Performance Evaluation of State-of-the-Art Edge Computing Devices for DNN Inference

Xalo Rancaño[1], Roberto Fernández Molanes[2], Carlos González-Val[2], Juan J. Rodríguez-Andina[1], José Fariña[1]
[1]Dept. of Electronic Technology, University of Vigo, Vigo, Spain. [2]AIMEN Technology Center, Porriño, Spain.
xrancano@alumnos.uvigo.es, {roberto.fernandez, carlos.gonzalez}@aimen.es, {jjrdguez, jfarina}@uvigo.es

***Abstract*—Fast Deep Neural Network (DNN) inference is nowadays possible thanks to the recent significant advancements in computing platforms and DNN frameworks. The advent of the Internet of Things, among other factors, leads to the massive deployment of edge computing devices, whose features and performance are very diverse. In order to help designers identify the hardware platforms and DNNs that best suit their target embedded applications, this paper presents a DNN inference performance analysis for state-of-the-art edge devices. These include Graphical Processing Units, Tensor Processing Units and Field-Programmable Systems-on-Chip. Different versions of the MobileNet and Inception DNNs and different frameworks are considered in the analysis.**

I. INTRODUCTION

During the last decade there have been significant advancements in the field of Deep Neural Networks (DNNs), which have replaced traditional machine learning methods in many areas. Some application domains where DNNs have outperformed other state-of-the-art techniques are computer vision [1], natural-language processing [2], self-driving cars [3], or autonomous robotics [4].

DNNs are becoming increasingly involved in Internet of Things (IoT) technology [5], because they are the best option to efficiently extract information and learn from the massive amounts of data being captured nowadays by IoT devices. In this scenario, these data are sent to local or cloud servers, and used to train DNNs. Once the DNN models are trained they can be used for inference (the actual operation of the DNN in its target application), which is carried out in the cloud or in discrete devices, depending on the application.

The current trend in this area is to move inference as close as possible to sensors, transforming IoT devices from simple sensor hubs or cloud gateways into intelligent devices, usually referred to as edge devices [6]. Current edge devices are able to collect data from one or several sensors and, at the same time, run DNN models in acceptable times to extract useful information from raw data. This information is used locally, in the process being monitored, and commonly sent to a local or cloud server for storage purposes. Therefore, edge devices act like data filters which send only high-level useful data, saving network bandwidth and server storage. For control systems, edge devices also enable the design of more reliable / resilient systems because of their lower response latency and higher robustness to network problems compared with other, more "traditional", IoT devices.

The implementation of DNNs in edge devices is currently a hot research topic. The main challenge is to find the best possible tradeoff among performance, power consumption, reduced size, and the specific requirements of the target application. Currently, all types of computing architectures are present in edge devices. Processors (CPUs) are used for very simple applications [7], which require low inference performance (e.g., below 5 fps in computer vision applications). To achieve higher performance, closer to that of desktop devices, edge devices based on Graphical Processing Units (GPUs), Application-Specific Integrated Circuits (ASICs), or Field Programmable Systems-on-Chip (FPSoCs, which combine processors and FPGA fabric on a single chip) are used.

Regarding GPU-based edge devices, NVIDIA offers Jetson Nano, based on the Maxwell architecture, which contains CUDA cores to run floating-point based DNNs. NVIDIA TX uses the Pascal architecture, which adds 8- and 16-bit instructions to CUDA cores, for efficient inference of quantized models, that is, models adapted for using integer data types instead of floating-point ones. Finally, the newer Xavier series uses the Volta architecture, which includes Tensor cores specifically designed to execute DNNs.

ASIC accelerators have been developed targeting DNN execution with high power efficiency. In this area, Google developed their Tensor Processing Units (TPUs). TPU v1 [8] contains a systolic multiplication array, particularly suited for efficient inference of DNN layers and hardware blocks executing activation functions and pooling operations. TPU v1 is used in the cloud and in edge devices. Recently, TPU v2 and v3, which include training capabilities, have been released, but they are only available in the cloud. Intel offers USB Neural Compute Stick (NCS) 2, based on the Movidius Vision Processing Unit, which can be attached to any computing platform as DNN coprocessor.

Regarding FPSoCs, Xilinx specific frameworks for the Zynq-7000 and Zynq Ultrascale families allow quantized versions of DNN models to be run with low latency and with the possibility of taking advantage of the flexibility provided by the reconfigurable fabric [7].

Some previous works compared the performance of these platforms but, to the best of authors' knowledge, none analyzes all but only some of them. In this paper, the performance of state-of-the-art edge computing devices for DNN inference is analyzed. All the aforementioned architectures, namely GPUs, ASICs, and FPSoCs, are tested when running the most current versions of popular DNN frameworks and the corresponding development tools. The FPSoC implementation is described in more detail, because its workflow is not as well-known as those for the other architectures. To give a wider perspective to the analysis, server-like CPU and GPU have also been tested.

The remainder of the paper is organized as follows. Section II briefly reviews previous works on inference benchmarking in edge devices. Section III describes the experimental setup used to carry out the tests. Experimental results are presented and discussed in Section IV. Finally, Section V summarizes the conclusions of the work.

II. Previous Works

Some previous works evaluated the performance of different computing platforms and frameworks for DNN inference. In [9] different open-source frameworks are tested using the CPU of a Raspberry Pi 3, Model B. These frameworks are Caffe, Caffe 2, TensorFlow v1, and OpenCV running different DNNs, namely Network in Network, GoogLeNet, SqueezeNet, and MobileNet v1. The work in [10] makes a more complete study comparing Desktop Intel CPUs, Intel NCS 2, and Google TPU using the TensorFlow v2, OpenVINO v2, and TensorFlow Lite v1 frameworks, respectively. Similarly, [11] studies a huge array of DNN architectures in a desktop NVIDIA GeForce RTX2080Ti GPU, an embedded NVIDIA AGX Xavier board, and an Intel NCS 2. CUDA 10 and cuDNN 7.5 are used with NVIDIA and OpenVINO with NCS 2. Lastly, [12] carries out a study on NVIDIA RTX Titan GPU and AGX Xavier, but focusing more on the frameworks than in the platforms. In this case Caffe 2, Pytorch v1.2, TensorFlow v1.14, Caffe 2, and Tensor RT are tested.

All these previous examples use CPU, GPU, or ASIC architectures with high-level frameworks like TensorFlow or cuDNN. On the other hand, works related to the FPSoC implementation of DNNs mostly deal with the use of highly optimized DNN IP cores [13]. These works usually compare their results to previous FPGA or desktop implementations. Fortunately, thanks to the availability of specific Xilinx libraries, which enable the FPSoC implementation of most common DNN architectures using TensorFlow and Caffe frameworks, it is now possible to compare FPSoC performance with that of other platforms in a reliable and fair manner, as shown in the following sections. The study is carried out with the most current versions of DNN frameworks and tools, hence also providing more up-to-date information with regard to the previously mentioned works.

III. Experimental Setup

A. Hardware Platforms

The following edge computing devices have been used to carry out the inference performance study:

- Jetson AGX Xavier developer-kit, which includes a 512-core Volta GPU with Tensor cores and an 8-core ARM v8.2 64-bit CPU. Tensor cores provide a precision range from FP32/FP16 to INT8/INT4, which results in a performance leap for deep learning training and inference compared to other GPUs.
- TPU USB Accelerator, including an Edge TPU v1 coprocessor that can be connected to any other processing platform using a 3.0 Type-C connector. It supports TensorFlow Lite models, which target the 8-bit architecture of the TPU.
- TPU Dev Board, which features a fully-integrated System-on-Module (SoM), consisting of an NXP iMX 8M SoC that includes ARM Cortex-A53 MPCore and ARM Cortex-M4F processors, an integrated Vivante GC7000 Lite Graphics GPU, and a TPU v1 coprocessor, which is the DNN accelerator hardware.
- ZedBoard Development Kit, featuring a Xilinx XC7Z020-1CLG484C Zynq-7000 AP SoC that combines a dual-core Cortex-A9 processor with FPGA fabric, where DNN hardware accelerators are implemented. In contrast with other edge devices, FPSoCs can better adapt to specific DNN algorithms thanks to the configurable nature of the FPGA fabric, at the expense of lower operating frequencies, in the 100-150MHz range.

The performance of these edge devices is also compared to that of the following desktop hardware:

- Desktop CPU: a dual Intel Xeon 64-bit E5-2620 v4 processor running at 2.10 GHz. Each processor features 8 cores working with up to 16 threads each.
- Desktop GPU: an NVIDIA GeForce RTX2080 GPU with 2944 CUDA cores and 384 Tensor Cores, with a nominal clock frequency of 1.515 GHz.

B. FPSoC Implementation Details

The acceleration of the DNN layers is achieved in the FPGA fabric through the implementation of a Xilinx

hardware IP block, the Deep Learning Processing Unit (DPU) v3.0. It is a programmable engine optimized for executing Convolutional Neural Networks (CNNs), which includes a scheduler module, a hybrid computing array module acting as the convolutional engine, an instruction fetch unit module, and a global memory pool module. The IP block uses a specialized instruction set, for which the DNNDK v3.1 package provides easy-to-use APIs. DPU variables are internally represented with 8-bit signed integers. Hence, the models to be used have to be quantized to this resolution. For the experiments reported in this paper, the DPU is configured using Vivado Design Suite 2019.2 with the settings listed on Table I.

TABLE I. DPU IP BLOCK CONFIGURATION

Number of Cores	1
DPU Architecture	B1152
RAM Usage	Low
Channel Augmentation	Enabled
DepthWiseConv	Enabled
AveragePool	Enabled
ReLU Type	ReLU + Leacky ReLU + ReLU6
Softmax	Not supported
S-AXI Clock Mode	Independent
dpu_2x Clock Gating	Disabled
DSP48 Usage	High
Ultra-RAM	Not available for XC7Z020
Register Clock	100MHz
Data Controller Clock	90MHz
Computation Clock	180MHz

Number of cores, architecture, DSP48 usage, and RAM usage are configured to take as many resources from the chip as possible in order to maximize throughput. Table II summarizes the FPGA resources used.

TABLE II. FPGA RESOURCE UTILIZATION

Resource	Utilization (units/total)	Utilization (%)
LUT	33413/53200	62.81
FF	63321/106400	59.51
Block RAM	123/140	87.86
DSP slices	212/220	96.36

The DPU IP block is integrated in the FPGA fabric and communicates with the integrated processors through the AXI interface to receive instructions and input data. Direct connections through two High Performance (HP) port masters are used for the DPU to exchange data with the processor, including the network's output results. Both ports, one used to read operations and the other for write ones, can directly access the RAM memory of the processor. The 32-bit instruction port is connected to an AXI HP slave. Lastly, the DPU configuration port is connected to the peripheral's address space using an AXI General Purpose 32-bit port. Fig. 1 shows the connections between the DPU and the processor.

The DNN models tested implement a squeeze and a softmax layer as their last layers. They are programmed in the Zynq-7000 processor, as the DPU doesn't support them.

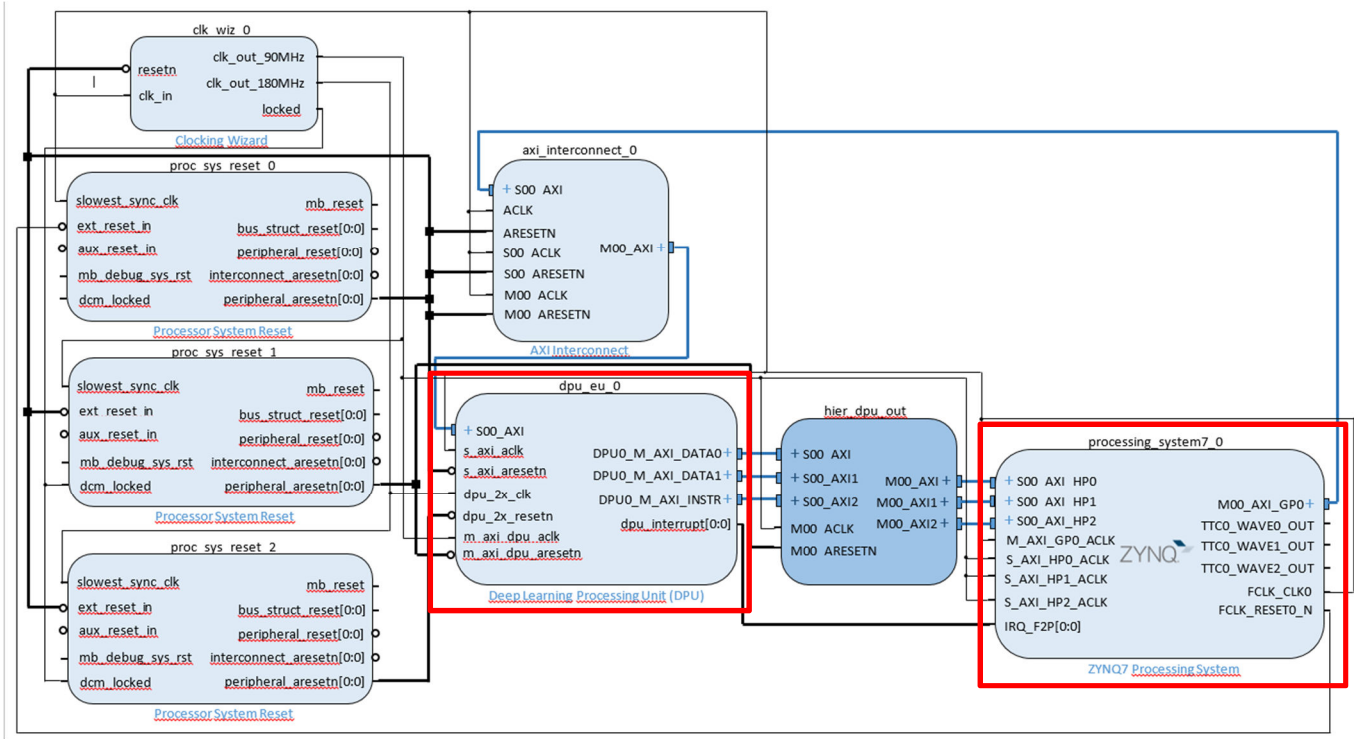

Fig. 1: Connection between the DPU and the Zynq ARM processor.

C. DNN Frameworks

Since the framework used has influence in the achieved performance [12], tests have been carried out using the most similar frameworks possible for the different hardware platforms. TensorFlow has been chosen because it can be used in all platforms. The specific versions of TensorFlow and the related tools used for each platform are:

- Jetson AGX Xavier: TensorFlow 2.0, CUDA 10.0 and cuDNN 7.5.
- TPU: TensorFlow Lite v2.0, which uses precompiled 8-bit quantized versions of the models and a specific API provided by Google/Coral.
- FPSoC: TensorFlow pre-trained models from the Xilinx Model-Zoo repository. It uses Xilinx DNNDK v3.1 quantization and compilation tools to adapt the models to work with 8-bit integers and compile them to the armv7-A processor architecture. The compiler generates the DPU kernel that can be handled by the user from the device's processor application. The PetaLinux 2019.2 operating system (OS) is used.
- Desktop CPU and GPU: TensorFlow Core v2.2.0, CUDA 10.1 and cuDNN 7.6, using TensorFlow pre-trained models from the Keras library.

D. DNNs Used for the Analysis

The DNNs used in the tests are MobileNet v1 [14], MobileNet v2 [15], Inception v1 [16], and Inception v3 [17]. All these networks are image classifiers based on CNNs, trained and validated with the ImageNet dataset. The MobileNet networks are designed for use with resource-restricted hardware like mobile phones. Inception networks are heavier and deeper networks not designed to run in embedded devices, so they are expected to run slower in embedded hardware. These networks have been selected because they are available for all the hardware platforms.

FPSoC models are quantized to 8-bit using the DECENT tool of the DNNDK package [18]. An analysis of the classification accuracy has been performed in order to validate the FPSoC inference when using quantized models, as presented in Table III. The evaluation is done with 2,500 images from ImageNet 2012 for top1 and top5 results. It can be seen that quantized 8-bit Inception models work very well, with negligible accuracy drops. On the other hand, the accuracy of MobileNet models dramatically drops below 12-bit quantization. In order to preserve accuracy, 8-bit quantization of the TensorFlow MobileNet networks requires Xilinx Fine Tunning Quantization, which is already available for Caffe networks in the Vitis-AI v1.1 package but will only be available for TensorFlow models in the next software release Vitis-AI v1.2. However, it has to be noted that this accuracy drop does not invalidate the analysis, because performance results are correct and models with good enough accuracy will be available in next versions of the optimization tools.

TABLE III. EVALUATION OF FROZEN AND QUANTIZED DNN MODELS FOR THE FPSOC DEVICE

Accuracy	MobileNet v1		MobileNet v2		Inception v1		Inception v3	
	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
ImageNet	70.9	89.8	74.9	92.5	68.8	89.6	78.0	93.9
Frozen	73.2	90.2	67.7	89.0	69.9	89.6	77.4	92.1
8-bit quant.	1.44	6.24	4.44	10.1	68.2	88.8	77.8	92.1
12-bit quant.	70.8	89.7	64.6	86.0	-	-	-	-
16-bit quant.	-	-	67.6	89.0	-	-	-	-

E. Measurements

The inference execution time has been measured for all DNNs and hardware platforms previously introduced. Single-image inference is used instead of image batches because this approach is the most commonly needed in edge applications, where latency is more important than throughput.

Measurements reflect the complete execution of the network, including the time needed to copy data back and forth to / from the co-processors. Pre-processing and post-processing times are not included in the measurements. Measurements are repeated 500 times and the mean value is provided as the result. The library used for time measurements is chrono in the case of the FPSoC and timeit (Python) for all other platforms.

To simplify time measurement tests, random images are used as input data, with resolution of 224x224x3 for MobileNet v1, MobileNet v2, and Inception v1, and 299x299x3 for Inception v3. As previously stated, 32-bit floating point data are used for CPU and GPU tests, whereas 8-bit integers are used for TPU and FPGA tests, because of the limitations imposed by their hardware.

IV. RESULTS AND DISCUSSION

Table IV shows performance results for all platforms and DNNs tested. The same results are plotted in Fig. 2.

TABLE IV. INFERENCE EXECUTION TIMES (MS)

Device	MobileNet v1	MobileNet v2	Inception v1	Inception v3
Desktop CPU	59.9	102.7	224.3	188.6
Desktop GPU	5.0	7.2	9.5	14.9
Jetson Xavier GPU	9.1	11.2	12.0	17.8
TPU USB Accel.	3.6	3.8	4.9	46.3
TPU Dev Board	2.4	3.3	4.4	51.3
Zedboard	22.5	31.1	45.3	156.0

Results show significant differences in performance depending on the hardware platform and the network used.

However, interestingly, all tested edge computing devices are faster than the high-end server-level CPU. The inference time for all edge computing devices is, with the exception of Inception v3 in TPU and FPGA, below 50ms. This implies that inference can be carried out at rates higher than 20fps. This is an acceptable value, demonstrating that current state-of-the-art edge devices are appropriate for a wide range of applications.

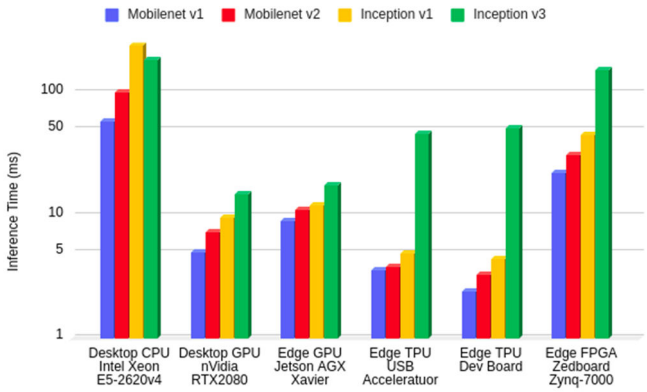

Fig. 2: Inference execution times (ms)

As expected, MobileNet networks are always faster than Inception ones. The reason is that they are specifically designed to target mobile and resource-constrained environments [14][15]. The reduction in resource needs is mainly obtained by implementing depthwise separable convolution layers rather than full convolution ones.

The fastest inference among all edge computing platforms is achieved by the TPU Dev board, except for Inception v3. For MobileNet v1, MobileNet v2, and Inception v1 the TPU Dev board is more than 2 times faster than the desktop GPU, around 3 times faster than the edge GPU, and around 10 times faster than the FPSoC, at smaller footprint. The performance of the TPU USB accelerator is in general only slightly lower than that of the TPU Dev board. The difference is mainly due to the fact that the TPU Dev board integrates the host CPU and the TPU coprocessor in a single SoM, whereas the USB accelerator connects them via an external USB3.0 link. Unexpectedly, TPU performance drops 10 times when moving from Inception v1 to Inception v3. The reason for this is the fixed architecture of the TPU, whose hardware includes specific resources targeting the operations required by a number of networks. When implementing other networks that require some operations which are not natively implemented in the TPU hardware, the corresponding computations are carried out by the host processor, resulting in execution bottlenecks. Therefore, the use of TPUs is recommended for applications that only require a set of operations matching the hardware architecture and that have strong restrictions of size and power. Examples of such applications are intelligent video surveillance cameras and traffic lights [19].

The edge GPU achieves good results compared to its desktop counterpart, which is only between 1.2 and 1.8 times faster. When compared to the other edge platforms, the edge GPU is always faster than the FPSoC and faster than the TPU for Inception v3. It achieves more than 100fps for MobileNet v1, MobileNet v2 and Inception v1 and more than 66fps for Inception v3. It can be concluded that, according to the conducted tests, the GPU provides the best performance-flexibility tradeoff. The use of the GPU is recommended for applications where DNNs are combined with other processing tasks for which GPUs are suitable, such as image and video processing applications, and which do not have power restrictions [20].

Lastly, the FPSoC shows the slowest performance as a DNN edge co-processor, but its results are better than those of a server-grade processor. It performs better for MobileNet networks than for Inception ones, with a large increase in execution time (more than 100ms) from Inception v1 to Inception v3. It is important to note that these results can be improved by just increasing the resources available for the DPU as well as the operating frequency. This is already possible with the high-end Zynq UltraScale family, with which performance can be improved more than 65% [18]. It is also important to note that, although the performance of FPSoCs as DNN co-processors is not as good as that of the other edge platforms tested, the main advantage of FPSoCs comes from their flexibility. For example, if a better DPU core becomes available, the FPGA fabric can be readily updated, increasing the value of the product and reducing the time to market of system upgrade. Moreover, FPSoCs are better suited for time-critical systems, which demand low latency. In case time-critical operations must be executed together with the DNN inference, they can be implemented in the FPGA fabric along with the DPU, saving time to transfer data to / from the processor. Examples of this operations are image preprocessing prior to the inference, or control tasks carried out from the results of the inference engine. FPSoC power consumption lies in between that of TPU and GPU. This makes FPSoCs particularly useful for DNN applications that require intensive preprocessing, for instance to accelerate image preprocessing and inference to match the speed of fast cameras, or in autonomous unmanned aerial vehicle applications [21], where high performance and low power consumption are essential.

V. Conclusions

A performance analysis of state-of-the-art edge hardware platforms for DNN inference has been carried out by implementing in them different types of image classification DNNs using different frameworks. Results show that embedded GPU, TPU, and FPSoC boards can outperform a server-level CPU. The TPU has been shown to be the fastest

platform for lighter networks, achieving between 200 and 400fps. However, its performance significantly decreases (below 20fps) for larger models or those that require operations not natively implemented in its fixed architecture. The edge GPU is better in terms of performance / flexibility ratio, achieving 80-110fps for light networks and close to 60fps for a heavy one. The FPSoC is slower than the other platforms, with performance in the range of 20-40 fps for light networks. However, its potential comes in low-latency applications requiring other computing-intensive tasks to be executed together with the DNN inference.

Future work will extend the tests to other edge platforms, such as NCS or Xilinx Zynq UltraScale+ MPSoC, and frameworks, such as TensorRT. The analysis will include not only execution time, but also power consumption in all platforms. Efforts will also be directed towards the improvement of FPSoC performance, for instance with the addition of optimization techniques benefitting from FPGA architecture, such as network pruning [22].

REFERENCES

[1] G.Campanella *et al.*, "Clinical-grade computational pathology using weakly supervised deep learning on whole slide images," *Nat. Med. 25*, 1301–1309 (2019).

[2] M.Gardner *et al.*, "AllenNLP: A deep semantic natural language processing platform," in *Proc. Workshop NLP Open Source Software, 56th Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 1-6.

[3] Z.Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *Proc. IEEE Intelligent Vehicle Symposium*, 2017, pp. 1856-1860.

[4] J.Mahler *et al.*, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," in *Proc. Robotics: Science and Systems Conference*, 2017.

[5] M.D.Valdes, J.J.Rodriguez-Andina, and M.Manic, "The Internet of Things: The role of reconfigurable platforms," *IEEE Industrial Electronics Magazine*, vol. 11, no. 3, pp. 6-19, Sep. 2017.

[6] W. Shi *et al.*, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, 2016.

[7] R.F.Molanes, K.Amarasinghe, J.J.Rodriguez-Andina and M.Manic, "Deep learning and reconfigurable platforms in the Internet of Things: Challenges and opportunities in algorithms and hardware", *IEEE Industrial Electronics Magazine*, vol. 12, no. 2, pp. 36-49, Jun. 2018.

[8] N.Jouppi, C.Young, N.Patil, and D. Patterson, "Motivation for and evaluation of the first Tensor Processing Unit," *IEEE Micro*, vol. 38, no. 3, pp. 10-19, May/Jun. 2018.

[9] D.Velasco-Montero *et al.*, "Performance analysis of real-time DNN inference on Raspberry Pi," in *Proc. Real-Time Image and Video Processing Conf.*, 2018.

[10] A.Reuther *et al.*, "Survey and benchmarking of machine learning accelerators," preprint arXiv:1908.11348, 2019.

[11] M.Almeida *et al.*, "EmBench: Quantifying performance variations of Deep Neural Networks across modern commodity devices," in *Proc. 3rd Int. Workshop on Deep Learning for Mobile Systems and Applications*. 2019.

[12] B.Ulker *et al.*, "Reviewing inference performance of state-of-the-art deep learning frameworks," in *Proc. 23rd Int. Workshop on Software and Compilers for Embedded Systems*, 2020.

[13] J.Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. 2016 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2016.

[14] A.G.Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," CoRR, abs/1704.04861, 2017.

[15] M.Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks. Mobile networks for classification, detection and segmentation," CoRR, abs/1801.04381, 2018.

[16] C.Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 1-9, 2015.

[17] C.Szegedy *et al.*, "Rethinking the inception architecture for computer vision," preprint arXiv:1512.00567

[18] DNNDK User Guide v1.6 [Online] Available: https://www.xilinx.com/support/documentation/sw_manuals/ai_inference/v1_6/ug1327-dnndk-user-guide.pdf

[19] IEEE Spectrum: Pittsburgh's AI traffic signals will make driving less boring [Online] Available: https://spectrum.ieee.org/cars-that-think/artificial-intelligence/machine-learning/pittsburgh-smart-traffic-signals-will-make-driving-less-boring.

[20] V.Mazzia, A.Khaliq, F.Salvetti, and M. Chiaberge, "Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application," IEEE Access, vol. 8, pp. 9102-9114, 2020.

[21] R.P.Padhy, S.Verma, S.Ahmad, S.K.Choudhury, and P.K. Sa, "Deep Neural Network for autonomous UAV navigation in indoor corridor environments," *Procedia Comput. Sci.*, vol. 133, pp. 643-650.

[22] S.Han, H.Mao, and W.J.Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," preprint arXiv: 1510.00149, 2016.