A Systematic Assessment of Embedded Neural Networks for Object Detection

Micaela Verucchi*, Gianluca Brilli*, Davide Sapienza*, Mattia Verasani†, Marco Arena†,
Francesco Gatti*‡, Alessandro Capotondi*, Roberto Cavicchioli*, Marko Bertogna and Marco Solieri*
*Università di Modena e Reggio Emilia, Italy - name.surname@unimore.it
‡Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy - 189382@studenti.unimore.it
†Tetra Pak, Italy - name.surname@tetrapak.com

Abstract—**Object detection is arguably one of the most important and complex tasks to enable the advent of next-generation autonomous systems. Recent advancements in deep learning techniques allowed a significant improvement in detection accuracy and latency of modern neural networks, allowing their adoption in automotive, avionics and industrial embedded systems, where performances are required to meet size, weight and power constraints.**

Multiple benchmarks and surveys exist to compare state-of-the-art detection networks, profiling important metrics, like precision, latency and power efficiency on Commercial-off-the-Shelf (COTS) embedded platforms. However, we observed a fundamental lack of fairness in the existing comparisons, with a number of implicit assumptions that may significantly bias the metrics of interest. This includes using heterogeneous settings for the input size, training dataset, threshold confidences, and, most importantly, platform-specific optimizations, that are especially important when assessing latency and energy-related values. The lack of uniform comparisons is mainly due to the significant effort required to re-implement network models, whenever openly available, on the specific platforms, to properly configure the available acceleration engines for optimizing performance, and to re-train the model using a homogeneous dataset.

This paper aims at filling this gap, providing a comprehensive and fair comparison of the best-in-class Convolution Neural Networks (CNNs) for real-time embedded systems, detailing the effort made to achieve an unbiased characterization on cutting-edge system-on-chips. Multi-dimensional trade-offs are explored for achieving a proper configuration of the available programmable accelerators for neural inference, adopting the best available software libraries. To stimulate the adoption of fair benchmarking assessments, the framework is released to the public in an open source repository.

I. Introduction

Visual detection is a pervasively-used technique that consists in finding, within digital images or video streams, instances of semantic objects of some predefined categories.

An autonomous car driving application typically detects road users, such as other cars, bicycles and pedestrians [1], [2]. An application in a highly-automated industrial machine [3] instead, could detect objects to be manipulated, or defects to be signalled. Many are the other applicative domains, ranging from robotics [4], to avionics [5] or simply surveillance. Many also are the vision tasks that can be built upon object detection, such as instance segmentation, image captioning, or object tracking.

Partially supported by Tetra Pak and by the H2020 CLASS Project (G.A. 780622). The first three authors contributed equally to the presented work.

Since 2012, Deep Neural Networks (DNNs) have surpassed the accuracy of classical methods, becoming the state-of-the-art technology for vision task [6].

A. Conflicting Goals in Designing a Solution

a) Performance: A good object detector is characterised by two main performance figures: *latency*, i.e. the time needed for a single frame to be processed, and *accuracy*, i.e. the quality of the output given the input. These represent a well-known bi-dimensional trade-off: improving one often worsen the other.

b) Power: When targeting embedded platforms, one-stage detectors are preferable to two-stage ones, because they are faster, and this leaves room for further trade-offs against the limited resources of the computing platform. Some of these constraints, like the platform's physical characteristics or price, are independent from performance indicators like the ones above. The *power absorption*, instead, directly correlates with the attainable precision and latency, hence compelling the system design to a third trade-off dimension.

B. Strategies to Optimise Inference

Regarding fast inference on-device computation, three major axes efforts have been identified [7] to maximise performances or contain power consumption.

a) Network model design: Reducing the number of parameters in the DNN model is very common approach to reduce memory and execution latency, while aiming at preserving high accuracy. Some examples include MobileNets [8], Single-Shot Detectors (SSD) [9], Yolo [10], and SqueezeNet [11], with the state of the art that is evolving rapidly. We consider Yolov3 and Yolov3-tiny [12], Mobilenetv2-SSDLite [13], Centernet-Resnet101 and Centernet-DLA34 [14], designed to achieve high throughput.

b) Model Compression: DNN models can also be compressed sacrificing a small accuracy loss compared with the original model. There are several popular model compression methods: parameter quantization, parameter pruning, and knowledge distillation. In this paper, we use quantization such as half floating-point precision (FP16) or INT8 inference;

c) Platforms: The x86 CPU architectures offer the greatest sequential performances (operation throughput and response time) and easiest programmability, and have long been dominating the high-end segment of industrial automation domain to deliver central control systems. General-Purpose computing on Graphics Processing Units (GPGPUs) revolutionised the offer by providing order-of-magnitude improvements in parallel throughput and in power usage, at a reasonable programming cost; Field-Programmable Gate Array (FPGA) platforms share a similar ambition, asking for higher programming cost, but providing more flexible communication paradigm and simpler computational units. There also is an emerging class of AI-dedicated accelerator implemented with Application-Specific Integrated Circuits (ASICs), e.g. Google TPU, Huawei NPU, and Intel Nervana NNP. Their design provides by construction the best inference efficiency, but we leave them out of our scope for their limited flexibility and evolvability, which is important for keeping the fast pace of research on object detection and NNs.

C. The Need of a Systematic Assessment

Given the wideness of optimisation means currently employed, and the number of conflicting and diversely-relevant goals, the existing reviews of the literature did not dare to address a comprehensive inspection of a such fragmented state of the art in embedded object detection. This paper goes beyond by providing a systematic assessment, evaluating all the combinations of aforementioned network, compression and platforms, against all the aforementioned evaluation metrics.

D. Contributions

This paper deliver the followings.

- A review of the state-of-the-art (SOTA) regarding embedded neural networks for Object Detection Convolutional Neural Networks (ODCNN).
- An optimized implementation of 5 different SOTA ODCNNs, to better exploit the embedded boards, it is open-source licensed and available for the Xavier AGX[1] and for the ZCU102 Ultascale+[2].
- A fair comparison of 5 different SOTA ODCNNs, in terms of mean Average Precision (mAP), latency and power consumption on three different embedded boards, i.e. the NVIDIA Xavier AGX, the Xilinx Zynq ZCU102 Ultrascale+ and an Industrial PC (IPC) i7-7700. To be as fair as possible same input size, same training dataset, same threshold for bounding boxes' (BBs) confidence and best implementation on each platform have been considered.

E. Outline

The following section offers an overview of the SOTA and tries to clarify what are the available results in the literature and what is missing. Section III explains the platforms chosen for the experiments and their major features, while Section IV is focused on the details of the considered networks and their porting on the chosen runtimes. Finally, all the results are shown in Section V, where also all the conclusions are reported.

II. Related Works

There exist several works that aim at improving a single Neural Network (NN) on a specific embedded platform, but not as many that offer a good comparison between different methods on different boards. While the former task is a natural process when developing a new method, the latter could require more effort. We summarized a curated selection of SOTA comparison works in Table I. Columns are grouped in four macro-areas: (i) inference platforms considered, (ii) ODCNN adopted for the task of object detection and classification, ordered by ascending release year, (iii) dataset used for training, (iv) metrics adopted to evaluate the results.

It is not trivial to run correctly the same NNs on different embedded platforms. An implementation from scratch could be needed for the following reasons: (i) the framework needed could not be supported,(ii) the framework needed is supported but its performance is degraded on specific hardware, (iii) great improvements can be obtained using a specific library for specific target hardware. Moreover, when comparing NNs, other factors should be taken into account and fixed to be fair, i.e. adopting the same dataset for training, using the same input size for the networks, consider the same thresholds to compute the accuracy, use the same data-type for the weights. Finally, a comparison could be performed for a single metric or, to be more exhaustive, for many.

For example, Rungsuptaweekoon et al. evaluate in [15] the power efficiency of the object detector Yolov2 on NVIDIA Jetson TX1, Jetson TX2, and Tesla P40. The authors compare mAP, accumulated energy consumption, mAP/Energy and frame rate factors. Even though all the metrics are considered, only one NN is investigated and only NVIDIA platforms. On the other hand, Yu et a. compare in [17] real-time object detection algorithms on several embedded platforms. They measure power efficiency, latency and accuracy of Faster RCNN, Yolo and SSD on NVIDIA TK1, Xilinx Zynq 7045 and Xilinx KU115. However the comparison is not very fair: not all the networks are implemented on all the boards, the datasets are different and they do not even mention the input size of the networks. An extensive comparison is offered by Lin et al. [20], where the authors benchmark several deep learning frameworks and investigate the FPGA deployment for performing traffic sign classification and detection. To evaluate inference performance, they consider inference latency, accuracy, and power efficiency, by varying different parameters such as floating-point precision and batch size. They show that TensorFlow is always among the frameworks with the highest inference accuracy. For object detection inference, they compare six SSD models with different base networks on an NVIDIA GTX 1050 Ti GPU and an Arria 10 FPGA development board. Compared to the reference results on the GPU, they notice that in most of cases inference speed on the GPU is higher than the FPGA, as well as accuracy. However, FPGA always achieves higher power efficiency than the GPU. It is also worth mentioning MLPerf [22] and MLMark[3] (main-

[1]https://github.com/ceccocats/tkDNN
[2]https://git.hipert.unimore.it/gbrilli/dpunn
[3]https://www.eembc.org/mlmark/scores.php

TABLE I: SOTA comparison for Object Detection and Classification Neural Networks.

	Platform								Object Detection and Classification Neural Network												Dataset		Metric		
	TK1	TX1	TX2	Xavier AGX	Zynq 70xx	Zynq ZCU102	Myriad-X	Not embedded	RCNN	FasterRCNN	VGGSSD	SqueezeNet	Yolo	MobileNetv1-SSD	Yolov2-tiny	Yolov2	MobileNetv2-SSDLite	Yolov3-tiny	Yolov3	Centernet	COCO	Other dataset	Time performance	mAP	Power
Year (20-)	14	15	16	19	12	14	17	-	14	15	15	16	16	17	17	17	18	18	18	19	14	-	-	-	-
[15]		✓	✓													✓					✓		✓	✓	✓
[16]			✓				✓		✓		✓	✓			✓	✓		✓	✓			✓	✓	✓	
[17]	✓				✓								✓									✓	✓	✓	✓
[18]			✓			✓										✓						✓	✓		✓
[19]		✓	✓	✓			✓			✓				✓	✓	✓		✓	✓			✓	✓		
[20]								✓		✓	✓	✓		✓			✓				✓		✓	✓	✓
[21]			✓		✓			✓							✓							✓	✓		✓
MLPerf [22]				✓			✓							✓							✓		✓		
MLMark				✓										✓							✓	✓	✓	✓	
this paper				✓		✓											✓	✓	✓	✓	✓		✓	✓	✓

tained Embedded Microprocessor Benchmark Consortium), two efforts in gathering a considerable number of performance records, offering a collection point for researchers. For both of them, the comparison is however difficult or impossible, due to the heterogeneity of collected tests, e.g. input size, dataset, threshold, and other details are omitted. Moreover, only a few CNN for object detections appear among the results, and the chosen metrics are limited to latency for MLPerf, latency and mAP for MLMark.

In this paper, three kinds of embedded platforms have been chosen: an Industrial PC to compare with the latest embedded board by NVIDIA and Xilinx, namely, the Xavier AGX and the ZCU102 Zynq UltraScale+. Regarding the methods, we decided to pick five of the most recent NNs, therefore we selected Yolov3 (Yolov3-tiny) over Yolov2, Mobilenetv2-SSDLite over Mobilenetv1-SSD and we also investigate two versions of a newly released detector, Centernet, which has not been evaluated on embedded platforms yet. Lastly, an exhaustive analysis has been carried on, comprehensive of latency, mean Average Precision (mAP) and power consumption.

III. Inference Platforms

A. GPGPU

The NVIDIA Xavier AGX is a recently released embedded platform featuring a Tegra System on Chip (Xavier AGX SoC) composed of 8 NVIDIA Carmel CPU Cores and an integrated GPU based on the NVIDIA Volta architecture with 512 CUDA cores and 64 Tensor Cores. Tensor Cores are programmable fused matrix-multiply-and-accumulate units that execute concurrently alongside CUDA cores and implement HMMA (Half-Precision Matrix Multiply and Accumulate) and IMMA (Integer Matrix Multiply and Accumulate) instructions for accelerating various applications and, in our interest, deep learning inference.

TensorRT is a framework provided by NVIDIA[4] and written in CUDA for optimizing the inference of deep learning models on their GPU. Using TensorRT allows one to reduce the precision data type, performing inference at 8-bit integer (INT8), or at half-precision floating-point (FP16) to replace the single-precision floating-point (FP32) in representing the weights and parameters of deep learning models. A common result [16][19][23] is that the overall latency of the model can be dramatically reduced, though the final accuracy could be degraded. However, exploiting this framework is not always trivial, especially if the networks that need to be ported have unusual layers (e.g. deformable convolutional layers). Therefore the effort of the programmer is not negligible and several plugins need to be implemented.

Lastly, tkDNN is our open-source Deep Neural Network library built with cuDNN and tensorRT primitives, specifically thought to work on NVIDIA Jetson Boards, whose main goal is to exploit those boards as much as possible to obtain the best inference performance.

B. FPGA

As a point of reference for FPGA-based System-on-Chip, we used the XCZU9EG SoC belonging to the Zynq UltraScale+ family, through the Xilinx ZCU102 development board [24]. Specifically, this SoC is composed of a *Processing System (PS)* having 4 ARM Cortex A53 CPU Cores for the user applications part and an ARM Cortex R5 dual-cores processor to handle Hard Real-Time applications. Next to the PS, there is a *Programmable Logic (PL)*, which hosts a processing unit for managing Deep Learning tasks, called *Deep Learning Processing Unit (DPU)* [25]. This system is a soft-processor (IP core) implemented in FPGA, containing a certain number of specialized engines for typical Deep Learning operations such as convolutions, pooling, etc. In this case, the A53 cores of the PS part only deal with (i) performing pre/post-processing operations and executing layers not supported by the accelerator; (ii) transmitting the images to the DPU; (iii) configuring the operations that the DPU will carry out, encoded in a specific binary file.

As development frameworks, to create custom NN models, Xilinx provides DNNDK [26] and Vitis-AI [27] suites, that contain some commonly used network models, collected in

[4]https://developer.nvidia.com/tensorrt

TABLE II: Comparison of the considered boards. N.U. stands for Not Used for the implementation.

	Embedded IPC	NVIDIA Xavier AGX	Xilinx Zynq UltraScale+ MPSoC ZCU102
CPU	Intel i7-7700 4 cores @3.60GHz	NVIDIA Carmel (Arm v8.2) 8 cores @2.13GHz	Arm Cortex-A53 (v8) 4 cores @1.5GHz
GPU	-	512-core Volta GPU with Tensor Cores	Mali-400 [N.U]
DNN Accelerator	-	2 Deep Learning Accelerators (DLAs)[N.U.]	3 Deep Learning Processing Units Soft IPs (DPUs)
Memory	16 GB RAM	16 GB 256-bit LPDDR4x	4 GB DDR4 64-bit SODIMM w/ ECC (PS)
Supported datatype	FP32	FP32, FP16, INT8	INT8
Operating system	Windows 10 Enterprise LTSC, Version 1809	Ubuntu 18.04.3 LTS, Jetpack 4.3	Debian Buster 10.0
Framework used	ONNX Runtime	tkDNN with TensorRT	DNNDK and Vitis-AI

their “model zoo” repository and tools that can be used to quantize and deploy the networks. These suites are integrated with Caffe [28] and Tensorflow [29] frameworks, through which it is possible to define the model of a NN and export the trained weights. Other NN engines for FPGA exists, i.e. Neuraghe presented by Meloni et al. [30], CHaiDNN developed by Xilinx [31] and FINN proposed by Umuroglu et al. [32] (natively supported for SoCs that integrate the Pynq framework e.g. Avnet Ultra96). However, in this work we focus on DPU TRD IP, because: (i) it is directly supported by Xilinx; (ii) it is well documented both for hardware design and drivers API; (iii) through the Vitis-AI and the Vitis Model Zoo repository it is possible to deploy a wide range of NNs, including Yolov3 and Yolov3-Tiny.

C. IPC

In industrial automation there are many advantages to PC based control: lower costs, maintainability, tried and true hardware with extensive processor power and nearly unlimited memory. In many cases there is a software layer that takes control of the processor, creating a Real-Time processing environment. Windows OS becomes a sub-process with limited time slices of processor time, while control, visualisation and motion tasks are multi-tasked on the processor. Compliance with the global automation standard, IEC-61131–3 is another advantage of PC control.

The development framework chosen for this platform is ONNX Runtime [33], a cross-platform, high-performance inference engine for ONNX (Open Neural Network Exchange [34]) for Machine Learning and Deep Learning models. It allows systems to integrate a single inference engine that supports models trained from a variety of frameworks, while automatically taking advantage of specific supported hardware accelerators and runtimes available on different platforms. The ONNX Runtime C++ API has been exploited, to guarantee relevant performances during the pre-processing and post-processing of the images. For ONNX Runtime the models have been exported from Pytorch 1.4 [35]. Pytorch natively supports ONNX layer export. All the conversions to the ONNX format exploit the Constant Folding graph optimization technique. A smooth conversion is guaranteed if Pytorch operators are supported by ONNX. Custom Pytorch operators can be still registered as ONNX operators through the ONNX Registration API. ONNX stable opset supported by Pytorch 1.4 is version 9. We have encountered no problem to exploit also opset version 11 for the conversion of the model studied. All the specifications of the chosen IPC, and details of all the other inference platforms are reported in Table II.

IV. Neural Networks

When picking the ODCNNs we wanted to find the best trade-off between execution time and mean average precision. In literature, there exist several surveys [36][37][6] on object detection, however, all of them focus on the mAP metric. Eventually, we picked three kinds of NNs that are designed to run in real-time, are well-established in literature and lead to the best results[5].

A. Yolo3

Yolov3 [12] is a one-stage detector which divides images into grid cells and predicts BBs using dimension clusters as anchor boxes. It adopts independent logistic classifiers to output an object score for each BB. The BBs are predicted at three different scales through extracting features from these scales. Yolov3 uses a backbone network, named Darknet-53, for performing feature extraction, which is a residual network with 53 convolutional layers. Due to the introduction of Darknet-53 and multi-scale feature maps, Yolov3 achieves great speed improvement and improves the detection accuracy of small-sized objects when compared with Yolov2 [38].

The porting of the original network, as it is, was possible on the Xavier AGX, XCZU9EG and i7-7700 boards.

B. Yolo3-tiny

Yolov3-tiny is a lighter version of Yolov3. It uses the same concepts (independent logistic and anchors), but the backbone is composed of only 10 convolutional layers and there are only two scales (therefore 6 anchors rather than 9). It has been chosen for this evaluation for its high inference speed.

On the Xavier AGX platform, the porting of the original model was possible. On the other hand, a small change has been required for the XCZU9EG implementation. Precisely, the last max-pool layer has been removed as it was not supported during the deployment phase of the model on the Xilinx DPU platform. The issue is caused by a mismatch between how Darknet and Caffe frameworks handle max-pooling layers of stride 1, with input size equal to output size. However, removing this single layer did not affect too much the performance of the network. The mAP of the network has been measured with and without the aforementioned layer and the tests showed a negligible accuracy drop (from a mAP of $11,9\%$ to $11,7\%$, computed on the Xavier AGX). The conversion of Yolov3 and Yolov3-tiny to ONNX has been done automatically by the Pytorch to ONNX exporter, without adding new operators.

[5]https://paperswithcode.com/sota/real-time-object-detection-on-coco (Accessed 2019-10-08)

C. Mobilenetv2-SSDLite

MobileNetv2 [13] is an efficient CNN model with depth-wise convolution layers, that have fewer weights compared with normal convolution layers. It is one of the most used models for embedded systems because it is lightweight and can achieve high FPS also on mobile devices. SSD [9] is a one-stage detector which divides images into grid cells, and for each grid cell, uses a pre-generated set of anchors with multiple scales and aspect-ratios to discretize the output space of BBs. SSD predicts objects on multiple feature maps, and each of them is responsible for detecting a certain scale of objects, according to its receptive fields.

On the Xavier AGX platform, the porting of the original model was possible. To obtain the original model on ONNX Runtime, it is necessary to highlight the switch from the ReLU6 Pytorch operator to the Clip ONNX operator with minimum value 0 and max value of 6. Finally, regarding the XCZU9EG we made a model conversion to Caffe [28] through open-source code[6], since Pytorch framework is currently not supported by the Xilinx ecosystem. Subsequently, all the final Reshape layers were replaced, in favor of Flatten type layers, since the first ones are not supported by the DPU architecture. In addition, the final Softmax layer has been removed from the model and implemented in Software and accelerated with OpenMP, because the memory movements between PL and PS would have resulted in worse performance.

D. CenterNet

Centernet [14] proposes modeling an object as a single point. It uses key point estimation to find center points and regresses all other object properties including 3D location, pose orientation, and size. In this model, an image is fed to a CNN which generates a heatmap, whose maximum values represent the centers of the objects in the image. The objects' size and pose are regressed from features of the image at the center location. CenterNet was tested with four different backbones, i.e. ResNet18, ResNet101, DLA34 and Hourglass, substituting the convolutional layers with deformable convolutional layers v2 [39]. Deformable convolutional networks (DCN) [40] are detectors able to adapt to the geometric variations of objects. Regular convolutional networks can only focus on features of fixed square size (according to the kernel), thus the receptive field does not properly cover each pixel of a target object to represent it. The DCN produce a deformable kernel and the offset from the initial convolution kernel (of fixed size) is learned during training.

Based on the results of the paper, we picked the two backbones with higher mAP and throughput. ResNet-18 was excluded because it has lower mAP than Yolov3, and Hourglass because of its poor inference speed. Due to the particular structure of this model, the porting on the NVIDIA platform required a great implementation effort, while the porting on the Xilinx platform was not possible. Indeed, at the moment the DPU microarchitecture does not support deformable convolutional layers. Although the DNNDK / Vitis-AI ecosystem supports the integration of software-implemented layers, the resulting network could not achieve satisfactory performance due to the inefficiency of the A53 cores in carrying out convolution operations with respect to the DPU cores, and due to the continuous shift of weights and activations from PS to PL. Neither was it possible to convert and properly port Centernet on ONNX Runtime. To the best of our knowledge, there is no supported DCNv2 operator for Pytorch, ONNX and ONNX Runtime for CPU. Further work involves the implementation of DCNv2 as Pytorch custom operator for CPU, its registration as a custom ONNX operator and as an ONNX Runtime operator.

[6]https://github.com/xxradon/PytorchToCaffe

E. Training

To properly compare ODCNNs performances, the configurations of all networks must be compliant. It is not fair to compare the latency of the networks for different input sizes, nor to compare the accuracy of networks trained for different input image resolution or, even worse, different datasets. Hence, we decided to choose one of the most widely used datasets in object detection, i.e. COCO [41]. Input size and training set were chosen after the newer network (CenterNet), using COCO 2017 with input size 512x512. This input size allows better discriminating the differences between the latencies, while achieving a good accuracy. COCO 2017 is divided into 118K images for training and 5K images for validation, but the classes are still the same 80 of the original version (2014). We trained Yolov3, Yolov3-tiny and Mobilenetv2-SSDLite, while for the Centernet networks we used the weights from the SOTA, that satisfied already our settings. A single training per network has been performed (in full-precision), and later the weights have been exported for the different frameworks.

V. Experiments

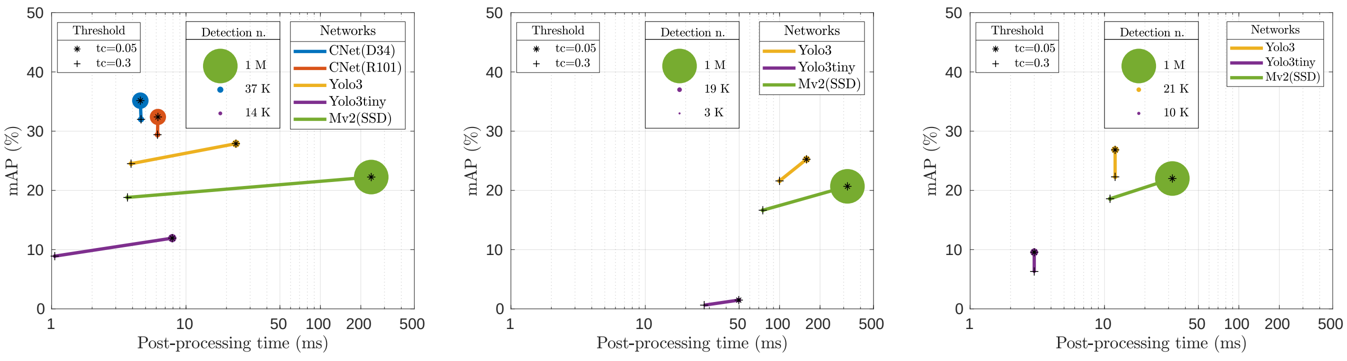
A. Experimental setup

All the tests have been performed on the COCO2017 validation tests (5K images). Worst-case end-to-end latency, average power consumption and mAP have been taken over all the 5K images. For the Xavier AGX platform, three data types where considered : FP32, FP16, and INT8. Only INT8 was considered for the XCZU9EG , and only FP32 for the i7-7700. The INT8 quantizations have been obtained on 1000 images of the COCO2017 training set. To maximize Xavier AGX performance, the mode has been set to MAX N and $jetson\_clocks$ has been launched before the tests. The i7-7700 cores have been divided into two groups: two are dedicated to a Real-Time environment and two used for these tests. From now on we will use CNet(D34) for Centernet-DLA34, CNet(R101) for Centernet-Resnet101, Yolo3 for Yolov3, Yolo3tiny for Yolov3-tiny and Mv2(SSD) for Mobilenetv2-SSDLite.

B. Metrics

For the sake of comparison, three metrics have been adopted: (i) worst-case end-to-end latency [ms]; (ii) mean Average Precision (mAP) 0.5:0.95 [%]; (iii) efficiency [FPS/W].

The *mAP 0.5:0.95* is the "de facto" metric for object detection [42] [43] and depends on two thresholds: (i) confidence threshold t_c, taking into account only BBs with a confidence score greater than t_c; and (ii) the IoU thresholds t_{IoU} to



(a) Xavier AGX (FP32) (b) XCZU9EG (INT8) (c) i7-7700 (FP32)

Fig. 1: Comparison of mAP (y-axis) versus worst case post-processing latency (x-axis) using confidence threshold $t_c = 0.05$ (*) or $t_c = 0.3$ (+). The radius of the point is proportional to the number of detections.

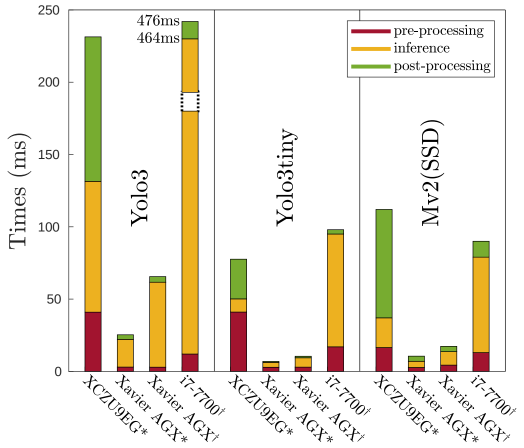


Fig. 2: Worst case execution time divided in pre-processing, inference and post-processing w.r.t. the end-to-end latency. * stands for INT8, † for FP32.

discriminate two BBs representing the same object if their classes match and if their IoU is greater than t_{IoU}.

The execution time of a method can be divided into 3 phases, i.e. (i) pre-processing to convert the image in the NN input, (ii) NN inference, (iii) post-processing to convert the output of a NN into BBs. The *end-to-end latency* is then the time elapsed between feeding an image to the detector and obtaining the BBs. In each board, the pre-processing and the post-processing is performed in full-precision (FP32), while the inference can be quantized (FP16 or INT8). Pre-/post-processing are performed on the CPU for every board, except for Xavier AGX, where pre-processing of every network and post-processing of CNet(D34) and CNet(R101) have been optimized on GPU, implementing a CUDA version of the corresponding (slower) OpenCV functions.

Finally, the *efficiency* is computed as Frame Per Seconds (FPS) over the power consumption (W). We only considered a batch of 1. The power has been sampled at 40Hz on the Xavier AGX and the XCZU9EG using *powerapp*[7], and at 1Hz on the i7-7700 using Open Hardware Monitor 0.9.2.

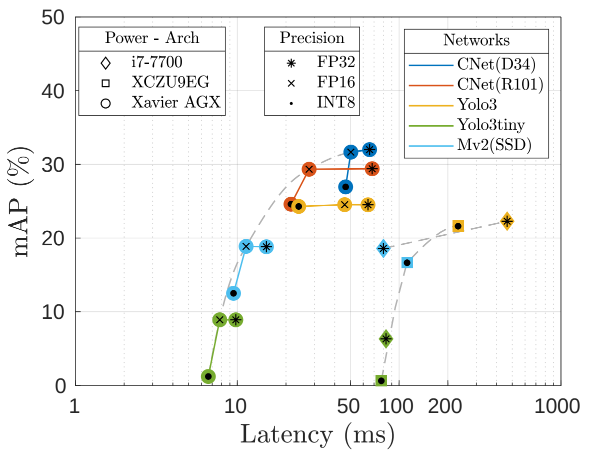
C. Impact of threshold on mAP and latency

To compute the mAP of a method, t_c is usually set to 0 (or 0.05, if the method does not allow 0, as for Yolo and SSD). However, most of the ODCNNs are used in applications with a $t_c = 0.3$. For this reason, we consider both $t_c = 0.05$ and $t_c = 0.3$. Figure 1 shows the results. In these charts, three aspects are considered: (i) the mAP of the network (y-axis); (ii) the worst-case post-processing latency of the network (x-axis); (iii) the number of detections (radius of the points). When changing the confidence threshold, more or less BBs are returned, affecting only the post-processing time. Varying the threshold affects the mAP and number of detections in all three platforms: when $t_c = 0.05$, mAP and number of detections are higher, increasing the post-processing latency to a different extent on the considered platforms. The Xavier AGX board has the highest variance in the post-processing phase, with a slowdown of 65× when using a smaller threshold for Mv2(SSD). The i7-7700 is way more stable, showing only a 3× slowdown with the smaller threshold for Mv2(SSD). The XCZU9EG is the slower platform in terms of post-processing time. The main reason for these differences is due to the post-processing operation being a sequential operation executed on the host side, with platforms having CPU clusters with very different performances (the Intel processor on the i7-7700 board has faster sequential performance than the ARM-based ones on the other two boards).

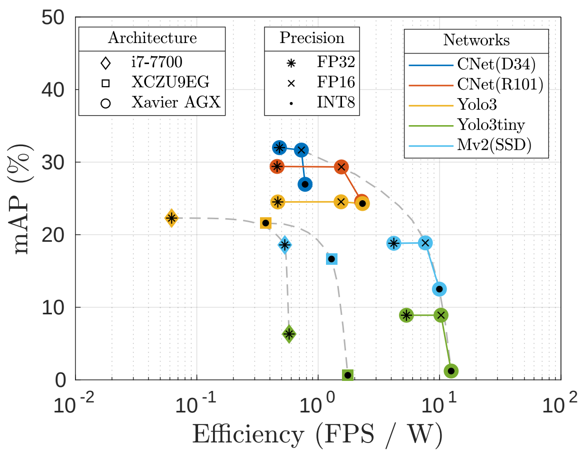
Considering mAP at full precision, it can be noticed that Xavier AGX obtains higher values than the i7-7700. Given that the models and the weights are the same, the small accuracy gap (below 1-2%) is due to the different runtimes and instruction sets architecture adopted – ONNX Runtime over x86-64 Intel's Kaby Lake, and TensorRT over PTX NVIDIA's Volta.

Regarding the networks, Yolo methods produce less BBs than the others, even with small thresholds, while the num-

[7]https://git.hipert.unimore.it/tetra-pak/dl-arch/powerapp



(a) Latency vs mAP



(b) Efficiency vs mAP

Fig. 3: Comparison of the different networks on the three platforms.

ber of detections for Mv2(SSD) explodes when using $t_c = 0.05$. CNet(D34) and CNet(R101) have stabler post-processing times, thanks to our optimization of this phase on the GPU.

In the following, we assume $t_c = 0.3$ to consider a realistic application scenario.

D. Platforms comparison

Figure 3a compares worst-case end-to-end latency in ms (x-axis) with mAP % (y-axis). On the top left corner, the fastest networks with higher accuracy can be found.

On the Xavier AGX, FP16 maintains the same mAP as FP32, while being faster. Indeed, FP16 points are predominant in the Pareto-optimality curve shown in grey-dashed lines.

The mAP of the different networks are similar on the various platform. Xavier AGX achieves the highest mAP for all networks except for Mv2(SSD) at INT8, where a higher mAP is obtained on the XCZU9EG. The Xavier AGX is also the platform that achieves the best latency performance for every considered network. No dominance relation can be extablished between the other two platforms in terms of latency or mAP.

Figure 2 shows the time spent on pre-processing, inference, and post-processing. For the XCZU9EG, the most expensive part is never the inference, which is performed on the DPU, but the pre- and post-processing, which are executed on a slower CPU. One of the reasons for the long post-processing can be found in the normalization functions. Due to missing API on the DPU (e.g. sigmoid for Yolo3 and Yolo3tiny) or slow implementations (i.e. softmax for Mv2(SSD)), those operations are executed on the CPU rather than on the accelerator (as in the Xavier AGX case). For Xavier AGX and i7-7700, inference represents the largest share of the end-to-end latency. Note that, for space reasons, we showed only results related to worst-case latencies. However, identical results have been found for average-case latencies.

Figure 3b compares efficiency in FPS/Watt (x-axis) with mAP in % (y-axis). On the top right corner, the most efficient networks with higher accuracy can be found. The i7-7700 is the platform that consumes the most. Being also slow, it is therefore the least efficient board according to this metric. In general, XCZU9EG is the board that consumes less. However, due to the low FPS achieved, it is not the most efficient one. Therefore, the Xavier AGX is confirmed as the most efficient board that also achieves the best mAP. Again, the FP16 is confirmed on the Pareto-optimality curve.

Regarding the power consumption, it is worth saying that the difference in power consumed by XCZU9EG in inference and idle mode is very small, as opposed to i7-7700 and Xavier AGX, where such difference is significant.

E. Networks comparison

Figure 3a shows that CNet(D34) always achieves the highest mAP. At FP32 it has a similar latency to CNet(R101) and Yolo3, but it is slower when using FP16 or INT8. Considering the Pareto optimality curve, it can be seen that Yolo3 is obsoleted by CNet(R101), which achieves a better accuracy and a lower latency. Yolo3tiny has always the lowest latency in each board, for each precision.

From Figure 3b, it can be noticed that Yolo3tiny and Mv2(SSD) are always the most efficient networks, with the former dominating the others in terms of efficiency. Yolo3 is again dominated by CNet(R101), and is the least efficient one on all platforms.

VI. Conclusions

This work represented a further step towards a systematic assessment of the performance of detection networks for high-performance embedded platforms. The best real-time object detection networks have been considered to be ported on three representative cutting-edge embedded boards, exploiting the corresponding ML frameworks. For the CenterNet models, only the porting on the Xavier AGX was possible, due to limitations of DPU and ONNX APIs.

An exhaustive evaluation of networks performance shown that the Pareto-optimality curve intercepts four of the five considered DNNs. Yolov3 is the only one that is dominated by the other ones. CenterNet is the one achieving the highest accuracy, while Yolov3-tiny is the best network in terms of latency and power consumption. Mobilenetv2-SSDLite is the most affected by the confidence threshold.

From a platform point of view, Xavier AGX is the clear winner in all considered aspects, achieving the best power efficiency as well as the highest mAP. The XCZU9EG has a very stable power consumption for all considered networks, and it dominates the i7-7700 in terms of efficiency and inference latency. The i7-7700 is the least efficient board, but it is also the one with better sequential performance, leading to a smaller post-processing latency variance.

Finally, considering data types, FP16 only negligibly deteriorates the accuracy of the considered networks, but obtaining much better real-time performance, always laying on the Pareto optimality curve. A significant accuracy deterioration is instead experienced with INT8. Therefore, FP16 represents our recommended choice for platforms that support this precision.

The code and the power monitor for the NVIDIA and Xilinx platforms have been made publicly available to stimulate the adoption of fair benchmarking assessments for real-time detection networks.

REFERENCES

[1] M. Bertogna *et al.*, "Adaptive coordination in autonomous driving: motivations and perspectives," in *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, 2017, pp. 15–17.

[2] E. Talpes *et al.*, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[3] L. Li *et al.*, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4665–4673, 2018.

[4] M. Pfeiffer *et al.*, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1527–1533.

[5] S. Jung *et al.*, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.

[6] Z. Zou *et al.*, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[7] J. Chen *et al.*, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[8] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[9] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[10] J. Redmon *et al.*, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[11] F. N. Iandola *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[12] J. Redmon *et al.*, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[13] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[14] X. Zhou *et al.*, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[15] K. Rungsuptaweekoon *et al.*, "Evaluating the power efficiency of deep learning inference on embedded gpu systems," in *2017 2nd International Conference on Information Technology (INCIT)*. IEEE, 2017, pp. 1–5.

[16] C. E. Kim *et al.*, "A comparison of embedded deep learning methods for person detection," *arXiv preprint arXiv:1812.03451*, 2018.

[17] J. Yu *et al.*, "Real-time object detection towards high power efficiency," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 704–708.

[18] H. Nakahara *et al.*, "A lightweight yolov2: A binarized cnn with a parallel support vector regression for an fpga," in *Proceedings of the 2018 ACM/SIGDA International Symposium on field-programmable gate arrays*, 2018, pp. 31–40.

[19] S. Hossain *et al.*, "Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices," *Sensors*, vol. 19, no. 15, p. 3371, 2019.

[20] Z. Lin *et al.*, "Benchmarking deep learning frameworks and investigating fpga deployment for traffic," *IEEE Transactions on Intelligent Vehicles, 4 (3)*, 2019.

[21] C. Ding *et al.*, "Req-yolo: A resource-aware, efficient quantization framework for object detection on fpgas," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 33–42.

[22] V. J. Reddi *et al.*, "Mlperf inference benchmark," *arXiv preprint arXiv:1911.02549*, 2019.

[23] X. Xu *et al.*, "Dac-sdc low power object detection challenge for uav applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[24] Xilinx, Inc., "Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit." [Online]. Available: {https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html}

[25] X. Inc., "Mpsoc dpu trd," https://github.com/Xilinx/Vitis-AI/tree/master/DPU-TRD, 2018.

[26] Xilinx, Inc., "DNNDK Deep Neural Network Development Kit." [Online]. Available: {https://www.xilinx.com/products/design-tools/ai-inference/edge-ai-platform.html\#dnndk}

[27] X. Inc., "Vitis-ai adaptable and real-time ai inference acceleration," https://github.com/Xilinx/Vitis-AI, 2019.

[28] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[29] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[30] P. Meloni *et al.*, "Neuraghe: Exploiting cpu-fpga synergies for efficient and flexible cnn inference acceleration on zynq socs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–24, 2018.

[31] X. Inc., "Chaidnn," https://github.com/Xilinx/CHaiDNN, 2018.

[32] Y. Umuroglu *et al.*, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[33] Microsoft, "ONNX Runtime," https://microsoft.github.io/onnxruntime/.

[34] ONNX, "ONNX," https://github.com/onnx/onnx.

[35] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*.

[36] X. Wu *et al.*, "Recent advances in deep learning for object detection," *Neurocomputing*, 2020.

[37] L. Jiao *et al.*, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128 837–128 868, 2019.

[38] J. Redmon *et al.*, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[39] X. Zhu *et al.*, "Deformable convnets v2: More deformable, better results," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9308–9316.

[40] J. Dai *et al.*, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 764–773.

[41] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[42] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[43] L. Liu *et al.*, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 261–318, 2020.