045. Compared to the baseline results on NVIDIA TK1 GPU, while the YOLO implementation using Angel-Eye showed a 41.37% improvement in frame rate (FPS), this for Face Alignment was 82.24%. In terms of throughput (GOPS), the improvement was 70.46% for YOLO compared to 463% for Face Alignment, indicating

TABLE 7. A summary of two FPGA architecture design approaches for implementing object detection models.

Architecture Type	Highlights	Related Works
Single computation engine	Uses one single computation block to perform all parts of the model, prioritizes flexibility over customization, uses time-shared computation resources, adaptable to various object detection models, less efficient, bottleneck in memory bandwidth (Figure 7)	[47], [68], [73], [198]–[200]
Streaming	Uses specialized blocks tailored for each part of the model, can take advantage of pipeline design, less adaptable to different models, faster, more resource consumer (Figure 8)	[15], [48], [90], [201]

that the performance gains can vary significantly across different models when utilizing such an accelerator architecture.

Furthermore, in this architecture, a controller, which is usually software-based, is required for hardware control and operation scheduling [197], which consequently reduces system efficiency [204].

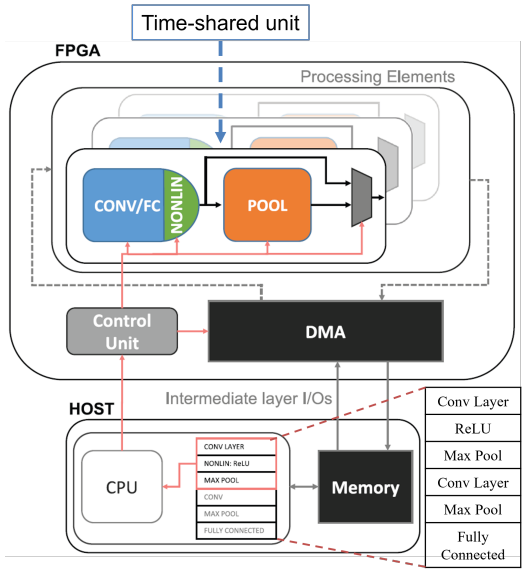


FIGURE 7. Example of an FPGA accelerator structure with a single computation engine architecture. In this approach, common computation resources are temporarily shared across different layers. A software-based controller is also deployed to control and schedule the operations. (CONV=CONVOLUTION, NONLIN= NON LINear layer, POOL=POOLing layer) (Figure adapted from [197]).

2) Streaming Architecture

On the other hand, streaming architectures are typically formed with a unique hardware block for each part of the targeted object detection model [15], [48], developing the architecture in space (rather than in time) (Figure 8).

Each computational block is optimized individually to exploit the inherent parallelism within its corresponding layer [197]. As depicted in Figure 8, in this design approach, once fetched from memory, image data passes sequentially through all dedicated hardware units for full processing until the final result is obtained and written back to memory. A pipelined architecture, where heterogeneous computational blocks are interconnected, can facilitate this data flow. This configuration enables the simultaneous execution of different layers, significantly enhancing overall system performance [15], [201].

However, it is crucial to design the PE for maintaining uniform processing times across individual layers to avoid idle states [144]. In this architecture, individual PEs are tailored and fine-tuned for each layer, creating an optimized structure specific to a particular object detection model. However, this specialization makes it less adaptable than a single computation engine architecture. Additionally, transitioning it to a new FPGA device can be challenging or even unfeasible due to the difficulty in customizing the implemented structure, especially when altering its size [197] [144]. Although pipelined accelerators can offer faster processing speeds due to reduced memory transfers, they incur higher hardware costs than one-size-fits-all configurable engines. This is because all individual layers must be simultaneously mapped in the hardware, making it less versatile and more resource-intensive [200].

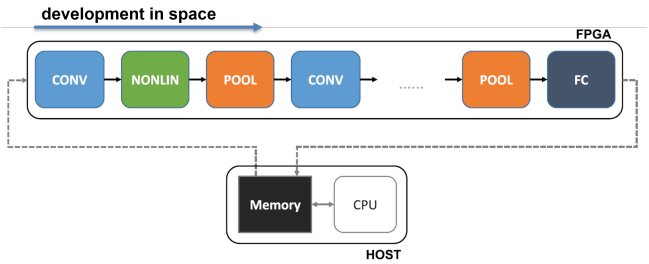


FIGURE 8. Example of an FPGA accelerator structure with a streaming architecture. Each block is individually designed and optimized to perform the required calculations of each layer. (Figure adapted from [197])

3) More Discussion

As mentioned, both architectures rely on PEs as their fundamental building blocks to execute computational tasks, primarily consisting of Multiply-ACcumulator (MAC) units [205]. In scenarios where the results of various input channels of a CNN need to be accumulated without multiplication,

costly multipliers remain unused. To improve resource utilization efficiency, particularly in streaming architectures, a specialized Convolutional PE *(Conv-PE)* could be employed [144]. The Conv-PE comprises a few multipliers based on the kernel size, succeeded by an adder tree to sum the intermediate results obtained from each input channel, typically followed by a non-linearity unit [206], [207]. Moreover, using Conv-PEs reduces latency by requiring fewer pipelined stages, helping object detection systems satisfy real-time constraints.

On the other hand, in a one-size-fits-all architecture, frequently reading or writing parameters and intermediate calculation results to external memory creates a bottleneck in memory access and bandwidth. This results in reduced system performance and increased power consumption. In real-time systems, in particular, to compensate for it, additional buffers can be utilized to store weights and input data on available on-chip memory [208].

B. HARDWARE ACCELERATION TECHNIQUES FOR FPGA-BASED OBJECT DETECTION

In the context of real-time object detection, given their heavy reliance on CNN-based models, as discussed in section II-A, there is significant interest in designing and deploying suitable hardware accelerators to enhance system performance.

Generally speaking, an accelerator refers to a specialized hardware component designed to execute a specific set of tasks more efficiently in terms of performance and energy consumption compared to a general-purpose processor or CPU [144], [209]. The development of floating point co-processors was one of the first attempts to adopt accelerators [210]. Since then, designers are increasingly taking advantage of hardware accelerators to enhance digital systems' power efficiency and performance. However, they have gained more attention in recent years, mostly due to AI breakthroughs, especially in DNNs [149].

Demands for powerful hardware accelerators have increased as AI-based applications have become more prevalent in various fields. Enhancing energy efficiency, improving performance, and addressing model complexity of modern AI models are the most important reasons why developing accelerators is being paid remarkable attention in this era [39], [144]. Recent trends to solve more challenging tasks more accurately in different fields, such as computer vision and natural languages, through AI-based applications, have made researchers introduce more complex and larger models year by year. Consequently, such models cannot be exclusively run on general-purpose processors, as they lack the computational power necessary for both training and inference within reasonable time frames. It becomes a more important issue when developing real-time applications.

Furthermore, as AI models continue to grow in size, the need for increased memory access and data movement also rises. Memory access is notably more energy-intensive compared to arithmetic computations [149], [233]. Considering the limited capacity of on-chip storage in general-purpose processors and the need for external memory access, energy efficiency is the most important reason for developing hardware accelerators. In addition to incorporating specialized hardware features to accelerate intensive computations, AI accelerators can be designed to reduce memory access and provide larger on-chip caches for improved performance.

FPGAs are highly regarded as suitable platforms for accelerating object detection because they can perform parallel and pipelined computations efficiently while maintaining high energy efficiency. Moreover, FPGAs' architectural flexibility and reconfigurability enable the implementation of custom logic units tailored to specific tasks in different object detection models. This flexibility can also make it possible to apply various optimization techniques, both at the hardware and software levels, aimed at improving system performance through architectural enhancements and object detection model simplifications. Therefore, FPGAs have emerged as the optimal hardware platform, offering both speed and energy efficiency for implementing complex object detection models and accelerating them at the edge.

This section delves into some commonly adopted techniques for designing FPGA-based hardware accelerators. Table 8 shows the methods discussed in the following two subsections, divided into two main categories: Model-related and Implementation-related techniques. The former includes the methods adopted to prepare and optimize object detection models for FPGA implementation, while the latter explains techniques applicable during hardware architecture design and FPGA implementation. It is crucial to note that accuracy and execution time are two pivotal parameters in real-time systems, particularly in hard real-time scenarios [51]. Therefore, achieving a balance between these factors is of utmost importance.

1) Model-related Techniques for Hardware Acceleration

Some typical techniques that can be used to make the implementation feasible in hardware or allow for more efficient execution of the target application are discussed in detail below, namely: a) Pruning; b) Quantization; c) Distillation; d) Hardware-aware Neural Architecture Search.

a: Pruning

Pruning is typically performed in software before deploying the model on hardware. However, performance can be further enhanced if the pruning process accounts for the capabilities and custom hardware features of the target FPGA, as highlighted in some of the works discussed below.

Pruning is defined as identifying and eliminating neurons, kernels, weights, and channels that have minimal or negligible impact on the final accuracy of an AI model to reduce network complexity [234]. It offers several advantages, including reducing computational load and required memory and improving accuracy per parameter and per operation [150], [211]. To compensate for the possible effect of this on the accuracy, the pruned model, also known as a sparse network [234], needs to be retrained [235]. To avoid slow convergence

TABLE 8. Commonly used Hardware Acceleration Techniques for FPGA-based Object Detection.

1. Model-related Techniques for Hardware Acceleration	FPGA-related references
a. Pruning	[47], [73], [211]–[214]
b. Quantization	[15], [47], [48], [69], [73], [77], [82], [193]
c. Knowledge Distillation	[221]–[224]
d. Hardware-aware Neural Architecture Search	[77], [192], [227]–[229]

2. Implementation-related Techniques for Hardware Acceleration	FPGA-related references
a. Data Reuse	[68], [69], [82], [193], [215], [216]
b. Mathematical-based Optimization	[68], [217]–[220]
c. Systolic Arrays	[82], [225], [226]
d. Code Modification	[15], [47], [69], [73], [90], [92]
e. Roofline-based Optimization	[47], [73], [211], [219], [230], [231]
f. Pipelining	[15], [47], [48], [74], [82], [83], [232]

during the retraining of a sparse network, pruning should be conducted incrementally, with each group of layers pruned in a separate stage [234].

Pruning methods can be broadly classified into two categories [236]: *unstructured pruning* and *structured pruning*. Figure 9 illustrates the distinction between these approaches in a fully connected layer.

In unstructured pruning [237]–[239], weights with low sensitivity are selectively removed throughout the network. This method allows for aggressive pruning, removing a significant portion of neural network parameters with insignificance accuracy loss. Wang et al. (2020) [73] apply unstructured pruning to YOLOv2 [71], demonstrating its effectiveness in enabling real-time object detection on FPGAs. Detailed results are provided in Sections V-A and V-B. However, unstructured pruning leads to sparse matrix operations, which are challenging to accelerate and are often memory-bound [240].

Structured pruning [198], [241], [242], on the other hand, involves removing a group of parameters, such as the entire kernel. This approach alters the input and output shapes of layers and weight matrices, allowing for dense matrix operations to continue. However, aggressive structured pruning often results in considerable accuracy degradation. Achieving state-of-the-art performance during training and inference with high levels of pruning remains an open challenge [243].

Pruning may result in data sparsity and inefficient load balancing, mainly because the target device characteristics are not considered. To address these issues, Ramhorst et al. [212] introduce an FPGA resource-aware structured pruning algorithm capable of capturing, during training, the underlying mapping of network weights to computational and memory resources.

To enhance inference speed, Plochaet et al. [213] propose a pruning method suitable for FPGA-based AI accelerators in which the hardware constraints are considered.

Sui et al. [214] concentrate on row pruning to introduce a hardware-friendly pruning technique. They eliminate all rows except one for each convolution kernel and skip all zero calculations during FPGA deployment.

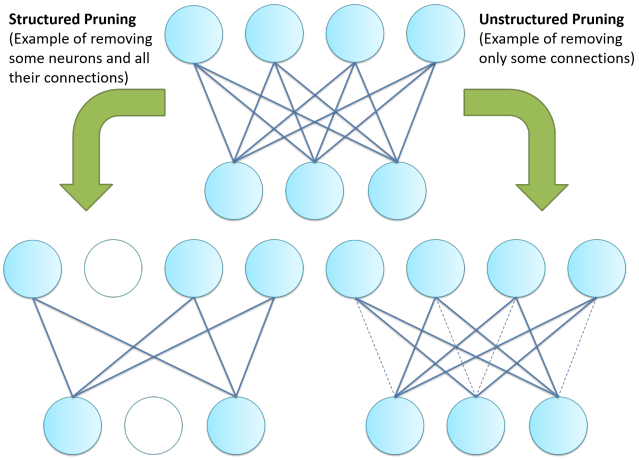


FIGURE 9. Example of applying structured and unstructured pruning on a fully connected layer. Removed (pruned) neurons and weights are shown by white circles and dotted lines, respectively.

When developing real-time object detection systems on FPGAs, pruning can be adopted to enhance throughput, reduce latency, optimize resource utilization, and improve power efficiency, albeit with a potential trade-off in accuracy [47], [73] (cf. Section V-A).

b: Quantization

Data quantization is another commonly employed method to decrease the size of CNN-based object detection models. This approach involves replacing conventional 32-bit floating-point weights and activation data with simpler representations, such as lower-bit floating-point or fixed-point numbers.

Therefore, to achieve this objective, FPGAs give the designer the maximum flexibility for choosing an arbitrary number for bits in the arithmetic operations and data representation.

Quantization can be deployed in both training and inference phases. However, despite its great achievements in

training [244], [245], the majority of recent research on quantization has primarily concentrated on inference [236]. Often combined with pruning, quantization plays a crucial role in achieving an efficient hardware implementation. It reduces the required memory capacity and bandwidth while simplifying arithmetic operations. Quantization can be applied to both weights and activations. However, adjusting the bit width of weights generally has a smaller impact on accuracy compared to modifying the bit width of activations [205].

Data quantization can be applied *uniformly* [246], [247] or *non-uniformly* [248]–[251]. In the former, quantization levels are uniformly spaced, whereas in the latter, they do not necessarily need to be uniformly spaced. Uniform quantization is more popular due to its simplicity and efficient mapping to hardware [236]. For instance, in a successful effort to develop FPGA-based real-time object detection, Nguyen et al. [15] shows the effectiveness of adopting uniform quantization for activations. This approach enhances the design's speed and power efficiency by eliminating the need for external memory access. Detailed results are provided in Sections V-A and V-B.

However, the dynamic range of data across various layers in a CNN tends to be large. Consequently, using uniform quantization with a fixed-point data format for all layers may result in significant performance degradation [199].

On the other hand, non-uniform quantization offers the potential for higher accuracy within a fixed bit-width framework. This is because it allows for more effective capture of distributions, emphasizing crucial value regions and identifying optimal dynamic ranges [236].

One possible and hardware-efficient approach is using mixed-precision quantization, in which different bit precision is considered for each layer of CNN based on the sensitivity of that layer to quantization. This can address the accuracy loss problem in object detection systems, which commonly happens in low-precision quantization, particularly below 8-bit [252], [253]. There exist successful examples of adopting this approach in developing efficient real-time object detection systems on FPGAs [77], [82] (cf. Section V-A). However, finding an optimum solution to decide on the best precision of each layer is an open challenge [236].

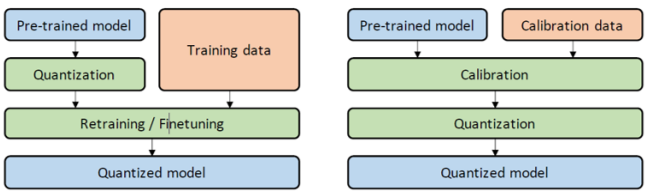


FIGURE 10. Illustration of Quantization-Aware Training (QAT) (Left) and Post-Training Quantization (PTQ) (Right) procedures. In QAT, a pre-trained model is quantized and then fine-tuned. In PTQ, a pre-trained model is calibrated using calibration data to do quantization based on the calibration result. (Figure adapted from [236].)

Data quantization can lead to a drop in model accuracy, which is not acceptable in some applications, such as hard real-time object detection systems. To mitigate this issue, the model parameters should be tuned or adjusted when applying quantization. In this regard, Quantization Aware Training *(QAT)* and Post-Training Quantization *(PTQ)* are two main approaches commonly used to achieve a more accurate and efficient quantization [236]. A pre-trained model undergoes quantization in the former, followed by fine-tuning using training data. This process adjusts parameters and aims to recover any accuracy degradation caused by quantization [236]. However, the retraining procedure can be time-consuming, especially when it comes to low-precision quantization. In PTQ, on the other hand, there is no need to model retraining. A pre-trained model is calibrated using calibration data, which is typically a small subset of training data. This calibration process computes the clipping ranges and scaling factors. Clipping ranges define the upper and lower bounds within which input data are constrained while scaling factors adjust the dynamic range of real-valued input data to adapt them to the desired output range. Subsequently, the model is quantized based on the calibration results [254]. Even though the process of obtaining the desired model is faster in the PTQ approach, this often results in lower accuracy compared to QAT [236], which may not be desirable in hard real-time applications, like object detection, where accuracy is critical. Figure 10 shows the overall procedure taken in QAT and PTQ.

In implementing object detection models on FPGAs, quantization is employed in various forms with different levels of precision, especially when real-time performance is required [15], [47], [48], [77], [193] (cf. Sections V-A and V-B). Like Pruning, the primary goals of deploying this technique in such systems are improving throughput, latency, resource utilization, and power efficiency, with possible consequences in accuracy drop.

c: Knowledge Distillation

Knowledge distillation is the process of transferring knowledge from a large and complex model or ensemble of models to a single smaller model, which can be feasibly deployed under real-world constraints.

The deployment of large object detection models poses a significant challenge, particularly for edge devices like FPGAs, which have restricted memory and computational resources. To address this challenge, a model compression method was initially proposed [221] to transfer knowledge from a large model into a smaller model without sacrificing performance. This process of training a small model from a larger one was formalized as a ''Knowledge distillation'' framework by Hinton et al. [222].

In knowledge distillation, as depicted in Figure 11, a small ''student'' model is trained to emulate a large ''teacher'' model. By leveraging the knowledge from the teacher, the student model aims to improve its accuracy.

Based on the discussions thus far, simplified models obtained through pruning and quantization techniques can be considered student models. By deploying knowledge distillation, we can bridge the accuracy gap between the simplified and original models, bringing them closer in performance.

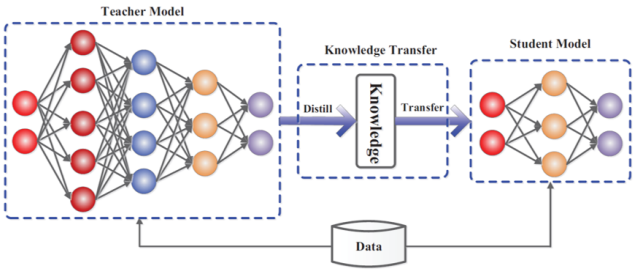


FIGURE 11. An illustration of performing knowledge distillation with a teacher-student framework. (Figure adapted from [255].)

In [223], for instance, a distillation technique tailored for quantized models was proposed, based on which quantized student networks attain accuracy levels comparable to their full-precision teacher model counterparts, with lower inference time.

In the context of real-time FPGA-based object detection, knowledge distillation is deployed to enhance the system's accuracy, especially after applying model compression techniques such as pruning and quantization [224] (cf. Sections V-A).

d: Hardware-aware Neural Architecture Search

When considering resource-constrained platforms like FPGAs for developing DNN-based systems, optimizing the DNN model to be streamlined and compact is crucial. This optimization should encompass reducing the number of parameters and computations while maintaining acceptable accuracy. However, achieving an efficient network tailored to the characteristics of the target hardware is a challenging and time-consuming endeavor [256].

Hardware-aware Network Architecture Search *Hardware-aware NAS* or *(HW-NAS)* seeks to automate the process of discovering the most optimal architectures and configurations of a DNN, tailored specifically for a given hardware platform [227]. This process aims to strike a balance between accuracy and performance, ensuring acceptable tradeoffs [227]. Furthermore, through the consideration of the distinctive attributes of the target device and the implementation of multi-objective optimization algorithms, HW-NAS can generate hardware architectures that are inherently more compatible with the target system and more efficient in terms of performance and resource utilization [228].

HW-NAS has notably influenced image classification and object detection tasks by consistently achieving state-of-the-art results [228]. Using reinforcement learning [257], Jiang et al. [229] introduced an FPGA-aware NAS technique, called FNAS, tailored to identify architectures that can meet specified inference latency requirements. By employing a performance abstraction model, it can estimate neural network latency without the need for extensive training. FNAS effectively eliminates networks that do not align with the constraints in the search space, boosting search efficiency by a remarkable factor of 11.13×.

HW-NAS can effectively optimize all critical metrics of FPGA-based object detection systems, including throughput, accuracy, latency, resource utilization, and power efficiency. By leveraging the flexibility of FPGA designs alongside the efficiency of HW-NAS, the optimization process for object detection models becomes significantly more streamlined and impactful. This synergy enables the development of low-latency models, which are essential for achieving real-time performance [192] (cf. Section V-A).

2) Implementation-related Techniques for Hardware Acceleration

a: Data Reuse

An object detection model often requires access to a large amount of data, usually stored in external memory when the target platform is a resource-constrained device. Particularly with complex and large state-of-the-art object detection models, this strong dependency on access to external memory and data movement can lead to a bottleneck for performance, energy, and computation efficiency [149], [258].

It is possible to reduce memory access by reusing prefetched or intermediate data, such as feature maps, weights, and convolution internal results, multiple times [258]. This can be achieved by leveraging the available on-chip memories of the target device or optimizing the algorithms. Also, convolution operations can be accelerated by maximizing data reuse [259]. This method involves utilizing pixels computed simultaneously at the same spatial position across various Output Feature Maps (OFMs). As a result, all pixels from Input Feature Maps (IFMs) and kernel data are accessed only once and stored in an on-chip BRAM of the FPGA (cf. Section III-A1) until they are reused.

By efficiently employing loop transformation and BRAM-based on-chip buffers to leverage data locality, Beric et al. [260] demonstrated that it is possible to achieve an 11× speedup compared to the standard implementation on similar FPGA resources while also being more energy-efficient due to reduced memory access.

Data reuse methods can be categorized into temporal and spatial reuse [149], both of which are critical for optimizing FPGA-based real-time object detection systems. In temporal reuse, one group of data is utilized multiple times by a single computational unit, allowing on-chip buffers to store a small set of required data, which minimizes memory access latency and power consumption. In spatial reuse, one set of data is simultaneously employed by different processing units, enabling parallelism and improving throughput without requiring additional buffering. These techniques are essential for maximizing the efficiency of FPGA implementations in real-time object detection applications. There are plenty of works in the context of FPGA-based real-time object detection adopting data reuse to achieve higher performance in terms of throughput and latency along with gaining more power efficiency [15], [82], [193] (cf. Section V-A).

b: Mathematical-based Optimization

To enhance performance by reducing computational complexity or memory access, various computation or transformation techniques can be employed, such as the Winograd transform [261], Fast Fourier Transform (FFT) [262].

Since its initial application in accelerating convolution execution in 2016 [263], the Winograd technique has garnered considerable attention in CNN implementations. Regarding FPGA-based implementation, there have been significant research achievements in neural network optimization based on the Winograd algorithm [217]–[220].

Winograd aims to reduce the number of multiplications using some pre-calculated values, considering the fact that the trainable parameters of a CNN model will remain fixed after training. In fact, it transforms overlapping kernels into non-overlapping ones to reduce the computation complexity of CNNs [261], [263]. As an example, Equation 3 shows that when convolving an input feature map I_n and a kernel W_n, both of size 1×3, the traditional approach requires 6 multiplications and 4 summations.

$$\begin{bmatrix} I_0 & I_1 & I_2 \\ I_1 & I_2 & I_3 \end{bmatrix} \begin{bmatrix} W_0 \\ W_1 \\ W_2 \end{bmatrix} = \begin{bmatrix} O_0 \\ O_1 \end{bmatrix} \tag{3}$$

However, adopting the Winograd approach, the number of multiplication and summation will be 4 and 12, respectively, as shown in equations (4) and (5).

$$\begin{aligned} M_1 &= (I_0 - I_2)W_0 \\ M_2 &= (I_1 + I_2)\frac{W_0 + W_1 + W_2}{2} \\ M_3 &= (I_2 - I_1)\frac{W_0 - W_1 + W_2}{2} \\ M_4 &= (I_1 - I_3)W_2 \end{aligned} \tag{4}$$

The key insight in these calculations is that since the weights are fixed, the summations of those weights in Equation 4 can be pre-computed, so we can avoid 4 additions. Additionally, division by 2 only requires a shifter rather than any arithmetic logic.

$$\begin{bmatrix} O_0 \\ O_1 \end{bmatrix} = \begin{bmatrix} M_1 + M_2 + M_3 \\ M_2 - M_3 - M_4 \end{bmatrix} \tag{5}$$

Therefore, the actual number of multiplications and adders required to implement the convolution in equation (4) are 4 and 8, respectively. This shows that employing this method reduces the number of multipliers from 6 to 4 for the same calculation compared to the traditional method. It is important to note that the hardware implementation of multiplication is typically more resource-intensive than that of summation, which justifies the increase in the number of required adders from 4 in Equation 3 to 8 for the implementation of Equation 4. Furthermore, in FPGAs, addition can be performed faster than multiplication [258].

However, the Winograd method limits the reuse of the weights because the pre-computed weight sets in Equation 4 are just used once while moving the convolution window. This issue can be partially addressed by adopting appropriate considerations during the design, such as integration of the pooling with convolution, to leverage the benefits of the Winograd method in reducing energy consumption and increasing throughput [144].

Cai et al. [68] demonstrate the effectiveness of employing this technique to develop an efficient FPGA-based object detection system, optimizing both speed and energy consumption. Further details about this effort are provided in Section V-A.

Another possible acceleration technique to perform convolution operations is using FFT [149], [262]. This method is similar to the Winograd transform and can reduce the number of required multipliers. The principle behind this approach is that convolution in the time domain can be transformed into multiplication in the frequency domain. Initially, we compute the FFT of both the weight and input. Subsequently, we obtain the output by taking the inverse FFT of their element-wise multiplication in the frequency space. As a result, there is a possible reduction in the number of multiplications for each input channel, strengthened from the order of $RSPQ$ to $PQlog2(PQ)$, where $P \times Q$ represents the output size and $R \times S$ signifies the filter size [149].

However, the advantages of this approach diminish with larger kernel sizes [205]. Additionally, combining FFT with sparsity, which often yields higher benefits, can be challenging [149]. Given that modern CNNs predominantly utilize small kernel sizes, FFT has declined in significance in modern hardware accelerators [144].

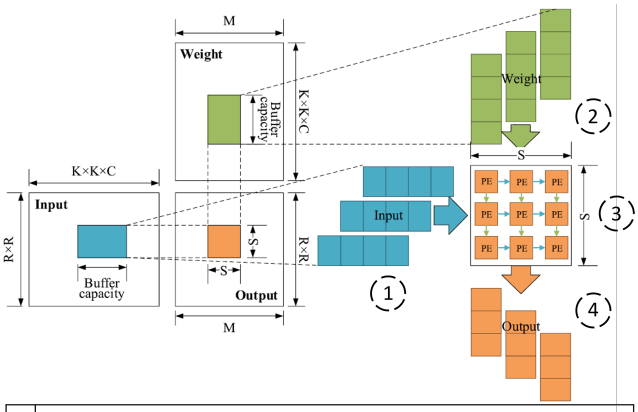
c: Systolic Arrays

First introduced by Kung and Leiserson [202], systolic arrays are parallel computing architectures comprising a network of Processing Elements *(PEs)* organized in a regular grid, typically with a two-dimensional structure. In these arrays, data flows rhythmically between neighboring PEs, enabling efficient and synchronized computations. Some of the key features of the systolic array design include regularity, reconfigurability, and scalability [264].

When data is fetched from memory, it is sequentially passed from one PE to another, enhancing data reuse while minimizing memory access—a key advantage for power-efficient, real-time systems. Furthermore, systolic arrays demonstrate exceptional adaptability to a variety of DNN-based models and are capable of achieving high levels of parallelism and clock frequencies [208]. These attributes make them an ideal choice for developing real-time object detection systems.

Considering its advantages, such as scalability and flexibility, systolic array architectures have been adopted to implement different computations on FPGAs [225], [226]. Figure 12 represents an example of using systolic array-based architectures for performing matrix multiplication. The regular and predictable architectures of systolic arrays make them highly suitable for hardware implementation. In addition, conven-

tional approaches to data spatial reuse in FPGA computation units often encounter challenges such as significant fan-out and difficulty in meeting timing constraints. On the other hand, each PE in a systolic array architecture can seamlessly transmit result data to its neighboring PEs. These localized connections between PEs effectively diminish the necessity for transmitting extensive data over extended distances, thereby enhancing system performance and reducing latency.



1	Each element of the input matrix rows is fed and propagated horizontally across PEs on each clock cycle, with a one-clock-cycle delay for each new row.
2	Each element of the Weight matrix columns is fed and propagated vertically across PEs every clock cycle, with a one-clock-cycle delay for each new column.
3	Each PE performs partial multiplications and accumulates the results over successive clock cycles, processing data as it flows horizontally and vertically
4	Each row of the result exits vertically, with a one-clock-cycle delay from the previous row.

FIGURE 12. An example of adopting a systolic array to perform matrix multiplications in CNNs. (Figure adapted from [265])

d: Code Modification

To maximize hardware usability by optimizing object detection models on the operator level, some hardware-specific code transformation techniques can be deployed. These techniques include loop optimization and operator fusion.

Loop optimization techniques, such as loop unrolling and loop tiling, can improve resource utilization efficiency and data reusability. This results in lower latency and increased energy efficiency by reducing the frequency of off-chip memory accesses. [264]. Given that matrix multiplication serves as a main computational component within CNN-based object detection models, it can be conceptualized as nested loops. Therefore, loop modification techniques are frequently employed in hardware accelerators to optimize performance [211], [266]–[268].

when designing real-time object detection systems on FPGAs, techniques such as loop reordering, loop unrolling, and loop pipelining can be employed to accelerate execution. Additionally, loop tiling is often utilized to optimize memory allocation, improving overall system efficiency and performance [144], [269].

Loop reordering aims to minimize redundant memory accesses and maximize the utilization of on-chip memory. It involves rearranging the sequence of uncoupled multiplication and additional operations within the convolution process to optimize efficiency [269], [270]. Loop unrolling is a technique where multiple copies of the loop body are created to allow some or all iterations to execute in parallel, improving data access and throughput. This technique can be deployed to optimize hardware utilization, reducing computation time effectively and maximizing data reuse by fully leveraging internal memory [266], [267]. Loop pipelining is a technique that allows the overlap of iterations in a loop by starting a new iteration before the previous one completes, enhancing execution efficiency and throughput. However, whether pipelining is feasible depends on the hardware accelerator's architecture (cf. Section IV-A). The concept of loop tiling, or loop blocking, involves dividing the loop's iteration space into smaller blocks to enhance memory hierarchy efficiency [269]. By doing so, the loop's data fits within the cache until it is reused, thus reducing cache misses and enhancing performance significantly.

In the context of the FPGA-based real-time object detection, adopting these techniques is popular (cf. Section V-A). Nguyen et al. [15] use loop reordering and tiling to optimize the data path of the deployed streaming architecture. To enhance data reuse, thereby optimizing memory access, Babu et al. [90] deploy loop tiling. Additionally, they leverage loop pipelining to enhance system throughput. For a different purpose, Fan et al. [47] employ loop unrolling and loop reordering to tackle the issue arising from the distinct kernel sizes of depth-wise and point-wise convolutions. This difference makes it difficult to directly utilize processing elements *(PEs)* designed for depth-wise convolution with point-wise convolution.

The operator fusion technique, also known as kernel or layer fusion technique, aims to reduce data exchanges between computation units and external memory by finding data dependencies between different parts of an object detection model [271], [272]. This process begins by partitioning the model into groups of operators to be fused. Within each group, intermediate feature maps are promptly accessible to subsequent units or layers that need the data. However, if none of the computation units in a group require the data at that time, it is transferred to external memory. It will then be reloaded into on-chip memory when another computation unit needs it. In designing FPGA-based object detection systems in particular, employing the operator fusion technique can offer significant advantages in improving the performance of the system and reducing memory access overhead [73], [76], [92]. For example, in their proposed FPGA design for real-time object detection based on YOLOv2 [71], Xu et al. [76] merge different components of the model, including convolution, batch normalization, and activation function units, to minimize memory access. This approach enables them to achieve a high throughput of up to 71 FPS for the system (see Section V-B for further details).

e: Roofline-based Optimization

When deploying an object detection model on a particular platform, such as an FPGA device, the achievable performance of the model is influenced by the compatibility between the model and the target platform [258]. In this regard, performance bottleneck analysis is a critical task that can be conducted using various methods [219], [230]. Among these methods, the ''Roofline Model'' [273] stands out as particularly popular. The Roofline model is a visual performance model that enables the estimation of the gap between the actual performance (in terms of computational performance (FLOPS) and memory bandwidth (Bytes/s)) of a compute kernel compared to the peak theoretical performance of the underlying system.

A more detailed discussion on the Roofline Model is presented in AppendixB.

In recent years, the Roofline model has been deployed to optimize the CNN implementations on FPGAs in various works [211], [231]. Some of these efforts have focused on using this technique combined with the High-Level Synthesis (HLS) design approach [274]–[276]. Calore et al. [277] optimized the neural network compiler through pipeline design and utilized the Roofline model to investigate the trade-off between memory and computational throughput, aiming to enhance FPGA performance. In particular, to implement real-time object detection models on FPGAs, many works have adopted this technique to find possible performance bottlenecks and optimize the design [47], [278] (cf. Section V-A).

f: Pipelining

In designing FPGA accelerators for CNN-based object detection, pipelining is advantageous when two adjacent convolutional iterations can be performed without data dependency [15], [47], [48], [232]. This allows the next convolutional operation to commence before the current one is fully completed, enhancing computational power.

Generally speaking, pipelining is a design technique aiming to enhance the maximum clock frequency and throughput of synchronous digital systems. This technique involves inserting registers or memory elements into the dataflow path to break down a large operation into several smaller ones [258]. In that way, each small operation requires less time, reducing the path length that a signal must traverse within a clock cycle and thereby enhancing the working frequency. Furthermore, smaller operations can be executed in parallel, leading to improved data throughput [258].

Pipelining proves particularly beneficial when processing a stream of data. In a pipelined circuit, various stages of the pipeline can handle different input stream data simultaneously within the same clock cycle. This concurrent processing enhances data processing throughput significantly [279], at the cost of more resource utilization and higher latency (cf. Section IV-C).

As a result, pipelining has become an indispensable operation in the design of most computational engines in the context of FPGA-based real-time object detection systems [14], [15], [74], [83], [258]. For example, Nguyen et al. [15] propose an FPGA-based streaming architecture tailored for real-time object detection showing high throughput up to 1877 GOPS, where all convolutional layers are fully pipelined (cf. Section V-A).

C. IMPACT AND TRADE-OFFS OF ACCELERATION TECHNIQUES IN FPGA-BASED OBJECT DETECTION

It is crucial to understand the potential effects of each technique, reviewed in Section IV-B, on system performance and efficiency when developing a real-time object detection system using FPGAs. This insight allows designers to select the most appropriate set of techniques, taking into account the requirements of the desired system and application, as well as the limitations and capabilities of the target FPGA device.

Metrics / Methods (Section)	Throughput	Accuracy	Latency	Resource Utilization	Power Efficiency
Pruning (IV-B.1a)	↑	↓	↑	↑	↑
Quantization (IV-B.1b)	↑	↓	↑	↑	↑
Knowledge Distillation (IV-B.1c)	=	↑	=	=	=
HW-NAS (IV-B.1d)	↑	↑	↑	↑	↑

Metrics / Methods (Section)	Throughput	Accuracy	Latency	Resource Utilization	Power Efficiency
Data Reuse (IV-B.2a)	↑	=	↑	↓	↑
Mathematical-based Optimization (IV-B.2b)	↑	↓	↑	↑	↑
Systolic Arrays (IV-B.2c)	↑	=	↑	↓	↑
Code Modification (IV-B.2d)	↑	=	↑	↓	↑
Roofline-based Optimization (IV-B.2e)	↑	=	=	↓	=
Pipelining (IV-B.2f)	↑	=	↓	↓	=

FIGURE 13. Possible impact of model-related (top) and implementation-related (bottom) acceleration techniques on key metrics in FPGA-based object detection systems, focusing on throughput and accuracy. The corresponding section where each technique is discussed in this work is also indicated.

Figure 13 highlights the possible impacts of different acceleration techniques on FPGA-based object detection systems, focusing on key metrics including throughput, accuracy, latency, resource utilization, and power efficiency. Each technique can have a positive effect (indicated by green arrows), a negative effect (indicated by red arrows), or a neutral effect (indicated by an equal sign).

Regarding model-related techniques, pruning effectively increases throughput, reduces latency, improves resource utilization, and enhances power efficiency, although it may lead to a decrease in accuracy. Quantization shows similar benefits, improving throughput, latency, resource utilization, and power efficiency, with a potential drop in accuracy. Knowledge distillation, on the other hand, offers improvements in

accuracy, making it a suitable compensator for previous techniques. Finally, HardWare-aware Neural Architecture Search *(HW-NAS)* can be employed to optimize all key metrics, leading to enhancements in throughput, accuracy, latency, resource utilization, and power efficiency. This positions HW-NAS as the most comprehensive acceleration technique for FPGA-based object detection systems, albeit with the caveat of its complex and time-consuming implementation process.

In the context of implementation-related techniques, data reuse enhances throughput and reduces latency by reusing intermediate results to avoid redundant computations. This technique tends to increase resource utilization due to higher memory requirements but improves power efficiency, with no impact on accuracy. Mathematical-based optimization methods can significantly improve throughput and power efficiency by simplifying calculations. However, these approaches might compromise accuracy because of approximations, although they can optimize resource utilization. Using systolic arrays can improve throughput and power efficiency by enabling parallel processing with a consistent data flow. While accuracy is maintained, this method may require more resources due to the dedicated hardware involved. Code modification, including techniques like loop unrolling, boosts throughput and power efficiency without affecting accuracy but may increase resource utilization and, in some cases, latency. Roofline-based optimization strikes a balance between computational load and memory access to enhance throughput, with no impact on accuracy but potentially higher resource demands and trade-offs in power efficiency. Pipelining improves throughput by allowing concurrent execution of multiple stages, thereby improving throughput. Although it does not affect accuracy, the additional hardware requirements for pipelining can lead to increased resource utilization and latency, with minimal effect on power efficiency.

It is important to note that these effects and results are not definitive and may vary with different implementations and deployments. However, the mentioned effects can be considered as the most anticipated outcomes. All in all, selecting the appropriate acceleration method involves weighing these trade-offs to achieve the desired performance while considering system constraints.

V. RESULTS ANALYSIS AND OPTIMIZATION

This section aims to review several case studies, illustrating how acceleration architectures and techniques discussed in Section IV have been applied to develop real-time object detection systems, along with an analysis of their outcomes. Then, recent advancements and works in this field are compared across multiple metrics—such as throughput, accuracy, pixel throughput, and power efficiency—highlighting state-of-the-art results and revealing trends in FPGA-based real-time object detection. Finally, optimization strategies specifically aimed at achieving real-time performance in FPGA-based object detection systems are briefly discussed.

A. CASE STUDIES: ADOPTION OF ACCELERATION TECHNIQUES IN FPGA-BASED REAL-TIME OBJECT DETECTION

This section discusses examples of applying the aforementioned acceleration techniques in developing real-time object detection systems on FPGAs. The works are chronologically reviewed to illustrate the progress in this field. The results of adopting these techniques are presented and analyzed to demonstrate how they have contributed to enhancing the performance, accuracy, and efficiency of such systems. Additionally, an overview of the overall architecture proposed by some recent works is briefly examined to provide context on the approaches adopted in state-of-the-art object detection solutions.

Fan et al. (2018) [47] demonstrated the effectiveness of channel pruning in boosting the performance and hardware efficiency of FPGA-based real-time object detection systems. By pruning up to 10% of the channels of a customized SSD model [19], they achieved a model 3 times smaller with a minimal accuracy loss of just 1.8%. Also, by proposing a partial quantization scheme, they minimized the accuracy drop caused by quantization. This quantization scheme preserves 32-bit precision for certain components like Pre- and post-processing modules while quantizing most layers in the feature extractor to 8 bits. They also used 8-bit weights and activation in their proposed system. By employing the roofline model technique tailored for the target platform, i.e., a Xilinx Zynq ZC706 with a DRAM memory bandwidth of 1.2 GB/s, they improved overall performance by caching all intermediate results in the on-chip memory. By applying these techniques, they achieved 64.8 FPS, a latency of 15.43 ms, and a power consumption of 9.9 W using a single computation engine architecture. These results represent a more than sixfold improvement in throughput and latency, along with a 94% reduction in power consumption, compared to running the original model on a TITAN X Pascal GPU.

Nguyen et al. (2019) [15] implemented a real-time object detection system using YOLOv2 [71] on a VC707 FPGA, employing 1-bit weights and flexible 3-to-6-bit activations. This approach led to a 30-fold reduction in model size and a 5.4-fold decrease in activation size. They demonstrated that even with such low-bit quantization, an acceptable accuracy (51.68% mAP) with a high inference speed (66 FPS) is possible with a power consumption of 8.7 W. They developed an efficient streaming architecture tailored for real-time object detection, focusing on optimizing the data path, where all convolutional layers are fully pipelined. Additionally, they enhanced loop computations using loop reordering and tiling techniques, enabling the convolutional layers to run effectively on their resource-constrained FPGA.

They also introduced a data reuse scheme to minimize memory accesses and enhance computation performance by reusing both the input data and the weights (Figure 14). The input is processed in sliding cubes (with $K \times K \times T_i$ pixels) that move across the image in row passes. During each pass, the T_0 weight blocks are reused to perform convolutions,

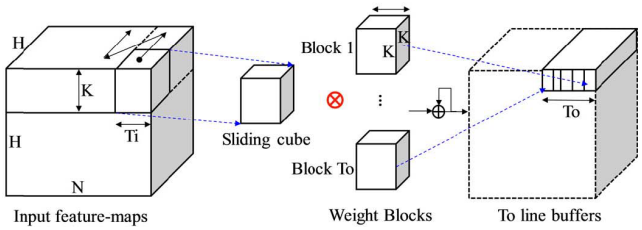


FIGURE 14. **Data reuse scheme proposed in [15], minimizing memory accesses by reusing both input data and weights, with intermediate results stored in line buffers. $K \times K \times T_i$ defines the size of the input sliding cube in pixels, and T_0 denotes the number of weight blocks concurrently convolved with the input sliding cubes in each iteration. (Figure adapted from [15]).**

generating temporary output values that are stored in line buffers. These computations are performed in parallel, and the results are accumulated across multiple row passes. After each pass, the input sliding cube shifts, and new weight blocks are fetched. The convolutional outputs are accumulated in the line buffers, reducing the need for additional memory accesses. The weights are fetched once per row pass (H times in total) and from on-chip SRAM, resulting in fewer memory accesses. Once all rows are processed, the final accumulated output is forwarded to the next layer, minimizing external memory access. The parameters T_i and T_0 are chosen to balance hardware cost and performance, ensuring efficient memory usage and parallel computation.

The results demonstrate approximately a 24% improvement in throughput and a 90% reduction in power consumption compared to running the same model with 32-bit floating-point precision on a GTX Titan X GPU.

Hao et al. (2019) [192] presented a co-search method that enhanced the performance of CNN-based object detection, leading to gains in accuracy, frame rate, and energy efficiency. The proposed approach was tested on an object detection task from the DAC-SDC competition [280], where it outperformed the first-place winner using a PYNQ-Z1 board. This implementation could meet real-time requirements, delivering up to 29.7 FPS and achieving 68% accuracy. Their solution delivered a 6.2% improvement in IoU, reduced power usage by 40%, and was 2.5 times more energy efficient compared to the state-of-the-art FPGA designs. Also, in comparison to GPU-based designs on Nvidia Jetson TX2, their method achieved nearly identical accuracy (approximately 69% IoU) while offering an energy efficiency advantage of 3.1 to 3.8 times.

Wang et al. (2020) [73] also applied an unstructured pruning to YOLOv2 [71] and showed the effectiveness of this technique in implementing real-time object detection on FPGAs. Their approach leveraged hardware-aware optimizations, including model pruning and quantization, to reduce the computational workload and memory footprint of the YOLOv2 network. Using the roofline model as an optimization flow guidance to adopt a fine-grained unstructured pruning scheme, they achieved a 7x reduction in computational workload with only a 2.35% loss in accuracy. This technique, along with applying 8-bit quantization on the whole model and adopting a layer fusion approach, allowed for high-throughput object detection at 61.9 FPS using a single computation engine architecture. They also achieved a high power efficiency of 81.92 GOPS/W on an Arria-10 FPGA, demonstrating the viability of these techniques for real-time object detection.

Chang et al. (2021) [82] showed that adopting a mixed-precision quantization approach is a more effective way to minimize the accuracy loss than applying fixed bit-width quantization. By quantizing weights and activations to 5-8 bits and 7-8 bits, respectively, they developed a real-time object detection system based on YOLOv3 [80] using the ZCU102 FPGA platform. They achieved a throughput of 22 FPS with 49.7% mAP, reflecting only a 1.5% accuracy loss compared to the 32-bit floating-point model.

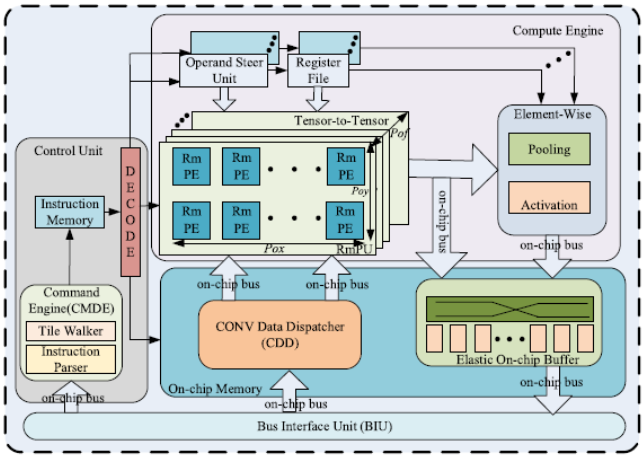


FIGURE 15. **An architecture for accelerating YOLOv3 on FPGA. It is based on a pipelined multicore processing architecture consisting of a Control Unit to manage all operations, a Compute Engine with multiple layers of 2D arrays of PEs, and an on-chip memory unit optimized for parallel data access (Figure adapted from [82], .**

As Figure 15 shows, the proposed architecture features a pipelined multicore unit composed of a computing engine, an elastic on-chip buffer, and a control unit.

The computing engine performs operations like convolution, activation, and pooling with mixed precision This unit consists of Reconfigurable Microprocessing Elements *(RmPEs)* organized in a 3D array to adapt to various convolutional layer requirements. Each RmPE performs parallel MAC operations, using DSP and LUT-based adders. Multiple RmPEs form Reconfigurable Macro Processing Units *(RmPUs)*, which combine into a 3D array for enhanced parallelism. The Control Unit manages configurations, including bit-widths, activation (e.g., ReLU), pooling types, and dataflow patterns, optimizing resource utilization. The unit supports flexible data processing by adapting to different CNN models through reconfigurable parameters for input/output widths and convolution array setup.

The on-chip buffer unit is a flexible memory system dynamically partitioned into physical banks, optimized for parallel data access, and configured by the Control Unit to match computational needs. It stores intermediate data for convolution, activation, and pooling tasks, reducing off-chip memory dependency and facilitating cross-layer data reuse.

The Control Unit orchestrates the pipelined multicore unit by breaking down convolution tasks into smaller, tile-based subtasks and coordinating the computing engine and on-chip buffer accordingly. It decodes instructions, queues them by type (control, memory access, and computation), and manages synchronized, parallel issuance, ensuring data dependencies are met for efficient execution.

Cai et al. (2021) [68] proposed an FPGA-based approach for real-time underwater object detection based on a single computation engine architecture, utilizing the Winograd algorithm to optimize various convolution operations within the network. They also improved data reuse through a ping-pong-based memory access approach. By adopting these techniques, they were able to achieve 33.14 FPS with MobileNetV3-SSDLite [19] when the accelerator was deployed on a Zynq XC7Z045 device running at 150 MHz. Compared to a CPU (Intel i7-8700), the proposed accelerator offers an 8.7× speedup, enhancing energy efficiency by 60×.

Huang et al. (2021) [193] showed that employing an effective data reuse strategy in FPGA implementations can greatly enhance the performance of real-time object detection by optimizing memory access patterns and increasing data locality. Their approach employed buffering techniques to reduce the frequency of memory fetches, lowering latency and boosting throughput. This strategy proved particularly beneficial for deformable convolutions, where irregular memory access patterns typically limit data reuse. They modify the deformable convolution operation to enhance performance further by restricting adaptive offsets to fixed ranges, enabling more efficient data reuse. Additionally, they replaced the full 3×3 deformable convolutions with 3×3 depthwise deformable convolutions and 1×1 convolutions, akin to the depthwise separable convolution approach used in Xception [281], thereby streamlining computation and enhancing the efficiency of the FPGA implementation. Using their developed object detection model on an Ultra96 board, they achieved up to 32.2 FPS on the Pascal VOC dataset [40] while consuming only 5.6 W of power.

In their proposed workflow for implementing YOLOv4 on a Zynq UltraScale+ MPSoC FPGA, **Liew et al. (2022)** [224] demonstrated how leveraging knowledge distillation can help recover some of the accuracy lost due to pruning and quantization. As shown in Figure 16, the size of the baseline model (with 32-bit floating point precision) was reduced by 56% and 88% after applying channel-wise model pruning and then 8-bit quantization (INT8), respectively. This reduction in model size was crucial for achieving real-time performance at 33.34 FPS, though it came with an 8% decrease in accuracy. However, adopting knowledge distillation led to a recovery of some accuracy by about 2.5%.

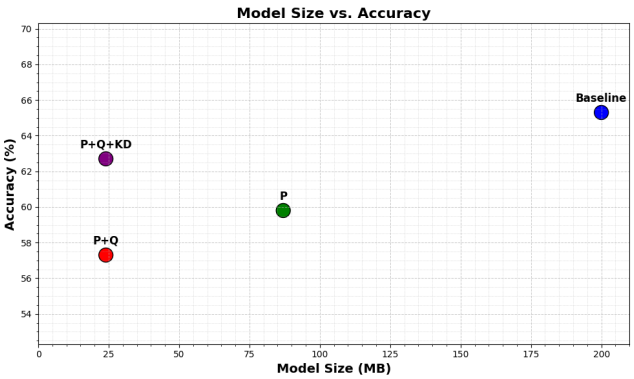


FIGURE 16. The effect of applying pruning (p), quantization (Q), and knowledge distillation (KD) techniques one after the other on the size and accuracy of the YOLOv4 model (the numbers extracted from [224]).

Jain et al. (2022) [77] developed a real-time object detection system based on tiny-YOLOv2 [71], in which a mixed-precision quantization scheme (8-bit weight and 16-bit activation) was adopted. By taking advantage of QAT, they could not only achieve a satisfactory processing throughput (23 FPS) for real-time performance on the XC7Z035 FPGA but also recover the accuracy loss caused by applying quantization to reach a high accuracy of 57.1%. To find the best spatial and temporal unrolling factor for designing an energy-efficient and high-speed architecture, they did a design space exploration using the ZigZag tool, an optimization framework [282].

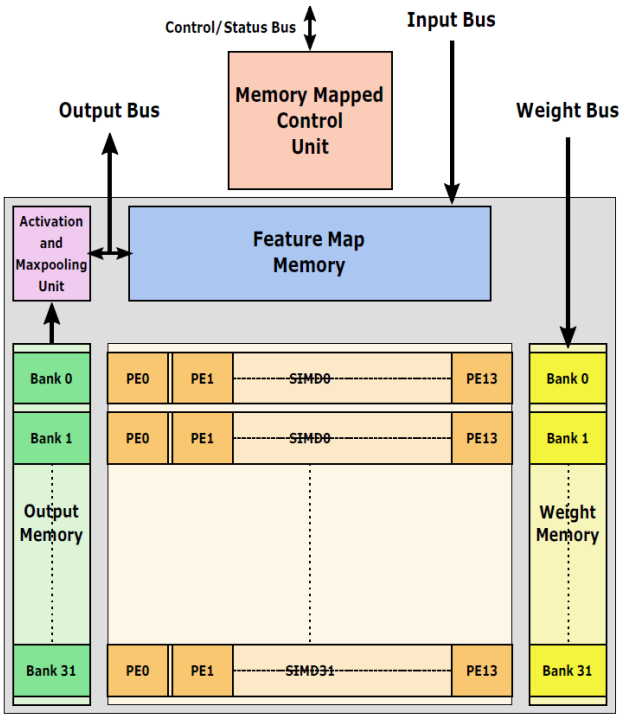


FIGURE 17. Overall accelerator architecture, consisting of a SIMD array, activation, pooling, and control units along with memories to store input, weight, and output values (Figure adapted from [77]).

Subsequently, as Figure 17 shows, they developed an architecture using an array consisting of 32 Single Instruction Multiple Data (SIMD) units, where each SIMD unit consists of 14 parallel PEs. The architecture also includes a 2×2 max pooling unit with strides of 1 and 2, along with an activation unit that performs ReLU and Leaky ReLU. To minimize latency, dual-port memories are used to enable parallel read and write operations. A control unit is also implemented to manage data movement and oversee the operation of the SIMD arrays and other components.

Suh et al. (2023) [69] utilized a low-precision quantization approach to implement the SSD model, trained on a custom drone dataset on the Zynq ZU3EG FPGA. They introduced a uniform quantization scheme with power-of-two (POT) quantization boundaries, termed UniPOT, designed to streamline the model, reduce its size, and eliminate the need for multipliers by substituting them with shift registers. Their results showed that applying UniPOT quantization with 8-bit precision resulted in only a minimal accuracy drop (0.24% mAP) compared to the baseline 32-bit model. Their proposed design achieved 88.42% mAP, an energy efficiency of 79 GOPS/W, and 158 GOPS throughput.

Figure 18 shows the overall proposed architecture. The design utilizes two separate DMA modules to handle read and write operations independently, each with its own descriptor buffer. The read DMA module loads image tiles and weights into designated input and weight buffers. A data router then rearranges the pixel and weight data for optimized reuse in MAC arrays, leveraging FIFOs to minimize buffer reads by shifting and reusing data in registers. This setup enables efficient pixel reuse within the register arrays, reducing memory access. Each Processing Element (PE) in the MAC array performs one multiply-accumulate (MAC) operation per cycle, benefiting from loop unrolling and tiling techniques to maximize efficiency.

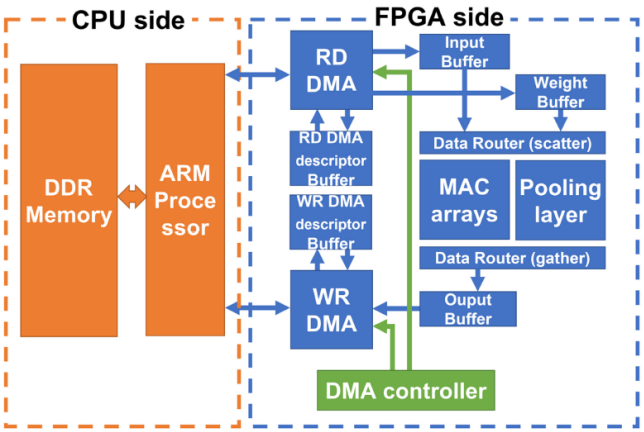


FIGURE 18. **Overall hardware block diagram of a proposed FPGA accelerator, with two separate DMA modules, each equipped with a dedicated DMA descriptor buffer. The data router will rearrange the pixels and weights to maximize data reuse capability (Figure adapted from in [69]).**

Anupreetham et al. (2024) [67] developed a high-performance, real-time object detection system on a Stratix 10 FPGA, achieving low latency and high throughput by introducing a novel pipelined Non-Maximum Suppression (NMS) algorithm. This new NMS design reduced latency by removing the traditional dependency on sequential processing, which typically slows down the object detection pipeline. By allowing continuous dataflow between stages, the pipelined NMS enabled concurrent processing of feature extraction and SSD [19] detection stages, significantly decreasing end-to-end latency. Additionally, they enhanced the system's efficiency by incorporating a multi-threaded NMS module that processes multiple images in parallel, aligning with the throughput capacity of the CNN backbone accelerator and thus preventing stalls.

As a result, the system achieved an end-to-end latency of only 2.13 milliseconds, marking a substantial improvement over prior FPGA implementations. It delivered 5.3× higher throughput than previous solutions with comparable accuracy, i.e., 22.8% mAP on MS COCO [41], demonstrating a significant advancement in balancing speed, efficiency, and accuracy for FPGA-based real-time object detection.

D. Zhang et al. (2024) [86] introduced a fully hardware-accelerated end-to-end object detection system, designed entirely to run on FPGA devices. To meet real-time requirements and minimize system latency, all components, including post-processing algorithms, were implemented on hardware, eliminating the need for a CPU.

To enhance design flexibility and support various object detection models, they adopted a single computation engine architecture. However, to address the primary challenge of this approach—intensive memory access—they employed three dedicated data access units alongside on-chip buffers to improve data reuse. These data access units leveraged burst transfer mechanisms to optimize memory bandwidth utilization and minimize data transfer overhead. Furthermore, the proposed data buffering strategy allowed data transfers to occur in parallel with computations, significantly boosting execution efficiency. A ping-pong buffer mechanism further enhanced data reuse within the design.

The system was evaluated using object detection models such as YOLOv2 and YOLOv3, demonstrating substantial performance improvements. Compared to state-of-the-art implementations, such as [15], [48], [67], it achieved up to 9x higher throughput and up to 5x lower latency.

When tested on a KC705 board, featuring a low-end XC7K325T FPGA, the system delivered an impressive throughput of 401 GOPS, consumed 12 W of power for the entire system, and achieved a frame rate of 65 FPS while running a quantized (8-bit) YOLOv3-tiny model.

B. COMPARATIVE ANALYSIS OF FPGA IMPLEMENTATIONS

Tables 9 and 10 present detailed information on recent studies, sorted based on the achieved pixel throughput, regarding the implementations of real-time object detection models on FPGAs. As shown in Table 9, the majority of these studies utilize two main detection model families: YOLO [18] and

TABLE 9. The recent advanced works of FPGA-based real-time object detection implementations (part 1/2).

Reference	Year	Detection Model	Precision (W,A)	Freq. (MHz)	Accuracy (mAP)(%)	Image Resolution (Pixels)	Frame Rate (FPS)	Pixel Throughput (MPPS)**	Latency (ms)	Throughput (GOPS)	Power (W)	Power eff. (GOPS/W)
Huang et al. [193]	2021	ShuffleNetV2 based	(4,8)	250	55.1	256×256	26.9	**1.76**	40	128	5.6	N/A
Cai et al. [68]	2021	SSDLite-Mobile-NetV3	(16,N/A)	150	56.8	300×300	33.14	**2.98**	N/A	N/A	9.34	N/A
Suh et al. [69]	2023	SSD-VGG16	(8,8)	200	76.2	300×300	34.18	**3.07**	29.25	138	2.4	57.5
Fan et al. [47]	2018	SSDLite-Mobile-NetV2	(8,8)	100	20.5	224×224	64.8	**3.25**	15.43	N/A	9.9	N/A
Jain et al. [77]	2022	Tiny YOLOv2	(8,16)	200	57.1	416×416	23	**3.98**	43.33	161.4	N/A	N/A
K. Kim et al. [191]	2023	SqueezeNet based	(8,8)	100	82.6	192×256	43.95	**4.14**	22.75	N/A	2.11	N/A
S. Kim et al. [46]	2021	SSDLite	(16,16)	200	78.6	224×224*	84.4	**4.23**	11.79	N/A	5.1	N/A
Li et al. [74]	2020	YOLOv2	(8,N/A)	200	73.6	416×416	25	**4.32**	N/A	740	27.2	27.2
M. Kim et al. [84]	2023	Tiny YOLOv3	(8,8)	100	81.19	320×320	76.75	**4.5**	13.03	95.08	2.2	43.16
Babu et al. [90]	2022	YOLOv4	N/A	100	N/A	416×416*	30	**5.19**	11.83	189.14	10.36	18.26
Chang et al. [82]	2021	YOLOv3	(2-8,2-8)	200	49.7	640×480	22	**6.75**	N/A	809	15.3	52.87
D. Zhang et al. [86]	2024	Tiny YOLOv3	(8,8)	200	58.1	416×416	65.7	**11.36**	N15.2	401	7.8	51.4
Nguyen et al. [15]	2019	Tiny YOLOv2	(1,6)	200	51.38	416×416	66.56	**11.51**	N/A	464.7	8.7	53.29
Hosseiny et al. [92]	2023	Tiny YOLOv7	(13,13)	100	34.1	640×640*	30	**12.28**	15	N/A	10.79	N/A
Xu et al. [76]	2021	Tiny YOLOv2	(8,8)	190	56	416×416	71	**12.28**	N/A	500	N/A	N/A
Wang et al. [73]	2020	YOLOv2	N/A	211	74.45	416×416	72.5	**12.54**	N/A	2,130	26	81.92
Pestana et al. [85]	2021	Tiny YOLOv3	(16,16)	143	31	768×576	32.4	**14.33**	30.9	180	3.87	46.51
J. Zhang et al. [48]	2021	Tiny YOLOv2	(8,8)	200	77.04	1280×384	43.7	**21.47**	27.78	464.5	10.25	45.3
Anupreetham et al. [67]	2024	SSDLite-Mobile-NetV1	N/A	400	22.8	640×640*	469	**192**	2.13	N/A	81	N/A

* It is not directly mentioned through the work; therefore, the image size is assumed based on the employed model or dataset.
** Pixel throughput, measured in Mega Pixels Per Second (MPPS), is calculated by multiplying the frame rate by the number of pixels per frame. (The table is sorted based on this parameter.)

TABLE 10. The recent advanced works of FPGA-based real-time object detection implementations (part 2/2).

Reference	FPGA/Board Model	Design Environment	Resource Utilization				Key Employed Techniques
			BRAMs #(size)	LUTs/ALMs (K)	DSP	FFs (K)	
Huang et al. [193]	Xilinx Ultra96	Vivado HLS[a]	216 (18k)	34	360	42	Data reusing, algorithm-hardware co-design
Cai et al. [68]	Xilinx XC7Z045	Vivado	446 (18k)	166	801	159	Wingrad-based, and memory access (using ping-pong operation) optimization
Suh et al. [69]	Xilinx ZU3EG	Vivado	102 (18K)	78	263	75	Low-precision quantization, loop tiling, model pruning
Fan et al. [47]	Xilinx ZC706	Vivado	311 (18K)	148	728	191	Partial quantization, resource sharing, channel pruning
Jain et al. [77]	Xilinx XC7Z035	N/A	462 (18K)	88	448	62	Design space exploration, quantization aware training, tiling strategy
K. Kim et al. [191]	Xilinx ZC702	Vivado	N/A	18	7	21	Model simplification and compression, parallel processing
S. Kim et al. [46]	Intel Arria 10	Quartus Prime	N/A	N/A	980	N/A	Arithmetic-based and memory access optimization, data reusing
Li et al. [74]	Intel Arria 10	OpenCL + RTL	N/A	N/A	410	N/A	Mixed-precision data, parallel design, pipelining
M. Kim et al. [84]	Xilinx Artix-7	Vivado	185 (18K)	50	240	58	Dynamic data reuse scheme, hardware-friendly quantization
Babu et al. [90]	Xilinx XC7Z020	Vivado HLS	115 (18k)	23	174	46	Code modification, memory access optimization
Chang et al. [82]	Xilinx ZCU102	Vivado	680 (18k)	185	2280	308	Mixed-precision quantization
D. Zhang et al. [86]	Xilinx XC7K325T	Vivado HLS	553 (18K)	136.5	687	N/A	Ping-pong buffer-based data reusing, memory access optimization
Nguyen et al. [15]	Xilinx VC707	Vivado HLS	1,026 (18K)	86	168	60	Binary weights, HW-aware quantization, data reusing
Hosseiny et al. [92]	Xilinx XC7Z100	Vivado HLS	1,092 (18k)	59	16	9	Code modification, pipelining, data reusing
Xu et al. [76]	Intel Arria-10 GX1150	OpenCL SDK	1849 (20K)	145	1092	N/A	Scalable pipeline design, layer fusion, SW-HW co-design
Wang et al. [73]	Intel Arria-10 GX1150	OpenCL	2,576 (20K)	188	76	N/A	Fine-grained pruning, Hardware/Software co-Design
Pestana et al. [85]	Xilinx XCKU040	N/A	384 (18K)	139	839	N/A	Post-training dynamic quantization
J. Zhang et al. [48]	Xilinx ZC706	N/A	256 (18k)	84	610	65	Resource sharing, pipelining, arithmetic-based optimization
Anupreetham et al. [67]	Intel Stratix 10 GX2800	Quartus Prime Pro	7,883 (20k)	N/A	5,009	N/A	Pipelining and parallel processing, code modification

[a] Vivado offers a complete design environment for developing and implementing FPGA designs, while Vivado HLS facilitates high-level synthesis, converting C/C++ code into RTL code.

SSD [19] (cf. Section II-A2). These one-stage detectors are favored due to their balanced accuracy-speed performance, which is crucial for developing real-time object detection systems. Additionally, there are many lightweight variants of these models suitable for resource-constrained devices like FPGAs.

As a common technique employed in this context, different quantization regimes have been adopted based on the available resources on the target FPGA device and the accuracy required by the application of interest. The best-achieved accuracy, processing speed (in FPS), and latency, as the most relevant metrics for evaluating real-time object detection systems, are provided for each study. It is important to note that the image resolution can affect system performance—the larger the image size, the higher the achievable accuracy with lower speed. Therefore, looking at the achieved pixel throughput may help make a more meaningful and fair quantitative comparison between the existing works. For this reason, the reviewed works in Table 9 and 10 are sorted based on this metric. Additionally, Figure 19 shows a comparison of the reviewed works based on their pixel throughput performance. Another critical metric is the achieved computational power, or throughput, measured in Giga Operations Per Second *(GOPS)*. While FPS reflects the processing throughput from a visual perspective, GOPS represents it from an architectural standpoint. Knowing the working Frequency *(Freq)* is also essential for a meaningful GOPS comparison. In addition, power consumption and power efficiency are critical factors, especially when developing systems for edge devices. Figure 20 shows the achieved power efficiency of the reviewed works (if reported in the corresponding manuscript) to provide a visual comparison and facilitate analysis.

Table 10 specifically highlights the deployed FPGA devices and the main development tools used in each work. All studies here have relied on either AMD (formerly Xilinx) or Intel (Altera) FPGA devices. This table also provides a report on resource utilization and mentions the primary acceleration techniques adopted in each study.

C. OPTIMIZATION STRATEGIES FOR REAL-TIME PERFORMANCE

To develop a real-time object detection system on resource-constrained devices such as FPGAs, it is essential to focus on key metrics like throughput, latency, and accuracy. A detailed review of these metrics within the context of this paper is provided in Section II-C2.

As outlined in Section II-A, deep neural network (DNN)-based models often provide the highest accuracy for object detection tasks. However, these models are computationally intensive, necessitating a careful balance between accuracy and execution time, as discussed by Castells et al. [51]. This balance becomes even more critical in hard real-time systems, where meeting stringent timing requirements is essential for safety and reliability.

Before adopting optimization techniques, discussed in Section IV-B, it is crucial to select the appropriate hardware architecture. Section III-C introduces two FPGA architectures popular for developing object detection systems: single computation engine and streaming architectures.

Among these, streaming architectures are more suitable for real-time object detection due to their reduced memory accesses, which enhance system speed and lower latency. These architectures are also optimized to perform specific tasks within each block, leveraging parallelism to boost performance in object detection models [197]. However, this level of optimization typically comes with higher hardware costs compared to one-size-fits-all configurable engines.

The rapid development of deeper and more powerful object detection models has shifted the focus from achieving high accuracy to optimizing the performance of these models on resource-constrained devices like FPGAs. Modern object detection models tend to be computationally demanding and require larger memory footprints, making performance optimization a primary challenge.

To address these challenges, the typical workflow for developing a real-time object detection system on FPGAs involves simplifying and optimizing the chosen model. In this regard, many works show the advantages of employing Binarized Neural Networks *(BNNs)* [283] and Ternary Neural Networks *(TNNs)* [284] to achieve real-time inference [15], [285]–[288]. BNNs [289] and TNNs [284] are two types of neural networks in which the weights and activations are quantized into two (-1,1) and three values (-1,0,1), respectively.

However, compression and simplification techniques, such as pruning and quantization, can result in accuracy degradation. To mitigate this problem, retraining the compressed models and utilizing knowledge distillation [222] are effective strategies. Mishra et al. [290] propose a method, called *Apprentice*, which combines low-precision arithmetic with knowledge distillation to achieve nearly the same accuracy as the original model. Remarkably, this approach results in less than a 1% accuracy loss compared to a 32-bit floating-point model, even when employing ternary precision within the ResNet architecture [121] on the ImageNet dataset [146].

In addition, a real-world object detection system comprises diverse components, some of which are best suited for execution on a processor due to their sequential nature. Considering this, adopting a hardware-software co-design approach can facilitate the development of an optimal system [67], [73], [291], [292]. This approach may become more effective when combining hardware-aware NAS, discussed in section IV-B. Adopting a co-design approach can enable developers to design efficient object detection systems that satisfy real-time constraints, including latency, throughput, and accuracy, using FPGAs and FPGA-oriented system-on-chip devices [192], [229].

VI. CHALLENGES AND FUTURE DIRECTIONS

A. EXISTING CHALLENGES

Seeking to enhance their accuracy, object detection models are becoming increasingly larger and deeper with more com-

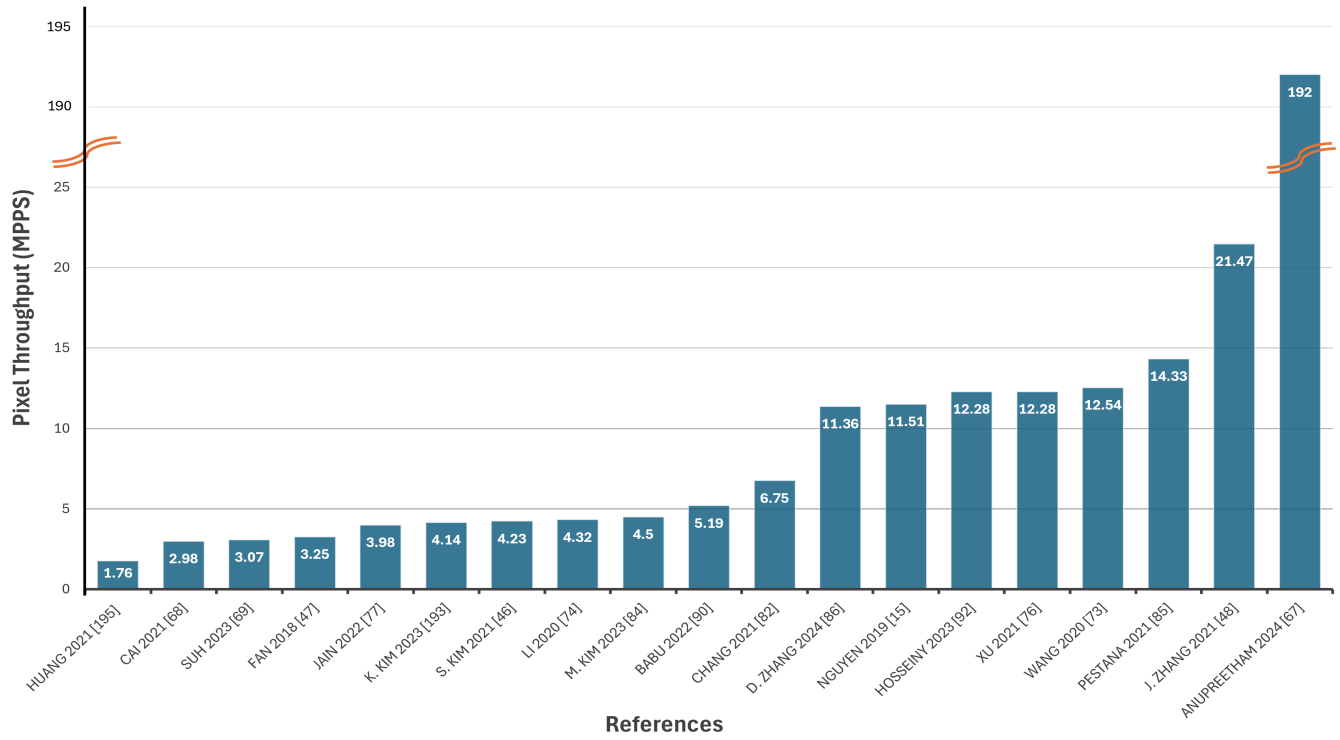


FIGURE 19. Comparison of the reviewed works based on the achieved pixel throughput.

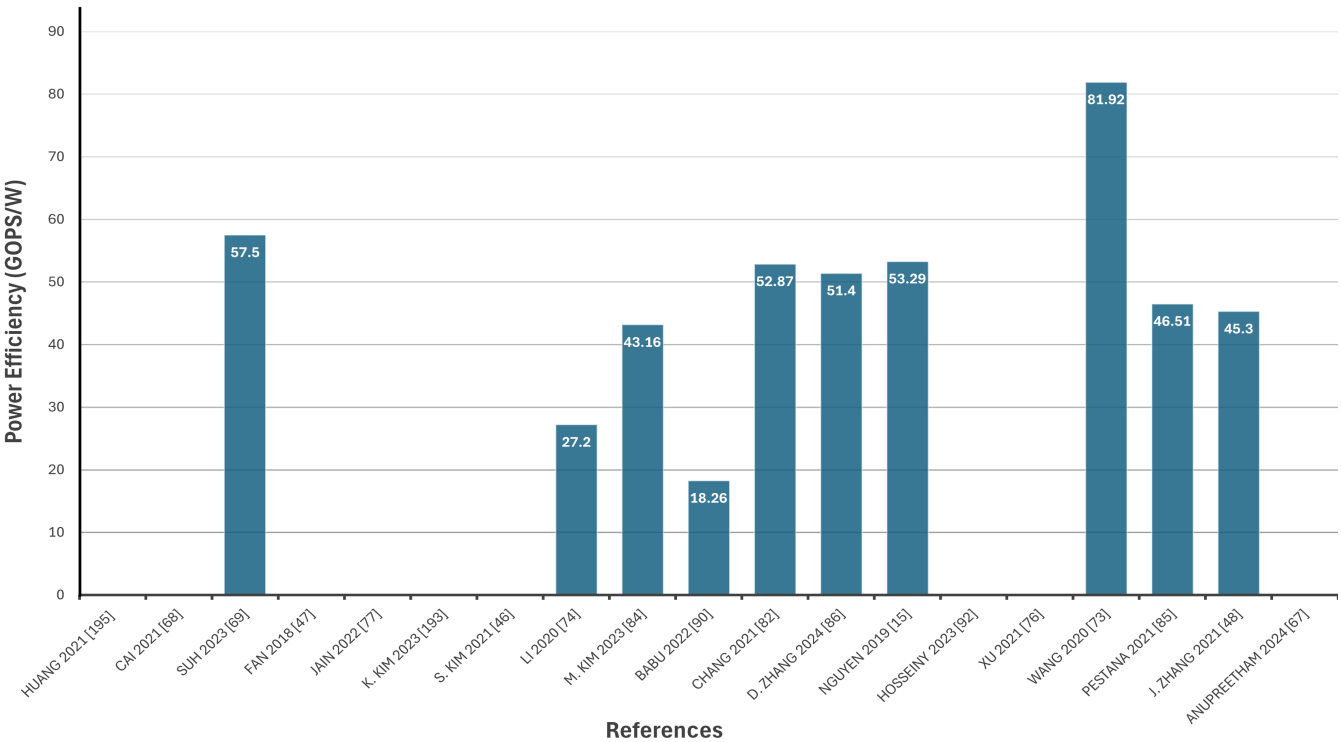


FIGURE 20. Comparison of the reviewed works based on their reported power efficiency (note: for some works power efficiency data is not available).

plex architectures. Consequently, implementing such models on FPGAs is challenging due to their limited memory and computational resources. Furthermore, achieving real-time performance with these complex models is another significant issue. As discussed, model simplification and compression techniques can be employed to address this problem, albeit at the expense of a drop in accuracy. Although the knowledge distillation technique can mitigate the accuracy loss issue [222], efficiently applying this method is not an easy task [255]. Therefore, the first challenge is to make an object detection model hardware-friendly while maintaining acceptable model accuracy and speed to meet real-time constraints.

Developing FPGA-based object detection systems is a complex and multifaceted procedure. It begins with developing a suitable model or modifying an existing one based on the project requirements, leading up to the final hardware implementation. Consequently, this process requires knowledge from various areas (in both software and hardware) and the ability to work with multiple tools to perform each part of the development flow. For example, a designer may need to work with an AI framework such as PyTorch (working with Python), a high-level synthesis tool (working with C/C++), an FPGA design tool like Vivado (working with HDL), and some other software tools and programming languages depending on the project. The important point is that in each development phase, having comprehensive knowledge about the subsequent steps is needed to design an efficient system.

Many efforts have been made to provide appropriate tools to integrate this development process and reduce the need for in-depth knowledge in all parts, especially concerning hardware implementation [195], [196]. However, these tools are typically designed to work with some predefined object detection models for specific FPGA devices.

Consequently, the lengthy and challenging design process remains one of the main challenges in this field, directly impacting the time-to-market and, subsequently, the final price of the system.

B. POSSIBLE FUTURE RESEARCH TRENDS

Naturally, addressing the existing challenges mentioned earlier remains a primary research focus. As these challenges evolve, the development of hardware-friendly models tailored to FPGA architectures, accounting for their computational and memory constraints, will likely remain a key direction. Such models should aim to maximize the inherent parallelism of FPGAs while minimizing resource utilization, leading to better trade-offs between speed, accuracy, and power efficiency. To this end, deeper exploration of co-design techniques, such as Hardware-aware Neural Architecture Search (HW-NAS) [227], can play a pivotal role. These approaches could pave the way for automatic model optimization, potentially enabling the deployment of more sophisticated detection models with less manual tuning.

A promising area is the creation of software tools that can seamlessly integrate the design, optimization, and deployment processes for FPGA-based object detection systems. Although tool development tends to be more commercially driven, incorporating open-source initiatives and academic collaborations could stimulate further innovation. Additionally, the emergence of flexible hardware libraries and parameterizable soft cores capable of executing diverse object detection algorithms on a wide range of FPGA devices could standardize and accelerate the development cycle. Such efforts could emphasize modularity, enabling rapid adaptation to new FPGA platforms and model architectures without extensive re-engineering.

The trend toward using distributed FPGA clusters for real-time object detection is another area ripe for investigation. Distributed configurations could unlock higher performance and enable the design of fault-tolerant and scalable systems suitable for hard real-time requirements [258]. Future studies could explore novel frameworks for inter-node communication, computational task partitioning, and efficient scheduling strategies. As object detection models continue to grow in size and complexity, research on scalable data partitioning schemes, latency minimization, and load balancing across multiple FPGA nodes becomes increasingly vital. While some initial studies [293]–[295] have explored these areas, the field could benefit from more refined techniques to address the demands of emerging high-performance detection models.

The adoption of transformers in computer vision, particularly for object detection, has gained significant traction, with transformer-based models showing performance that rivals or surpasses that of conventional convolutional neural networks (CNNs) [93], [110]. However, implementing these models on FPGA platforms presents unique challenges due to their complex architecture, extensive parameter sets, and substantial computational and storage requirements [94]. Thus, more in-depth research is required to bridge the gap between transformer-based models' capabilities and the hardware constraints of FPGA devices. Future studies could explore the following directions to make FPGA-based transformer implementations feasible and efficient:

- **Hardware-Friendly Transformer Architectures:** Development of new transformer-based object detection models optimized specifically for FPGA deployment, focusing on reducing model complexity while maintaining accuracy.
- **Innovative Hardware Accelerator Designs:** Creation of specialized FPGA accelerators that match the unique structural characteristics of transformers, such as self-attention operations. These designs should emphasize parallelism and custom data paths to minimize latency and optimize resource usage. Furthermore, leveraging reconfigurable hardware properties to allocate resources for different transformer layers based on workload adaptively could improve overall system efficiency.
- **Enhanced Dataflow Techniques:** Efficient dataflow management can significantly reduce memory access and power consumption. Exploring new memory hierarchy designs, data compression techniques, and efficient

caching strategies can enhance parameter sharing and data reuse across the transformer's layers. Furthermore, research could focus on dynamic memory allocation schemes tailored to transformer models to optimize on-chip memory utilization.

Ultimately, the future of FPGA-based real-time object detection will likely involve a combination of approaches, from the development of optimized hardware accelerators and software tools to advancements in model co-design. As object detection models become increasingly sophisticated, the field must evolve to address the growing demands for higher accuracy, lower latency, and better energy efficiency. Researchers will need to push the boundaries of both algorithmic and hardware architecture innovations to meet these challenges, setting the stage for breakthroughs in real-time object detection on FPGAs.

VII. CONCLUSION

Real-time object detection is a critical task in applications such as autonomous vehicles and robotics. Key requirements like low latency, high throughput, and acceptable accuracy, tailored to the specific use case, underline the importance of efficient system designs. FPGAs have emerged as a suitable platform for these systems, offering unique advantages such as true parallelism, low and deterministic latency, high throughput, and the ability to process images directly from external imaging sources.

This paper provides a detailed review of the implementation and optimization techniques commonly employed in FPGA-based real-time object detection systems. We have discussed the concepts of soft and hard real-time systems, the advancements in object detection algorithms, particularly one-stage CNN-based architectures due to their ability to balance speed and accuracy in real-time scenarios, and commonly used evaluation metrics and datasets. Furthermore, we have analyzed existing literature on FPGA-based implementations and compared their performance using pixel throughput as a fair metric.

Despite significant progress, several challenges persist, such as integrating newer, computationally demanding object detection models, efficiently utilizing FPGA resources, and scaling designs to support higher-resolution video streams. Addressing these challenges requires innovative hardware-software co-design approaches, enhanced toolchains, and exploration of hybrid systems combining FPGAs with other accelerators.

Despite significant progress, implementing real-time object detection systems on FPGAs faces challenges such as accommodating increasingly complex models within limited hardware resources, maintaining real-time performance, and managing the multifaceted development process requiring expertise across software and hardware domains. Techniques like model simplification and knowledge distillation offer partial solutions but involve trade-offs in accuracy. Additionally, the lack of flexible, integrated design tools prolongs development cycles and impacts time-to-market. Addressing these issues demands advancements in hardware-aware model optimization, streamlined toolchains, and modular hardware architectures to enable efficient, scalable, and adaptable FPGA-based systems.

Future research may focus on optimizing hardware-friendly models, advancing co-design techniques like HW-NAS, and developing integrated software tools for efficient design. Exploring distributed FPGA clusters, modular hardware libraries, and specialized accelerators for transformer-based models could improve scalability, accuracy, and performance, enabling more efficient real-time systems across diverse applications.

In summary, this paper aims to provide a consolidated overview of the field, offering insights into state-of-the-art techniques, comparative analyses, and guidance for researchers working to leverage FPGA platforms for real-time object detection. By addressing the challenges and opportunities highlighted, FPGA-based implementations can continue to advance as a robust solution for meeting real-time application requirements.

APPENDIX A
SOME GENERAL DEFINITIONS IN THE CONTEXT OF OBJECT DETECTION METRICS

Measuring the Intersection over Union *(IoU)* is a popular way to evaluate an object detector's localization accuracy. Given a ground truth Bounding box *(Bbox)*, it calculates the ratio of the common, or overlapped, area between the ground truth Bbox and the predicted Bbox to the area of their union, as illustrated in Figure 21. The greater the IoU ratio means the more detection accuracy. By setting a threshold, e.g., IoU > 0.5, as done in [40], it can be determined how precisely the object is localized.

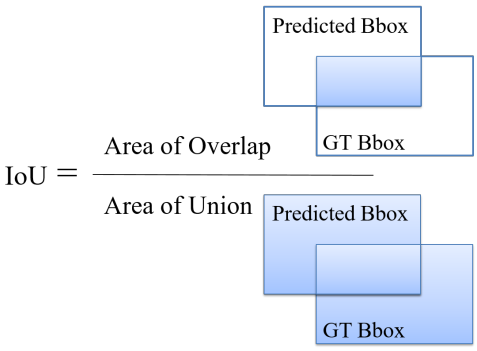


FIGURE 21. Definition of Intersection over Union (IoU), calculated by dividing the intersection of the predicted bounding box (Bbox) and ground truth (GT) Bbox by the union of them.

Based on the calculated IoU and a pre-determined threshold, some other metrics, such as precision and recall, can be obtained. The IoU greater than or equal to the threshold value indicates that the prediction is correct. It should be noted that IoU is basically related to the localization aspect of object detection tasks and is not directly related to classification.

Given the real label and the predicted one for each object, the prediction result can be classified as shown in Table 11. Accordingly, accuracy, precision, and recall values are defined as follows:

TABLE 11. Prediction result category in object detection models based on the label provided by the dataset (real label).

Real Label	Predicted Label	Prediction Category
Positive	Positive	True Positive (TP)
Positive	Negative	False Negative (FN)
Negative	Positive	False Positive (FP)
Negative	Negative	True Negative (TN)

$$Accuracy := \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

$$Precision := \frac{TP}{(TP + FP)}$$

$$Recall := \frac{TP}{(TP + FN)}$$

Accuracy determines how accurately the model makes predictions, showing the overall performance of the model. However, this metric has a significant drawback: it performs poorly with imbalanced data, where one class significantly outnumbers the others [296].

Two alternatives for accuracy are "precision" and "recall," which focus on "true positive" predictions. The former represents how precisely the model can predict the positive class, measuring the quality of the detection task, while the latter, also known as "sensitivity", refers to the proportion of "true positive" predictions to all positive instances in the dataset. From another point of view, if we need to minimize the false positive predictions, we should focus on improving precision, while recall is more important when the ability to predict all positives outweighs the detection accuracy.

The "F1 score", also known as the "balanced F-score" or "F-measure", provides a means to evaluate the trade-off between recall and precision. Representing the harmonic mean of precision and recall, it ranges from 0 (indicating the worst value) to 1 (indicating the best value). The F1-score disregards variations in confidence values, limiting its utility to comparing object detectors solely at a predetermined confidence threshold level [297].

$$F1 := \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{(2 * Precision * Recall)}{(Precision + Recall)} \tag{6}$$

Average precision *(AP)* has recently emerged as the most commonly utilized evaluation metric for detection tasks [30]. AP can be derived by calculating the area under the precision and recall curve (PR Curve). In other words, it indicates the average precision values across all recall values ranging from 0 to 1.

Widely utilized in computer vision, AP serves as a popular evaluation measure for assessing the prediction accuracy of object detection models [45]. AP can be assessed across various IoU threshold ranges. For instance, it can be computed for 10 IoU values ranging from 50% to 95%, with increments of 5%, typically denoted as "AP@50:5:95". Additionally, it can be evaluated at specific IoU thresholds, commonly 50% and 75% denoted as "AP50" and "AP75" respectively [34].

APPENDIX B
ROOFLINE MODEL

In this visual model, the peak computational performance provided by the hardware platform and the maximum off-chip memory bandwidth are the two critical factors for estimating the attainable performance [273].

The equation 7, represents a "roofline"-type curve on the Cartesian plane where the X-axis is the Operational Intensity *(OI)* or Compute-To-Communication *(CTC)* measured in Floating Point Operations per Byte or just Operations per Byte [203] and the Y-axis represents the computational performance measured in Floating Point Operations per second or just Operations per second (Figure 22).

$$AchievablePerformance = min \begin{cases} PeakComputationalPerformance \\ PeakMemoryBandwidth \times CTC \end{cases} \tag{7}$$

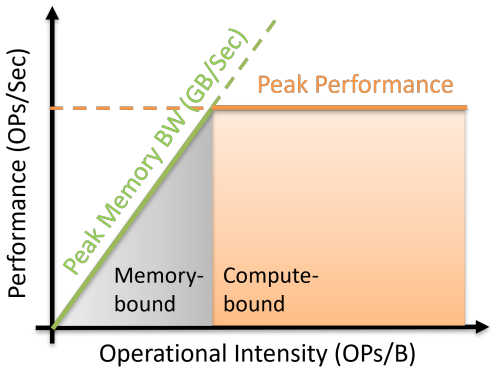


FIGURE 22. The Roofline model.

In the Roofline model, an application's performance can be represented as a point on a Cartesian plane, which visualizes the distance between the actual performance and the obtainable one.

In an application, the CTC ratio, also known as operational intensity, represents the ratio of the total number of executed operations to the amount of data transferred to and from external memory and can be measured as operation per byte *(op/Byte)* or floating point operation per Byte *(Flops/Byte)*.

The performance of the application is calculated at run time while executing it.

The roofline constrains the achievable performance, delineating the figure into two distinct regions: *memory-bound* and *compute-bound* areas. The ridge point, also known as the machine balance point, highlighted in Figure 22, is at the intersection between the diagonal and horizontal lines. Applications with CTC ratios on the left side of the ridge point are considered ''memory-bound'', indicating that the system cannot efficiently utilize all computational resources due to limited off-chip communication. On the other hand, applications with a CTC ratio on the right side of the ridge point are referred to as ''compute-bound''.

If the application is memory-bound, optimizing memory inefficiencies is often a fruitful strategy, focusing on factors such as memory access pattern, data locality, and cache reuse [273], [275], [298]. If the application is compute-bound, the bottleneck lies in the computational power or efficiency of the accelerator. In this case, applying some optimization techniques such as loop unrolling and loop reordering may help [273].

ACKNOWLEDGMENT

The authors thank Xilinx/AMD and NVIDIA for their continuous support and donation of hardware devices to our research laboratory. These contributions have been invaluable for us to conduct cutting-edge research in large-scale computing. The European Commission partially supported this work under the EDGE-ME, AXIOM H2020 (id. 645496), TERAFLUX (id. 249013), and HiPEAC (id. 101069836) projects. This work is partly funded by the European Union - NextGenerationEU - via the PNRR M4C2-Inv1.4 Italian Research Center on High-Performance Computing, Big-Data and Quantum Computing, cascade funding project EDGE-ME, MUR-ID: CN0000013. The University of Siena, Campera-ES, and the Tuscany Region also partially supported this work under the HIPERAIHL project.

REFERENCES

[1] Tolga Turay and Tanya Vladimirova. Toward performing image classification and object detection with convolutional neural networks in autonomous driving systems: A survey. *IEEE Access*, 10:14076–14119, 2022.

[2] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 129–137, 2017.

[3] Hakan Karaoguz and Patric Jensfelt. Object detection approach for robot grasp detection. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4953–4959. IEEE, 2019.

[4] Sayantan Chatterjee, Faheem H Zunjani, and Gora C Nandi. Real-time object detection and recognition on low-compute humanoid robots using deep learning. In *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, pages 202–208. IEEE, 2020.

[5] Amrita Kaur, Yadwinder Singh, Nirvair Neeru, Lakhwinder Kaur, and Ashima Singh. A survey on deep learning approaches to medical images and a systematic look up into real-time object detection. *Archives of Computational Methods in Engineering*, pages 1–41, 2021.

[6] Ruixin Yang and Yingyan Yu. Artificial convolutional neural network in object detection and semantic segmentation for medical imaging analysis. *Frontiers in oncology*, 11:638182, 2021.

[7] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[8] Luca Pezzarossa, Martin Schoeberl, and Jens Sparsø. Reconfiguration in FPGA-based multi-core platforms for hard real-time applications. In *2016 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2016.

[9] Sangeet Saha, Shoaib Ehsan, Adrian Stoica, Rustam Stolkin, and Klaus McDonald-Maier. Real-time application processing for FPGA-based resilient embedded systems in harsh environments. In *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 299–304. IEEE, 2018.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[11] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[12] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.

[13] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. Detrs beat yolos on real-time object detection. *arXiv preprint arXiv:2304.08069*, 2023.

[14] Kai Zeng, Qian Ma, Jia Wen Wu, Zhe Chen, Tao Shen, and Chenggang Yan. FPGA-based accelerator for object detection: a comprehensive survey. *Journal of Supercomputing*, 78:14096–14136, 8 2022.

[15] Duy Thanh Nguyen, Tuan Nghia Nguyen, Hyun Kim, and Hyuk-Jae Lee. A high-throughput and power-efficient FPGA implementation of yolo cnn for object detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(8):1861–1873, 2019.

[16] Temitope Ibrahim Amosa, Patrick Sebastian, Lila Iznita Izhar, Oladimeji Ibrahim, Lukman Shehu Ayinla, Abdulrahman Abdullah Bahashwan, Abubakar Bala, and Yau Alhaji Samaila. Multi-camera multi-object tracking: a review of current trends and future advances. *Neurocomputing*, 552:126558, 2023.

[17] Lunlin Fei and Bing Han. Multi-object multi-camera tracking based on deep learning for intelligent transportation: A review. *Sensors*, 23(8):3852, 2023.

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[20] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2015.

[21] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[23] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[24] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023.

[25] Payal Mittal. A comprehensive survey of deep learning-based lightweight object detection models for edge devices. *Artificial Intelligence Review*, 57(9):242, 2024.

[26] Ayoub Benali Amjoud and Mustapha Amrouch. Object detection using deep learning, cnns and vision transformers: a review. *IEEE Access*, 2023.

[27] Vidya Kamath and A Renuka. Deep learning based object detection for resource constrained devices: Systematic review, future trends and challenges ahead. *Neurocomputing*, 531:34–60, 2023.

[28] Ravpreet Kaur and Sarbjeet Singh. A comprehensive review of object detection with deep learning. *Digital Signal Processing*, 132:103812, 2023.

[29] Arief Setyanto, Theopilus Bayu Sasongko, Muhammad Ainul Fikri, and In Kee Kim. Near-edge computing aware object detection: A review. *IEEE Access*, 12:2989–3011, 2024.

[30] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 111(3):257–276, 2023.

[31] Zhenhua Huang, Shunzhi Yang, MengChu Zhou, Zheng Gong, Abdullah Abusorrah, Chen Lin, and Zheng Huang. Making accurate object detection at the edge: Review and new approach. *Artificial Intelligence Review*, 55(3):2245–2274, 2022.

[32] Jaskirat Kaur and Williamjeet Singh. Tools, techniques, datasets and application areas for object detection in an image: a review. *Multimedia Tools and Applications*, 81(27):38297–38351, 2022.

[33] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, and Brian Lee. A survey of modern deep learning-based object detection models. *Digital Signal Processing*, 126:103514, 2022.

[34] Rafael Padilla, Sergio L Netto, and Eduardo AB Da Silva. A survey on performance metrics for object-detection algorithms. In *2020 international conference on systems, signals and image processing (IWSSIP)*, pages 237–242. IEEE, 2020.

[35] Muhammad Ahmed, Khurram Azeem Hashmi, Alain Pagani, Marcus Liwicki, Didier Stricker, and Muhammad Zeshan Afzal. Survey and performance analysis of deep learning based object detection in challenging environments. *Sensors*, 21(15):5116, 2021.

[36] Youzi Xiao, Zhiqiang Tian, Jiachen Yu, Yinshu Zhang, Shuai Liu, Shaoyi Du, and Xuguang Lan. A review of object detection based on deep learning. *Multimedia Tools and Applications*, 79:23729–23791, 2020.

[37] Lixuan Du, Rongyu Zhang, and Xiaotian Wang. Overview of two-stage object detection algorithms. In *Journal of Physics: Conference Series*, volume 1544, pages 1–7. IOP Publishing, 2020.

[38] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. Recent advances in deep learning for object detection. *Neurocomputing*, 396:39–64, 2020.

[39] Anamika Dhillon and Gyanendra K Verma. Convolutional neural network: a review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, 9(2):85–112, 2020.

[40] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[42] Yanjiao Chen, Baolin Zheng, Zihan Zhang, Qian Wang, Chao Shen, and Qian Zhang. Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions. *ACM Computing Surveys (CSUR)*, 53(4):1–37, 2020.

[43] Taiwo Samuel Ajani, Agbotiname Lucky Imoize, and Aderemi A Atayero. An overview of machine learning within embedded and mobile devices–optimizations and applications. *Sensors*, 21(13):4412, 2021.

[44] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. A survey on deep neural network compression: Challenges, overview, and solutions. *arXiv preprint arXiv:2010.03954*, 2020.

[45] Mark Everingham and John Winn. The pascal visual object classes challenge 2012 (voc2012) development kit. *Pattern Anal. Stat. Model. Comput. Learn., Tech. Rep*, 2007(1-45):5, 2012.

[46] Suchang Kim, Seungho Na, Byeong Yong Kong, Jaewoong Choi, and In-Cheol Park. Real-time ssdlite object detection on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(6):1192–1205, 2021.

[47] Hongxiang Fan, Shuanglong Liu, Martin Ferianc, Ho-Cheung Ng, Zhiqiang Que, Shen Liu, Xinyu Niu, and Wayne Luk. A real-time object detection accelerator with compressed ssdlite on FPGA. In *2018 International conference on field-programmable technology (FPT)*, pages 14–21. IEEE, 2018.

[48] Jinming Zhang, Lifu Cheng, Cen Li, Yongfu Li, Guanghui He, Ningyi Xu, and Yong Lian. A low-latency FPGA implementation for real-time object detection. In *2021 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2021.

[49] Jacinto C Nascimento and Jorge S Marques. Performance evaluation of object detection algorithms for video surveillance. *IEEE Transactions on Multimedia*, 8(4):761–774, 2006.

[50] Sudan Jha, Changho Seo, Eunmok Yang, and Gyanendra Prasad Joshi. Real-time object detection and tracking system for video surveillance system. *Multimedia Tools and Applications*, 80(3):3981–3996, 2021.

[51] David Castells-Rufas, Vinh Ngo, Juan Borrego-Carazo, Marc Codina, Carles Sanchez, Debora Gil, and Jordi Carrabina. A survey of FPGA-based vision systems for autonomous cars. *IEEE Access*, 10:132525–132563, 2022.

[52] Ali KZ Tehrani and Hassan Rivaz. Displacement estimation in ultrasound elastography using pyramidal convolutional neural network. *IEEE Transactions on ultrasonics, ferroelectrics, and frequency control*, 67(12):2629–2639, 2020.

[53] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, et al. T-cnn: Tubelets with convolutional neural networks for object detection from videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2896–2907, 2017.

[54] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII 13*, pages 297–312. Springer, 2014.

[55] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3150–3158, 2016.

[56] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[57] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[58] Qi Wu, Chunhua Shen, Peng Wang, Anthony Dick, and Anton Van Den Hengel. Image captioning and visual question answering based on attributes and external knowledge. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1367–1381, 2017.

[59] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages 1–9, 2001.

[60] Peter Irgens, Curtis Bader, Theresa Lé, Devansh Saxena, and Cristinel Ababei. An efficient and cost-effective FPGA based implementation of the viola-jones face detection algorithm. *HardwareX*, 1:68–75, 2017.

[61] Felix Noel Sitorus, Yustina Manihuruk, and Good Fried Panggabean. Implementation of viola-jones algorithm for object detection using FPGA. In *2019 International Conference of Computer Science and Information Technology (ICoSNIKOM)*, pages 1–6. IEEE, 2019.

[62] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. Ieee, 2008.

[63] Daniele Tasson, Alessio Montagnini, Roberto Marzotto, Michela Farenzena, and Marco Cristani. FPGA-based pedestrian detection under strong distortions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 66–71, 2015.

[64] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[65] Jianjing An, Dezheng Zhang, Ke Xu, and Dong Wang. An opencl-based FPGA accelerator for faster r-cnn. *Entropy*, 24(10):1346, 2022.

[66] Zhenguang Li and Jintao Wang. An improved algorithm for deep learning yolo network based on xilinx zynq FPGA. In *2020 International Conference on Culture-oriented Science & Technology (ICCST)*, pages 447–451. IEEE, 2020.

[67] Anupreetham Anupreetham, Mohamed Ibrahim, Mathew Hall, Andrew Boutros, Ajay Kuzhively, Abinash Mohanty, Eriko Nurvitadhi, Vaughn Betz, Yu Cao, and Jae-Sun Seo. High throughput FPGA-based object detection via algorithm-hardware co-design. *ACM Transactions on Reconfigurable Technology and Systems*, 17(1):1–20, 2024.

[68] Liangwei Cai, Ceng Wang, and Yuan Xu. A real-time fpga accelerator based on winograd algorithm for underwater object detection. *Electronics*, 10(23):2889, 2021.

[69] Han-Sok Suh, Jian Meng, Ty Nguyen, Vijay Kumar, Yu Cao, and Jae-Sun Seo. Algorithm-hardware co-optimization for energy-efficient drone detection on resource-constrained fpga. *ACM Transactions on Reconfigurable Technology and Systems*, 16(2):1–25, 2023.

[70] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[71] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[72] Hiroki Nakahara, Masayuki Shimoda, and Shimpei Sato. A demonstration of fpga-based you only look once version2 (yolov2). In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 457–4571. IEEE, 2018.

[73] Zixiao Wang, Ke Xu, Shuaixiao Wu, Li Liu, Lingzhi Liu, and Dong Wang. Sparse-yolo: Hardware/software co-design of an FPGA accelerator for yolov2. *IEEE Access*, 8:116569–116585, 2020.

[74] Shuai Li, Yukui Luo, Kuangyuan Sun, Nandakishor Yadav, and Kyuwon Ken Choi. A novel FPGA accelerator design for real-time and ultra-low power deep convolutional neural networks compared with titan x gpu. *IEEE Access*, 8:105455–105471, 2020.

[75] June Wai Yap, Zulkalnain bin Mohd Yussof, Sani Irwan bin Salim, and Kim Chuan Lim. Fixed point implementation of tiny-yolo-v2 using OpenCL on FPGA. *International Journal of Advanced Computer Science and Applications*, 9(10), 2018.

[76] Ke Xu, Xiaoyun Wang, Xinyang Liu, Changfeng Cao, Huolin Li, Haiyong Peng, and Dong Wang. A dedicated hardware accelerator for real-time acceleration of yolov2. *Journal of Real-Time Image Processing*, 18:481–492, 2021.

[77] Vikram Jain, Ninad Jadhav, and Marian Verhelst. Enabling real-time object detection on low cost fpgas. *Journal of Real-Time Image Processing*, 19(1):217–229, 2022.

[78] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[79] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European conference on computer vision (ECCV)*, pages 734–750, 2018.

[80] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[81] Jin Wang and Shenshen Gu. FPGA implementation of object detection accelerator based on vitis-ai. In *2021 11th International Conference on Information Science and Technology (ICIST)*, pages 571–577. IEEE, 2021.

[82] Libo Chang, Shengbing Zhang, Huimin Du, Yue Chen, and Shiyu Wang. A reconfigurable neural network processor with tile-grained multicore pipeline for object detection on fpga. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(11):1967–1980, 2021.

[83] Zhewen Yu and Christos-Savvas Bouganis. A parameterisable FPGA-tailored architecture for yolov3-tiny. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th International Symposium, ARC 2020, Toledo, Spain, April 1–3, 2020, Proceedings 16*, pages 330–344. Springer, 2020.

[84] Minsik Kim, Kyoungseok Oh, Youngmock Cho, Hojin Seo, Xuan Truong Nguyen, and Hyuk-Jae Lee. A low-latency fpga accelerator for yolov3-tiny with flexible layerwise mapping and dataflow. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[85] Daniel Pestana, Pedro R Miranda, João D Lopes, Rui P Duarte, Mário P Véstias, Horácio C Neto, and José T De Sousa. A full featured configurable accelerator for object detection with yolo. *IEEE Access*, 9:75864–75877, 2021.

[86] Dezheng Zhang, Aibin Wang, Ruchan Mo, and Dong Wang. End-to-end acceleration of the yolo object detection framework on FPGA-only devices. *Neural Computing and Applications*, 36(3):1067–1089, 2024.

[87] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6569–6578, 2019.

[88] Roman A Solovyev, Dmitry V Telpukhov, Irina I Romanova, Alexander G Kustov, and Ilya A Mkrtchan. Real-time object detection with FPGA using centernet. In *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, pages 2029–2034. IEEE, 2021.

[89] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[90] Praveenkumar Babu and Eswaran Parthasarathy. Hardware acceleration for object detection using yolov4 algorithm on xilinx zynq platform. *Journal of Real-Time Image Processing*, 19(5):931–940, 2022.

[91] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 13029–13038, 2021.

[92] Adib Hosseiny and Hadi Jahanirad. Hardware acceleration of yolov7-tiny using high-level synthesis tools. *Journal of Real-Time Image Processing*, 20(4):75, 2023.

[93] Yong Li, Naipeng Miao, Liangdi Ma, Feng Shuang, and Xingwen Huang. Transformer for object detection: Review and benchmark. *Eng. Appl. Artif. Intell.*, 126(PC), feb 2024.

[94] Takeshi Senoo, Ryota Kayanoma, Akira Jinguji, and Hiroki Nakahara. A light-weight vision transformer toward near memory computation on an FPGA. In *International Symposium on Applied Reconfigurable Computing*, pages 338–353. Springer, 2023.

[95] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[96] Amr Suleiman and Vivienne Sze. An energy-efficient hardware implementation of hog-based object detection at 1080hd 60 fps with multi-scale support. *Journal of Signal Processing Systems*, 84:325–337, 2016.

[97] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[98] Serge Belongie, Jitendra Malik, and Jan Puzicha. Matching shapes. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 454–461. IEEE, 2001.

[99] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.

[100] Ross Girshick, Pedro Felzenszwalb, and David McAllester. Object detection with grammar models. *Advances in neural information processing systems*, 24, 2011.

[101] Ross Brook Girshick. *From rigid templates to grammars: object detection with structured models*. PhD thesis, University of Chicago, USA, 2012. AAI3513455.

[102] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[103] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

[104] Ershat Arkin, Nurbiya Yadikar, Xuebin Xu, Alimjan Aysa, and Kurban Ubul. A survey: object detection methods from cnn to transformer. *Multimedia Tools and Applications*, 82(14):21353–21383, 2023.

[105] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[106] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[107] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[108] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[109] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[110] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16

words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[111] Manuel Carranza-García, Jesús Torres-Mateo, Pedro Lara-Benítez, and Jorge García-Gutiérrez. On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data. *Remote Sensing*, 13(1):89, 2020.

[112] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[113] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*, 29, 2016.

[114] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. Yolov9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616*, 2024.

[115] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*, 2024.

[116] Darknet: Open source neural networks in c. https://pjreddie.com/darknet/. (Accessed on 04/05/2024).

[117] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[118] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

[119] Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. In *18th international conference on pattern recognition (ICPR'06)*, volume 3, pages 850–855. IEEE, 2006.

[120] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[121] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[122] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[123] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[124] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[125] Josh Beal, Eric Kim, Eric Tzeng, Dong Huk Park, Andrew Zhai, and Dmitry Kislyuk. Toward transformer-based object detection. *arXiv preprint arXiv:2012.09958*, 2020.

[126] Willy Fitra Hendria, Quang Thinh Phan, Fikriansyah Adzaka, and Cheol Jeong. Combining transformer and cnn for object detection in uav imagery. *ICT Express*, 9(2):258–263, 2023.

[127] Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *European conference on computer vision*, pages 108–126. Springer, 2020.

[128] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. *Advances in neural information processing systems*, 32, 2019.

[129] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[130] Tahira Shehzadi, Khurram Azeem Hashmi, Didier Stricker, and Muhammad Zeshan Afzal. 2d object detection with transformers: a review. *arXiv preprint arXiv:2306.04670*, 2023.

[131] Zhigang Dai, Bolun Cai, Yugeng Lin, and Junying Chen. Up-detr: Unsupervised pre-training for object detection with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1601–1610, 2021.

[132] Zhuyu Yao, Jiangbo Ai, Boxun Li, and Chi Zhang. Efficient detr: improving end-to-end object detector with dense prior. *arXiv preprint arXiv:2104.01318*, 2021.

[133] Hwanjun Song, Deqing Sun, Sanghyuk Chun, Varun Jampani, Dongyoon Han, Byeongho Heo, Wonjae Kim, and Ming-Hsuan Yang. Vidt: An efficient and effective fully transformer-based object detector. *arXiv preprint arXiv:2110.03921*, 2021.

[134] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.

[135] Kang G Shin and Parameswaran Ramanathan. Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, 1994.

[136] John A Stankovic. Real-time and embedded systems. *ACM Computing Surveys (CSUR)*, 28(1):205–208, 1996.

[137] S Baskiyar N Meghanathan. A survey of contemporary real-time operating systems. *Informatica*, 29(2), 2005.

[138] Bruce Powel Douglass. *Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns*, volume 1. Addison-Wesley Professional, 1999.

[139] Bruce Powel Douglass. *Real-time design patterns: robust scalable architecture for real-time systems*. Addison-Wesley Professional, 2003.

[140] Qing Li and Caroline Yao. *Real-time concepts for embedded systems*. CRC press, 2003.

[141] Phillip A Laplante et al. *Real-time systems design and analysis*. Wiley New York, 2004.

[142] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[143] Huizi Mao, Song Yao, Tianqi Tang, Boxun Li, Jun Yao, and Yu Wang. Towards real-time object detection on embedded systems. *IEEE Transactions on Emerging Topics in Computing*, 6(3):417–431, 2016.

[144] Lukas Baischer, Matthias Wess, and Nima TaheriNejad. Learning on hardware: A tutorial on neural network accelerators and co-processors. *arXiv preprint arXiv:2104.09252*, 2021.

[145] Arish Sateesan, Sharad Sinha, Smitha KG, and AP Vinod. A survey of algorithmic and hardware optimization techniques for vision convolutional neural networks on FPGAs. *Neural Processing Letters*, 53(3):2331–2377, 2021.

[146] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[147] The latest in machine learning | papers with code. https://paperswithcode.com/. (Accessed on 04/07/2024).

[148] Ivan Rodriguez-Conde, Celso Campos, and Florentino Fdez-Riverola. On-device object detection for more efficient and privacy-compliant visual perception in context-aware systems. *Applied Sciences*, 11(19):9173, 2021.

[149] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. *Efficient processing of deep neural networks*. Springer, 2020.

[150] JunKyu Lee, Lev Mukhanov, Amir Sabbagh Molahosseini, Umar Minhas, Yang Hua, Jesus Martinez del Rincon, Kiril Dichev, Cheol-Ho Hong, and Hans Vandierendonck. Resource-efficient convolutional networks: A survey on model-, arithmetic-, and implementation-level techniques. *ACM Computing Surveys*, 55(13s):1–36, 2023.

[151] Ian Kuon, Russell Tessier, Jonathan Rose, et al. FPGA architecture: Survey and challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2):135–253, 2008.

[152] Umer Farooq, Zied Marrakchi, Habib Mehrez, Umer Farooq, Zied Marrakchi, and Habib Mehrez. FPGA architectures: An overview. *Tree-Based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*, pages 7–48, 2012.

[153] Bajaj Ronak and Suhaib A Fahmy. Mapping for maximum performance on FPGA dsp blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(4):573–585, 2015.

[154] Uwe Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, 2014.

[155] Srdjan Coric, Miriam Leeser, Eric Miller, and Marc Trepanier. Parallel-beam backprojection: an FPGA implementation optimized for medical imaging. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, pages 217–226, 2002.

[156] Rajesh Mehra, Garima Saini, and Sukhbir Singh. FPGA based high speed bch encoder for wireless communication applications. In *2011 International Conference on Communication Systems and Network Technologies*, pages 576–579. IEEE, 2011.

[157] Zeyad Assi Obaid Nasri Sulaiman, MH Marhaban, and MN Hamidon. Design and implementation of FPGA-based systems-a review. *Australian Journal of Basic and Applied Sciences*, 3(4):3575–3596, 2009.

[158] Tarek El-Ghazawi, Dave Bennett, Dan Poznanovic, Allan Cantle, Keith Underwood, Rob Pennington, Duncan Buell, Alan George, and Volodymyr Kindratenko. Is high-performance reconfigurable computing the next supercomputing paradigm? In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, pages 71–es, 2006.

[159] Griffin Lacey, Graham W Taylor, and Shawki Areibi. Deep learning on FPGAs: Past, present, and future. *arXiv preprint arXiv:1602.04283*, 2016.

[160] Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang. Understanding performance differences of FPGAs and gpus. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 93–96. IEEE, 2018.

[161] Wolf Wayne et al. *FPGA-based system design*. Pearson Education India, 2004.

[162] Aswin C Sankaranarayanan, Ashok Veeraraghavan, and Rama Chellappa. Object detection, tracking and recognition for multiple smart cameras. *Proceedings of the IEEE*, 96(10):1606–1624, 2008.

[163] Andrew Boutros and Vaughn Betz. FPGA architecture: Principles and progression. *IEEE Circuits and Systems Magazine*, 21(2):4–29, 2021.

[164] Declan O'Loughlin, Aedan Coffey, Frank Callaly, Darren Lyons, and Fearghal Morgan. Xilinx vivado high level synthesis: Case studies, 2014.

[165] Jason Cong, Jason Lau, Gai Liu, Stephen Neuendorffer, Peichen Pan, Kees Vissers, and Zhiru Zhang. FPGA hls today: successes, challenges, and opportunities. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(4):1–42, 2022.

[166] Jerry Chan Ting Hai, Ooi Chee Pun, and Tan Wooi Haw. Accelerating video and image processing design for FPGA using hdl coder and simulink. In *2015 IEEE Conference on Sustainable Utilization And Development In Engineering and Technology (CSUDET)*, pages 1–5. IEEE, 2015.

[167] Nasser Kehtarnavaz and Sidharth Mahotra. *Digital Signal Processing Laboratory: LabVIEW-Based FPGA Implementation*. Universal-Publishers, 2010.

[168] Xilinx. *UG1262*. Xilinx, 2nd edition, October 2019. Available at https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1262-model-composer-user-guide.pdf.

[169] Gabriela Nicolescu and Pieter J Mosterman. *Model-based design for embedded systems*. Crc Press, 2018.

[170] AMD Xilinx. Dpuczdx8g for zynq ultrascale+ mpsocs. product guide, 2022.

[171] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[172] Victor Wiley and Thomas Lucas. Computer vision and image processing: a paper review. *International Journal of Artificial Intelligence Research*, 2(1):29–36, 2018.

[173] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(1):154–180, 2020.

[174] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

[175] Xu Zheng, Yexin Liu, Yunfan Lu, Tongyan Hua, Tianbo Pan, Weiming Zhang, Dacheng Tao, and Lin Wang. Deep learning for event-based vision: A comprehensive survey and benchmarks. *arXiv preprint arXiv:2302.08890*, 2023.

[176] Jürgen Beyerer, Fernando Puente León, Christian Frese, Jürgen Beyerer, Fernando Puente León, and Christian Frese. Preprocessing and image enhancement. *Machine Vision: Automated Visual Inspection: Theory, Practice and Applications*, pages 465–519, 2016.

[177] Scott Krig and Scott Krig. Image pre-processing. *Computer Vision Metrics: Textbook Edition*, pages 35–74, 2016.

[178] Mainak Sen, Ivan Corretjer, Fiorella Haim, Sankalita Saha, Shuvra S Bhattacharyya, Jason Schlessman, and Wayne Wolf. Computer vision on FPGAs: Design methodology and its application to gesture recognition. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops*, pages 133–133. IEEE, 2005.

[179] Abhishek Gupta. Current research opportunities for image processing and computer vision. *Computer Science*, 20:387–410, 2019.

[180] Eyad Elyan, Pattaramon Vuttipittayamongkol, Pamela Johnston, Kyle Martin, Kyle McPherson, Chrisina Jayne, Mostafa Kamal Sarker, et al. Computer vision and machine learning for medical image analysis: recent advances, challenges, and way forward. *Artificial Intelligence Surgery*, 2, 2022.

[181] Xin Feng, Youni Jiang, Xuejiao Yang, Ming Du, and Xin Li. Computer vision algorithms and hardware implementations: A survey, 11 2019.

[182] Deepayan Bhowmik and Kofi Appiah. Embedded vision systems: A review of the literature. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 14th International Symposium, ARC 2018, Santorini, Greece, May 2-4, 2018, Proceedings 14*, pages 204–216. Springer, 2018.

[183] Seunghun Jin, Junguk Cho, Xuan Dai Pham, Kyoung Mu Lee, Sung-Kee Park, Munsang Kim, and Jae Wook Jeon. FPGA design and implementation of a real-time stereo vision system. *IEEE transactions on circuits and systems for video technology*, 20(1):15–26, 2009.

[184] Sparsh Mittal, Saket Gupta, and Sudeb Dasgupta. FPGA: An efficient and promising platform for real-time image processing applications. In *National Conference On Research and Development In Hardware Systems (CSI-RDHS)*, 2008.

[185] Kah Phooi Seng, Paik Jen Lee, and Li Minn Ang. Embedded intelligence on FPGA: Survey, applications and challenges. *Electronics*, 10(8):895, 2021.

[186] Arif Irwansyah, Omar W Ibraheem, Jens Hagemeyer, Mario Porrmann, and Ulrich Rueckert. FPGA-based multi-robot tracking. *Journal of Parallel and Distributed Computing*, 107:146–161, 2017.

[187] Gige vision. https://www.automate.org/vision/vision-standards/vision-standards-gige-vision. (Accessed on 05/27/2024).

[188] M Ali Altuncu, Taner Guven, Yasar Becerikli, and Suhap Sahin. Real-time system implementation for image processing with hardware/software co-design on the xilinx zynq platform. *International Journal of Information and Electronics Engineering*, 5(6):473, 2015.

[189] Interface specifications for mobile products | mipi allliance. https://www.mipi.org/. (Accessed on 06/25/2024).

[190] Jahanzeb Ahmad and Alexander Warren. FPGA based deterministic latency image acquisition and processing system for automated driving systems. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[191] Kyungho Kim, Sung-Joon Jang, Jonghee Park, Eunchong Lee, and Sang-Seol Lee. Lightweight and energy-efficient deep learning accelerator for real-time object detection on edge devices. *Sensors*, 23(3):1185, 2023.

[192] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. FPGA/dnn co-design: An efficient design methodology for iot intelligence on the edge. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[193] Qijing Huang, Dequan Wang, Zhen Dong, Yizhao Gao, Yaohui Cai, Tian Li, Bichen Wu, Kurt Keutzer, and John Wawrzynek. Codenet: Efficient deployment of input-adaptive object detection on embedded FPGAs. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 206–216, 2021.

[194] Zhichao Zhang, MA Parvez Mahmud, and Abbas Z Kouzani. Resource-constrained FPGA implementation of yolov2. *Neural Computing and Applications*, 34(19):16989–17006, 2022.

[195] Amd together we advance_ai. https://www.amd.com/en.html. (Accessed on 05/28/2024).

[196] Intel | data center solutions, iot, and pc innovation. https://www.intel.com/content/www/us/en/homepage.html. (Accessed on 05/28/2024).

[197] Stylianos I Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions. *ACM Computing Surveys (CSUR)*, 51(3):1–39, 2018.

[198] Jincheng Yu, Kaiyuan Guo, Yiming Hu, Xuefei Ning, Jiantao Qiu, Huizi Mao, Song Yao, Tianqi Tang, Boxun Li, Yu Wang, et al. Real-time object detection towards high power efficiency. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 704–708. IEEE, 2018.

[199] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. Angel-eye: A complete design flow for mapping cnn onto embedded FPGA. *IEEE transactions*

on computer-aided design of integrated circuits and systems, 37(1):35–47, 2017.

[200] Miguel Reis, Mário Véstias, and Horácio Neto. Designing deep learning models on FPGA with multiple heterogeneous engines. *ACM Transactions on Reconfigurable Technology and Systems*, 17(1):1–30, 2024.

[201] Alexander Montgomerie-Corcoran, Petros Toupas, Zhewen Yu, and Christos-Savvas Bouganis. Satay: a streaming architecture toolflow for accelerating yolo models on FPGA devices. In *2023 International Conference on Field Programmable Technology (ICFPT)*, pages 179–187. IEEE, 2023.

[202] Hsiang Tsung Kung and Charles E Leiserson. Systolic arrays (for vlsi). In *Sparse Matrix Proceedings 1978*, volume 1, pages 256–282. Society for industrial and applied mathematics Philadelphia, PA, USA, 1979.

[203] Erwei Wang, James J Davis, Ruizhe Zhao, Ho-Cheung Ng, Xinyu Niu, Wayne Luk, Peter YK Cheung, and George A Constantinides. Deep neural network approximation for custom hardware: Where we've been, where we're going. *ACM Computing Surveys (CSUR)*, 52(2):1–39, 2019.

[204] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 37–47, 2010.

[205] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[206] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 26–35, 2016.

[207] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. Optimizing the convolution operation to accelerate deep neural networks on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(7):1354–1367, 2018.

[208] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput cnn inference on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[209] Chao Wang, Wenqi Lou, Lei Gong, Lihui Jin, Luchao Tan, Yahui Hu, Xi Li, and Xuehai Zhou. Reconfigurable hardware accelerators: Opportunities, trends, and challenges. *arXiv preprint arXiv:1712.04771*, 2017.

[210] Stanley F Anderson, John G Earle, Robert Elliott Goldschmidt, and Don M Powers. The ibm system/360 model 91: Floating-point execution unit. *IBM Journal of research and development*, 11(1):34–53, 1967.

[211] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170, 2015.

[212] Benjamin Ramhorst, Vladimir Lončar, and George A Constantinides. FPGA resource-aware structured pruning for real-time neural networks. In *2023 International Conference on Field Programmable Technology (ICFPT)*, pages 282–283. IEEE, 2023.

[213] Jef Plochaet and Toon Goedemé. Hardware-aware pruning for FPGA deep learning accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4481–4489, 2023.

[214] Xuefu Sui, Qunbo Lv, Liangjie Zhi, Baoyu Zhu, Yuanbo Yang, Yu Zhang, and Zheng Tan. A hardware-friendly high-precision cnn pruning method and its FPGA implementation. *Sensors*, 23(2):824, 2023.

[215] Gianmarco Dinelli, Gabriele Meoni, Emilio Rapuano, Tommaso Pacini, and Luca Fanucci. Mem-opt: A scheduling and data re-use system to optimize on-chip memory usage for cnns on-board FPGAs. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(3):335–347, 2020.

[216] Xuan-Quang Nguyen and Cuong Pham-Quoc. An FPGA-based convolution ip core for deep neural networks acceleration. *REV Journal on Electronics and Communications*, 12(1-2), 2022.

[217] S Kala, Jimson Mathew, Babita R Jose, and S Nalesh. Uniwig: Unified winograd-gemm architecture for accelerating cnn on FPGAs. In *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, pages 209–214. IEEE, 2019.

[218] Xuan Wang, Chao Wang, Jing Cao, Lei Gong, and Xuehai Zhou. Winonn: Optimizing FPGA-based convolutional neural network accelerators using sparse winograd algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4290–4302, 2020.

[219] Chun Bao, Tao Xie, Wenbin Feng, Le Chang, and Chongchong Yu. A power-efficient optimizing framework FPGA accelerator based on winograd for yolo. *Ieee Access*, 8:94307–94317, 2020.

[220] Yingjie Wei, Rugang Wang, Yuanyuan Wang, Feng Zhou, and Naihong Gou. Optimizing FPGA-Based Target Detection: A Hybrid Winograd-GEMM Accelerator with Enhanced Resource Efficiency and Throughput. *Research Square pre-print*, 2024.

[221] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[222] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[223] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[224] Lit-Yang Liew and Sheng-De Wang. Object detection edge performance optimization on fpga-based heterogeneous multiprocessor systems. In *2022 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6. IEEE, 2022.

[225] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, and Jason Cong. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In *Proceedings of the ACM Turing Award Celebration Conference-China 2023*, pages 47–48, 2023.

[226] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C Ling, and Gordon R Chiu. An opencl™ deep learning accelerator on arria 10. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 55–64, 2017.

[227] Krishna Teja Chitty-Venkata and Arun K Somani. Neural architecture search survey: A hardware perspective. *ACM Computing Surveys*, 55(4):1–36, 2022.

[228] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021.

[229] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Accuracy vs. efficiency: Achieving both through FPGA-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[230] Liqiang Lu, Yun Liang, Qingcheng Xiao, and Shengen Yan. Evaluating fast algorithms for convolutional neural networks on FPGAs. In *2017 IEEE 25th annual international symposium on field-programmable custom computing machines (FCCM)*, pages 101–108. IEEE, 2017.

[231] Bing Liu, Danyin Zou, Lei Feng, Shou Feng, Ping Fu, and Junbao Li. An FPGA-based cnn accelerator integrating depthwise separable convolution. *Electronics*, 8(3):281, 2019.

[232] Hongbo Zhang, Jiaqi Jiang, Yunhao Fu, and Yuchun Chang. Yolov3-tiny object detection soc based on FPGA platform. In *2021 6th International Conference on Integrated Circuits and Microsystems (ICICM)*, pages 291–294. IEEE, 2021.

[233] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[234] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[235] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in neural information processing systems*, 29, 2016.

[236] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

[237] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[238] Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. Lookahead: A far-sighted alternative of magnitude-based pruning. *arXiv preprint arXiv:2002.04809*, 2020.

[239] Xia Xiao, Zigeng Wang, and Sanguthevar Rajasekaran. Autoprune: Automatic network pruning by regularizing auxiliary parameters. *Advances in neural information processing systems*, 32, 2019.

[240] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks.(2019). *arXiv preprint cs.LG/1902.09574*, 2019.

[241] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

[242] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[243] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[244] Fartash Faghri, Iman Tabrizian, Ilia Markov, Dan Alistarh, Daniel M Roy, and Ali Ramezani-Kebrya. Adaptive gradient quantization for data-parallel sgd. *Advances in neural information processing systems*, 33:3174–3185, 2020.

[245] Brian Chmiel, Liad Ben-Uri, Moran Shkolnik, Elad Hoffer, Ron Banner, and Daniel Soudry. Neural gradients are near-lognormal: improved quantized and sparse training. *arXiv preprint arXiv:2006.08173*, 2020.

[246] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[247] Gaofeng Zhou, Jianyang Zhou, and Haijun Lin. Research on nvidia deep learning accelerator. In *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 192–195. IEEE, 2018.

[248] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4350–4359, 2019.

[249] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5456–5464, 2017.

[250] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 580–595, 2018.

[251] Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7873–7882, 2018.

[252] Peng Hu, Xi Peng, Hongyuan Zhu, Mohamed M Sabry Aly, and Jie Lin. Opq: Compressing deep neural networks with one-shot pruning-quantization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 7780–7788, 2021.

[253] Hongyang Liu, Sara Elkerdawy, Nilanjan Ray, and Mostafa Elhoushi. Layer importance estimation with imprinting for neural network quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2408–2417, 2021.

[254] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H Hassoun. Post-training piecewise linear quantization for deep neural networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 69–86. Springer, 2020.

[255] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[256] Md Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I Morshed. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1):42–91, 2022.

[257] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[258] Chenghao Wang and Zhongqiang Luo. A review of the optimal design of neural networks based on FPGA. *Applied Sciences*, 12(21):10771, 2022.

[259] Shayan Moini, Bijan Alizadeh, Mohammad Emad, and Reza Ebrahimpour. A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(10):1217–1221, 2017.

[260] Aleksandar Beric, Jef van Meerbergen, Gerard de Haan, and Ramanathan Sethuraman. Memory-centric video processing. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(4):439–452, 2008.

[261] Shmuel Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980.

[262] Henri J Nussbaumer. *The Fast Fourier Transform*. Springer, 1982.

[263] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4013–4021, 2016.

[264] Anouar Nechi, Lukas Groth, Saleh Mulhem, Farhad Merchant, Rainer Buchty, and Mladen Berekovic. FPGA-based deep learning inference accelerators: Where are we standing? *ACM Transactions on Reconfigurable Technology and Systems*, 16(4):1–32, 2023.

[265] Rui Xu, Sheng Ma, Yaohua Wang, Yang Guo, Dongsheng Li, and Yuran Qiao. Heterogeneous systolic array architecture for compact cnns hardware accelerators. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2860–2871, 2021.

[266] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 45–54, 2017.

[267] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. An automatic rtl compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2017.

[268] Chaoyang Zhu, Kejie Huang, Shuyuan Yang, Ziqi Zhu, Hejia Zhang, and Haibin Shen. An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(9):1953–1965, 2020.

[269] Sparsh Mittal. A survey of FPGA-based accelerators for convolutional neural networks. *Neural computing and applications*, 32(4):1109–1139, 2020.

[270] Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 19:64–77, 2018.

[271] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[272] Yu Xing, Shuang Liang, Lingzhi Sui, Zhen Zhang, Jiantao Qiu, Xijie Jia, Xin Liu, Yushun Wang, Yi Shan, and Yu Wang. Dnnvm: End-to-end compiler leveraging operation fusion on FPGA-based cnn accelerators. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 187–188, 2019.

[273] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[274] Marco Siracusa, Lorenzo Di Tucci, Marco Rabozzi, Samuel Williams, Emanuele Del Sozzo, and Marco D Santambrogio. A cad-based methodology to optimize hls code via the roofline model. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[275] Bruno Da Silva, An Braeken, Erik H D'Hollander, and Abdellah Touhafi. Performance modeling for FPGAs: extending the roofline model with high-level synthesis tools. *International Journal of Reconfigurable Computing*, 2013:7–7, 2013.

[276] Philippe Coussy, Daniel D Gajski, Michael Meredith, and Andres Takach. An introduction to high-level synthesis. *IEEE Design & Test of Computers*, 26(4):8–17, 2009.

[277] Enrico Calore and Sebastiano Fabio Schifano. Performance assessment of FPGAs as hpc accelerators using the FPGA empirical roofline. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pages 83–90. IEEE, 2021.

[278] Guoqing Li, Jingwei Zhang, Meng Zhang, and Henk Corporaal. An efficient FPGA implementation for real-time and low-power uav object detection. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1387–1391. IEEE, 2022.

[279] Pipelining. https://www.intel.com/content/www/us/en/docs/oneapi-fpga-add-on/optimization-guide/2023-1/pipelining-001.html. (Accessed on 03/29/2024).

[280] Xiaowei Xu, Xinyi Zhang, Bei Yu, Xiaobo Sharon Hu, Christopher Rowen, Jingtong Hu, and Yiyu Shi. Dac-sdc low power object detection challenge for uav applications. *IEEE transactions on pattern analysis and machine intelligence*, 43(2):392–403, 2019.

[281] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[282] Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators. *IEEE Transactions on Computers*, 70(8):1160–1174, 2021.

[283] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[284] Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. Ternary neural networks for resource-efficient ai applications. In *2017 international joint conference on neural networks (IJCNN)*, pages 2547–2554. IEEE, 2017.

[285] Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks*, 100:49–58, 2018.

[286] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. Rebnet: Residual binarized neural network. In *2018 IEEE 26th annual international symposium on field-programmable custom computing machines (FCCM)*, pages 57–64. IEEE, 2018.

[287] Dominika Przewlocka-Rus, Syed Shakib Sarwar, H Ekin Sumbul, Yuecheng Li, and Barbara De Salvo. Power-of-two quantization for low bitwidth and hardware compliant neural networks. *arXiv preprint arXiv:2203.05025*, 2022.

[288] Tian Xia, Boran Zhao, Jian Ma, Gelin Fu, Wenzhe Zhao, Nanning Zheng, and Pengju Ren. An energy-and-area-efficient cnn accelerator for universal powers-of-two quantization. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(3):1242–1255, 2022.

[289] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[290] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[291] Jude Haris, Perry Gibson, José Cano, Nicolas Bohm Agostini, and David Kaeli. Secda: Efficient hardware/software co-design of FPGA-based dnn accelerators for edge inference. In *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 33–43. IEEE, 2021.

[292] Afzal Ahmad, Muhammad Adeel Pasha, and Ghulam Jilani Raza. Accelerating tiny yolov3 using FPGA-based hardware/software co-design. In *2020 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2020.

[293] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. Energy-efficient cnn implementation on a deeply pipelined FPGA cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 326–331, 2016.

[294] Yasuyu Fukushima, Kensuke Iizuka, and Hideharu Amano. Parallel implementation of cnn on multi-FPGA cluster. *IEICE TRANSACTIONS on Information and Systems*, 106(7):1198–1208, 2023.

[295] Yasuyu Fukushima, Kensuke Iizuka, and Hideharu Amano. Parallel implementation of vision transformer on a multi-FPGA cluster. In *2023 Eleventh International Symposium on Computing and Networking (CANDAR)*, pages 100–106. IEEE, 2023.

[296] Fadi Thabtah, Suhel Hammoud, Firuz Kamalov, and Amanda Gonsalves. Data imbalance in classification: Experimental evaluation. *Information Sciences*, 513:429–441, 2020.

[297] Rafael Padilla, Wesley L Passos, Thadeu LB Dias, Sergio L Netto, and Eduardo AB Da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3):279, 2021.

[298] Marco Siracusa, Emanuele Del Sozzo, Marco Rabozzi, Lorenzo Di Tucci, Samuel Williams, Donatella Sciuto, and Marco Domenico Santambrogio. A comprehensive methodology to optimize FPGA designs via the roofline model. *IEEE Transactions on Computers*, 71(8):1903–1915, 2021.

SEYED HANI HOZHABR received the B.S. degree from the University of Applied Science and Technology in electronic engineering and M.S. degrees in electrical engineering-electronics from Malek-Ashtar University of Technology, Tehran, Iran. He is currently working toward a PhD degree in information engineering and science at the University of Siena, Siena, Italy. Since 2022, he has been actively collaborating with the Computer Architecture Lab as a researcher. His current research interests include hardware implementation, FPGA-based AI accelerators, and SoC design for vision applications.

ROBERTO GIORGI (M'88–SM'04) Roberto Giorgi is an Associate Professor at the Department of Information Engineering, University of Siena, Italy (qualified for Full Professorship). He received his PhD in Computer Engineering and his Master in Electronics Engineering, Summa cum Laude both from the University of Pisa, Italy. He has been the coordinator of a 4-year Future and Emerging Technology European project (TERAFLUX), coordinator of a 3-year H2020 project (AXIOM), Workpackage leader of the Embedded Reconfigurable Architecture project, deputy steering committee member in HiPEAC (High-Performance Embedded-system Architecture and Compiler), participating in SARC (Scalable ARChitectures). He participated in the ChARM project, developing software for performance evaluation of ARM-processor-based embedded systems with cache memory. He has been selected by the European Commission as an ICT/HPC independent expert. He is the author of more than 130 scientific papers. His current interests include Computer Architecture themes such as Embedded Systems, Multiprocessors, Memory System Performance, Workload Characterization, and Reconfigurable Computing. He is a Lifetime member of ACM and a Senior Member of the IEEE, IEEE Computer Society.