Artificial Intelligence Research and Development
M. Villaret et al. (Eds.)

doi:10.3233/FAIA210147

On Determining Suitable Embedded Devices for Deep Learning Models

Daniel PADILLA [a,b], Hatem A. RASHWAN [a] and Domènec Savi PUIG [a]
[a] *DEIM, Universitat Rovira i Virgili, Tarragona, Spain*
[b] *Department of Research & Development, Quercus Technologies, Reus, Spain*

Abstract. Deep learning (DL) networks have proven to be crucial in commercial solutions with computer vision challenges due to their abilities to extract high-level abstractions of the image data and their capabilities of being easily adapted to many applications. As a result, DL methodologies had become a de facto standard for computer vision problems yielding many new kinds of research, approaches and applications. Recently, the commercial sector is also driving to use of embedded systems to be able to execute DL models, which has caused an important change on the DL panorama and the embedded systems themselves. Consequently, in this paper, we attempt to study the state of the art of embedded systems, such as GPUs, FPGAs and Mobile SoCs, that are able to use DL techniques, to modernize the stakeholders with the new systems available in the market. Besides, we aim at helping them to determine which of these systems can be beneficial and suitable for their applications in terms of upgradeability, price, deployment and performance.

Keywords. Embedded systems, Deep Learning, FPGA, GPU, DSP, SoC

1. Introduction

Deep learning (DL) has been considered to be one of the most cutting-edge Artificial intelligence (AI) techniques. However, AI companies must overcome a huge problem that they must upgrade their old systems based on traditional ML and computer vision techniques to successful and established products with new features based on DL. Old embedded systems, such as surveillance cameras, are the ones that can make more benefits from DL techniques. And, even though some intelligent cameras are able to process images with lightweight ML algorithms, but these systems have many times low precision for complex problems, e.g., object detection. Thus, upgrading such systems into something more intelligent and reliable that can actually extract information from the input data is the next target for many AI companies.

Not only the high precision achieved by the DL solutions and the low effort to be accomplished comparing to manual engineered (hand-crafted) solutions make them a promising technology. But also, the acquisition and storage of data become more available. The only drawback for using these techniques is the significant increase in the computation complexity that not all commercial applications can afford.

Since DL training is a heavy computational task, the mainstream research device is to use GPUs. Ordinarily, a PCIe card fit for a PC or Server. However, with devices with

limited resources, such as CPU, Memory or even space, and the requirements for high throughput like real-time solutions, DL solutions seem to be a bit far away until now. For example, surveillance cameras can not be expanded with a PCIe card to use GPUs. Not only because they will not have enough space in its case, but also because inserting a PCIe card would mean redesign the whole camera hardware with lots of implications. As a result, some applications are not yet ready to work with DL techniques. However, the technology giants are moving towards more capable tools of making DL integration possible even in a mobile phone.

Several works are focusing on the main hardware options, however some of them are partial and others are not focused enough on the commercial side. For example, several surveys have focused on specific devices, such as Nvidia Jetson Series, with an overview of other devices like FPGAs [16]. Regarding FPGAs, there also are other kinds of surveys that involve several domains tagging and taxonomy [3,18].

In this work, we will first state the main problem related to the performance of DL models on embedded systems. We will then list and overlook some of the alternatives to the current systems. After that, we will state some of the characteristics of every device. Finally, making a decision based on some defined parameters to help the stakeholders to determine which of these embedded systems can be beneficial and suitable for their applications in terms of upgradeability, price, deployment and performance characteristics.

2. Methodology

We will analyse the current status of common embedded devices, such as FBGA, GPUs, CPUs, DSP, etc. Besides, we will define some formulae that can help in adapting an embedded product by adding or replacing some components to substitute the traditional computer vision solution with a DL solution. Thus, to precisely evaluate each embedded device and offer the reader the possibility to adapt the decision to its necessities, and based on a weighting value λ that factor score from 0 to 1, we evaluate each device with four different factors:

Upgradeability (U): For each embedded system to be upgraded, this factor measures the system ability, which means to move from the previous system to one capable of using DL techniques in terms of hardware changes. We use this ranking formula for weighting redesigns on the system:

$$U = (1-\gamma)\lambda_{U0} + \gamma\lambda_{U1},$$

where λ_{U0} stands for a subjective value for the number of changes required for the system design to include a new device and λ_{U1} measures the number of components that have to be changed to include this new device. γ is a weighting value between 0 to 1.

In this work, we set $\gamma = 0.4$, since redesigning the whole system to introduce a new block should penalize more than adapting the printed circuit board (PCB) of that device.

Deployment (D): This factor is related to software difficulties when applying the developed DL model to the target device. On the formula, we equally weight the number of frameworks (TensorFlow, Pytorch, etc.) being used (λ_{D0}) and the time used for different operations, such as compiling, compressing or readjusting the DL model to the deployment mode (λ_{D1}) from the checkpoint mode. As a rule of thumb, we set:

$$\lambda_{D0} = \begin{cases} 1 & \text{if } n = 1 \\ 0.5 & \text{if } n = 2 \\ 0 & \text{if } n > 2 \end{cases}, \qquad \lambda_{D1} = \begin{cases} 1 & \text{No need for operations} \\ 0.5 & \text{Few and quick operations} \\ 0 & \text{Operations takes long time} \end{cases},$$

where n is the number of frameworks being used.

$$D = (1-\alpha)\lambda_{D0} + \alpha\lambda_{D1},$$

where α is a weighting value between 0 to 1.

In this work, we set $\alpha = 0.6$, since we consider the compilation time and modification of the model (compression, etc.) are slightly more important than the number of frameworks used.

Price (P): Since every company has a target for its budget to decrease the final cost of a low-end product. The *P* factor ranks the prices for the embedded device required to use. Assume the target price of the required device is T_p set in this workaround 100 Eur that can be considered the target price of most standard distributed architectures of embedded systems of AI companies. Consequently, we will use the following formula to evaluate the price factor:

$$P = 1 - ((mean(p) - T_p)/max(p)),$$

where p is the list of prices for available devices that can be used in the new target systems. The *P* value will be close to 1, if the $mean(p)$ is close to the target price. Thus, the devices with low prices will have a high value, while low values will be related to expensive devices.

Performance (A): Finally, this factor measures the precision and speed (inference time or frames per second) of the new device. Since various devices will individually be tested, even with the same model (different frameworks, quantization, device's optimal adjustments, etc.), we need to know the variability between the performance on a PCIe GPU and the selected device. For that, we compute the Performance factor based on:

$$A = (1-\theta)\lambda_{R0} + \theta\lambda_{R1},$$

$$\lambda_{R0} = \begin{cases} 1 & \text{if } \lambda_{Inf} <= 1.5 \\ 0.5 & \text{if } 1.5 < \lambda_{Inf} < 2 \\ 0 & \text{if } \lambda_{Inf} >= 2 \end{cases}, \qquad \lambda_{Inf} = \frac{\sum_{i=1}^{j} \log Inf_i}{j}$$

where Inf_i stands for each inference time of Table 2 for that device, and j is the number of available inferences in the table for the same device.

$$\lambda_{R1} = \begin{cases} 1 & \text{if } \lambda_{Prec} <= 1 \\ 0.5 & \text{if } 1 < \lambda_{Prec} < 5 \\ 0 & \text{if } \lambda_{Prec} >= 5 \end{cases}, \qquad \lambda_{Prec} = \frac{\sum_{i=1}^{k} (P_{PCIe_i} - P_i)}{k}$$

where P_{PCIe_i} is the precision value of the DL system on PCIe GPU devices, P_i is the precision values in Table 2 for each device, and k is the number of available precision values in the table for each device.

In this work, we set $\theta = 0.2$. Since performances values are more subjective to compression and model modifications done to fit into low hardware.

3. DL Networks study

Deep Neural Networks (DNN) are known for their large quantity of parameters and operations, such as enormous quantities of trainable parameters. That is translated into a complex structure and, in terms of memory, a big data bulk. In contrast, embedded systems tend to reduce the capacity in terms of memory and computation time. The most common deep convolutional neural networks (DCNN) on low-end embedding systems for object classification are VGG-16 [23] and MobileNet [22] and their variations. These networks are characterized by having low complexity and good enough precision compared to other deep architectures with more precision. In addition, there are three imposing families of Object Detection architectures: YOLO V1 to V5 [21], RCNN [6], and SSD [15,22]. The fastest algorithms amongst them is those based on YOLO.

There is a constant flow of research to make the DNN compact, quicker or simpler to reduce the overall resources needed for a DL inference application. These works depend on two different main lines. The first line is to reduce the number of connections, parameters, and architecture to reduce the model complexity. Examples of such types of LD models are tuning networks to get simpler ones, such as in MobileNet V2 [22]. While the other works go to compress the DNN networks [8]. In turn, the second line focuses on changing the operations insides of DNN networks to get quicker and/or more efficient operations. For instance, some works used Fourier Transforms [13], and binarized models [11] or even they have modified internal network architectures looking for faster operations [5].

In this work, we use the most common DNN networks to provide a fair comparison between several embedded devices. The tested networks are MobileNetv2, VGG-16 and VGG-19 for Object Classification, and YOLO tiny, YOLOv2, YOLOv3 and SSDLite+MobileNetv2 for Object Detection.

4. Hardware

Since the main focus of this paper is to enhance an AI physical product using DL, the ideal solution would be simply changing the Integrated Circuit of the product allowing us to improve the product with DL. That is not often possible since changing an IC usually means changing many more hardware components. Different devices and some specific classifications can allow us to work with DL techniques. We will focus on the most desirable embedded system by AI companies: GPUs, FPGAs and NPUs in turn, ASIC, CPUs and DSPs will be out of scope in this paper.

To be able to compare the different market devices, we have added Table 2 with a compilation of various embedded systems with comparable experiments using trained DL models. We performed three Object Classification tests and 4 Object Detection tests based on well-known DNN network references.

For Object Classification, the most referenced models are MobileNetV2, VGG-16 and VGG-19 that were trained using the ImageNet dataset. The results are given with two factors: the inference time ($Inf.(ms)$) expressed in milliseconds and Top-1 accuracy ($Top1$), as shown in Table 2). For the Object Detection problem, the referenced models are YOLO tiny, YOLOv2, YOLOv3, SSDLite-MobileNetV2 that were trained with the

Device Type		CPU	Nvidia PCIe GPU				Nvidia Jetson Series				FPGA					Mobile SoC					
Model		Intel Xeon E5-2650	Titan X	RTX 2080 Ti	GTX 1080	GTX 1050 Ti	Nano	Tx1	TX2	AGX	Xilinx Virtex 7	Zynq SoC Z-7100	Arria 10 GX	Cyclone V	Stratix V	Kirin 990 5G	Snapdragon 855+	Snapdragon 835	Snapdragon 821	Snapdragon 820	Helio P22
MobileNetV21	Inf.(ms)	6.6	1	0.7	1	-	16	-	-	-	-	-	-	-	-	6	15	181	75	98	243
	Top1	71.9	71.9	71.9	71.9	-	71.9	-	-	-	-	-	-	-	-	69.6	70.5	71.9	72	71.9	71.9
VGG-16	Inf.(ms)	-	-	-	-	-	-	347	-	-	519	-	110	1928	254	-	-	-	-	-	-
	Top1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
VGG-19	Inf.(ms)	-	-	0.6	-	-	100	-	43	7	-	-	-	-	-	42	182	3754	-	4995	7053
	Top1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
YOLO tiny	Inf.(ms)	-	5	-	-	-	-	150	58	-	125	-	15	-	-	-	-	-	-	-	-
	mAP(%)	-	57.1	-	-	-	-	-	32	57.1	-	-	57.1	-	-	-	-	-	-	-	-
YOLOv2	Inf.(ms)	-	15	-	-	-	-	-	172	-	-	80	54	-	-	-	-	-	-	-	-
	mAP(%)	-	76.8	-	-	-	-	-	51	-	-	69.1	76.1	-	-	-	-	-	-	-	-
YOLOv3	Inf.(ms)	133	-	-	31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	mAP(%)	30	-	-	30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SSDLite-	Inf.(ms)	-	-	-	28	16	26	-	-	-	-	-	18	-	-	-	-	-	200	-	-
MobileNetV2	mAP(%)	-	-	-	-	77	-	-	-	-	-	-	73	-	-	-	-	-	-	-	-
References		∗1,2	∗1,3	∗1,4	∗1,2	∗5,2	∗6	∗7	∗8,9	∗10,1	∗7	∗12	∗5,13,14	∗12	∗13	∗1	∗1	∗1	∗15	∗1	∗1

[1] [9] [2] Omni-benchmarking Object Detection. [3] [30] [4] RTX 2080 Ti Deep Learning Benchmarks with TensorFlow. [5] [14]
[6] Jetson Nano: Deep Learning Inference Benchmarks [7] [7] [8] Giant Leaps in Performance and Efficiency for AI Services, from the Data Center to the Network's Edge [9] [10] [10] Jetson AGX Xavier: Deep Learning Inference Benchmarks [11] [29] [12] [17] [13] [28] [14] [30] [15] [22]

Table 2. Comparison between different devices.

COCO dataset. For object detection, the result of inference time is maintained, but the mean Average Precision (*mAP%*) is used instead of Top-1 accuracy. Finally, we add an Estimated Price extracted for some of these devices and in cases of Mobile SoCs, the mobile associated.

4.1. GPUs

The first ideal solution is the one with less effort for the researchers by using GPUs. The solution is a training environment that the company can only transfer the trained DL model to a target device and execute it. The GPUs, such as the NVIDIA Titan series, can be easily integrated with a PC or even a dedicated server with dedicated PCI cards. But including these solutions could be an overshoot, because of the higher prices of these GPUs. Thus we can note this solution is not the preferable solution for the industry.

Model	Estimated Price (Eur)
Intel Xeon E5-2650	875
Titan X	1200
RTX 2080 Ti	1150
GTX 1080	500
GTX 1050 Ti	150
Nano	100
Tx1	300
Tx2 4G	272
TX2	416
TX2i	718
AGX 8G	618
AGX	909
Zynq SoC Z-7020	100
Xilinx Virtex 7	300
Zynq SoC Z-7100	3K
Arria 10 GX	800 - 2K5
Cyclone V	50
Stratix V	6K -16K
Stratix 10 GX	7K - 40K
Kirin 990 5G	800[m]
Snapdragon 855+	420[m]
Snapdragon 835	150[m]
Snapdragon 821	100[m]
Snapdragon 820	75 [m]
Helio P22	100[m]

Table 1.: Prices of different devices. Where [m] stands for the price listed for the mobile using this SoC.

[2] https://towardsdatascience.com/omni-benchmarking-object-detection-b390cc4114cd
[4] https://lambdalabs.com/blog/2080-ti-deep-learning-benchmarks/
[6] https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks
[8] https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/t4-inference-print-update-inference-tech-overview-final.pdf
[10] https://developer.nvidia.com/embedded/jetson-agx-xavier-dl-inference-benchmarks

However, NVIDIA has detected this niche and has promptly developed some smaller units for embedded devices. We are talking about the Jetson Series: Mini-Computer modules with a specific power needed to execute some DL networks. These low-end Jetson GPU modules allow some CUDA capabilities, being very easy to run the same model as your high-end desktop PC's GPU. The computational power and memory of these series limit the complexity of DL models.

With lower prices of the Jetson modules is not weird to see a large number of academic developments focused on Jetson Nano. That is very popular in robotics [24,25,31], also in other applications like adapting YOLO[21] network to fit into this low-end module [29]. Other modules like Jetson TX2 is used in medical image-analysis [19] or in other cases, such as detecting ships [32]. But since the price goes up, as expected, the research tends to lessen, although different benchmarks from unofficial parts [9,1][1][2] and official ones[3][4] seems to indicate their good performance for real-time embedded systems. All in all, Jetson modules seems a good solution for upgrading embedded system.

Table 2 listed different capabilities for computer power evaluation and approximated prices for some of NVIDIA's GPUs and NIVIDIA Jeston series.

4.2. FPGAs

Recently, there are many discussions about another desirable device for applying DL techniques[5], which is related to the Field Programmable Gate Arrays (FPGAs). Since their output results are not instructions-based anymore. Several articles have been presented for improving even further FPGAs. performance [27,12]. However, one of its main important flaws is in the compilation and programming experience. Although nowadays there are possible frameworks for programming FPGAs [20,4], which provide us with a better programming experience for FPGAs, these frameworks are on top of the other DL frameworks (i.e., Tensorflow and Pytorch). Therefore, these FPGA frameworks are not at the same accessibility as CPU/GPU cores yet. Besides, FPGA needs to be reprogrammed for every little change on the DL models, contrary to the GPUs where the change is made on memory and GPU has not to be changed.

The historical weak point of FPGA was the floating-point operations. That is maintained with the help of specialized DSPs blocks embedded in FPGAs to enhance floating-point operations. These blocks had granted a big impact of FPGAs on DL techniques. Thus, in combination with high memory bandwidth and very fast response, FPGAs become a tough competitor for NVIDIA's GPUs. Indeed, large companies like Microsoft and Google move towards these solutions. There are already some interesting applications achieved on cheaper FPGAs, e.g., Zynq 7000, like Facial Expression recognition [33], and Underwater real-time image recognition [33], and other research available on

[1] RTX 2080 Ti Deep Learning Benchmarks with TensorFlow: https://lambdalabs.com/blog/2080-ti-deep-learning-benchmarks/

[2] NVIDIA Jetson AGX Xavier Benchmarks: https://www.phoronix.com/scan.php?page=article&item=nvidia-jetson-xavier&num=4

[3] 2019 Machine Learning Benchmark: https://www.nvidia.com/en-us/data-center/2019-machine-learning-benchmarks/

[4] Jetson AGX Xavier: Deep Learning Inference Benchmarks: https://developer.nvidia.com/embedded/jetson-agx-xavier-dl-inference-benchmarks

[5] Why use an FPGA instead of a CPU or GPU? https://blog.esciencecenter.nl/why-use-an-fpga-instead-of-a-cpu-or-gpu-b234cd4f309c

benchmarks with Intel's Arria 10 [14]. Promising results with FPGAs have been summarised in Table 2.

However, there exist a large gap between low-end and high-end FPGAs price. For instance, high-end FPGAs use SoCs that are used in GPUs. These SoCs tend to have more computational Power per Euro ratio[1]. It is noted that any embedded system with a big budget for upgrading will get the most benefits in terms of performance and efficiency, as shown in Table 3.

4.3. NPUs

The final targets are those low powered and low-cost Mobiles. Many manufacturers had proposed their solutions for mobiles and smartphones. For instance, tensor processing unit, neural network processor, intelligence processing unit, vision processing unit and graph processing unit are some of the names doted by manufacturers known more globally by Neural Processor Unit (NPU). These SoCs have less performance than dedicated GPUs, however, with their tiny modules and cost, they can be more suitable and affordable solutions for real-time applications for embedded systems.

Until recently, mobile applications with such kind of AI applications were server-based. These applications packed and sent the input data (e.g., voice or video) to a dedicated server to make the inference. This could take some time and real-time applications will be more difficult. Nowadays, we have several SoCs with DL features to allow us to jump that barrier of online applications and execute the DL model in the same smartphone [26]. In this case, we can, for example, execute YOLO for object detection in an iOS mobile [2] and the inference is done inside an iOS system and not in a cloud-based server.

There are several new SoCs[9] that gives us a similar performance to old GPUs and even better performance than CPUs. Table 2 summarise some of these values with the MobileNetV2 network. We can observe that inference times can go under 10 msec with the latest SoC at the expense of 2 percentiles in precision, while some of the precision values near the CPU baseline can go up to speeds of 75 milliseconds per inference, which should be enough for some real-time applications. Being able to get prices for SoCs is very difficult.Usually, manufacturers do not give prices for a single SoC, and their values are limited. However, we can compare the cost of the mobiles themselves that come with these SoCs. For example, Table 2 shows some current prices for some mobiles. The prices range is between 20-200 Euros, with a mean of 80 Eur.

5. Discussion

In this paper, we have reviewed the state of the art including the system upgrading from traditional AI systems to promising DL-based systems. For the industry, upgrade an AI embedded system has its cost. However, the high precision, which could achieve with DL, may well deserve it. The community is also going through many different types of

[1]GPU vs FPGA Performance Comparison White Paper 2: https://www.bertendsp.com/pdf/whitepaper/BWP001_GPU_vs_FPGA_Performance_Comparison_v1.0.pdf

		GPU	FPGA	NPU
U	$\lambda_U 0$	0	0.5	1
	$\lambda_U 1$	0.5	0.5	0.75
	Total	0.1	0.5	0.9
D	λ_{D0}	1	2+	2
	λ_{D1}	1	0	0.75
	Total	1	0.1	0.5
P	Total	0.41	0.19	0.95
A	Total	0.5	0.8	0.6
E		2.01	1.59	2.95

Table 3.: Evaluation factors values for each SoCs based on four factors.

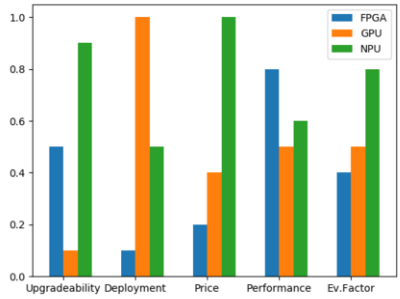

Figure 1.: FPGA, GPUs, NPUs evaluation scores.

research to improve devices to run more complex models, i.e., DL models, and improve the efficiency of the networks. We depend on the four factors mentioned in section 2 (i.e., Upgradeability (U), Deployment (D), Price (P) and Performance (A)) to evaluate the three well-known embedded ystems: GPUs, FPGAs and NPUs. The final evaluation factor (E) can be defined as a weighted sum of the four factors:

$$E = \xi_1 U + \xi_2 D + \xi_3 P + \xi_4 A,$$

where we set $\xi_1 = \xi_2 = \xi_3 = \xi_4 = 1$. As shown Table3, we presented quantitative results of the three embedded systems with the four factors. NPUs provided the best evaluation value of 2.95 in terms of U, D, P and A factors. In turn, with FPGAs, the evaluation value was degraded to 1.59. Although FPGAs provided the best performance value ($A = 0.8$) among the three systems. Similarly,in Figure 1, the results show NPUs should be the best target embedded system for AI companies, followed by GPUs and finally FPGAs supporting our discussion.

The mobile market is a great push for the DL embedding sector. It may not have the best performance, and some kind of trade-off is usually needed when choosing between these devices, but the prices of their ICs, the current investment of the sector in DL and the deployment, may well make these ICs the best ones to be used when upgrading old embedded products.

As things are, this mobile sector will be the ones which, most probably, will quickly grow in the coming years due to the growing demand. Besides, even though other solutions may suit better when they are appropriately chosen, they are much more application-specific. A good example would be where there is no need for an operating system, which the FPGA could fit quite well. Or a large-case product with already I/O pins could be easier to adapt with an NVIDIA Jetson SoM. Mobiles are already capable of running low-end models. And the complexity of these models will keep increasing as the sector moves toward using more AI locally. We can add the low price of devices compared to other ones and, although we would think that low precision and speed may be limited. Thus, we could use several methods to make the trade-off:

- Limit the speed: not all embedded systems must be real-time
- Accept a little loss: The cost could compensate the little loss
- Use the lower cost: The system could upgrade and improve weak points (i.e. using more sensors, extra memory, etc)

References

[1] Mario Almeida, Stefanos Laskaridis, Ilias Leontiadis, Stylianos I Venieris, and Nicholas D Lane. EmBench: Quantifying Performance Variations of Deep Neural Networks across Modern Commodity Devices. 2019.

[2] Maneesh Apte, Simar Mangat, and Priyanka Sekhar. Yolo net on ios. Technical report, 2017.

[3] Ahmed Ghazi Blaiech, Khaled Ben Khalifa, Carlos Valderrama, Marcelo A.C. Fernandes, and Mohamed Hedi Bedoui. A Survey and Taxonomy of FPGA-based Deep Learning Accelerators, sep 2019.

[4] Roberto Di Cecco, Griffin Lacey, Jasmina Vasiljevic, Paul Chow, Graham Taylor, and Shawki Areibi. Caffeinated FPGAs: FPGA framework for convolutional neural networks. In *Proceedings of the 2016 International Conference on Field-Programmable Technology, FPT 2016*, pages 265–268. Institute of Electrical and Electronics Engineers Inc., may 2017.

[5] Wei Ding, Zeyu Huang, Zunkai Huang, Li Tian, Hui Wang, and Songlin Feng. Designing efficient accelerator of depthwise separable convolutional neural network on FPGA. *Journal of Systems Architecture*, 97:278–286, aug 2019.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5). Technical report, 2014.

[7] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Song Yao, Song Han, Yu Wang, and Huazhong Yang. From model to FPGA: Software-hardware co-design for efficient neural network acceleration. In *2016 IEEE Hot Chips 28 Symposium, HCS 2016*. Institute of Electrical and Electronics Engineers Inc., may 2017.

[8] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. oct 2015.

[9] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. AI Benchmark: Running Deep Neural Networks on Android Smartphones. Technical report, 2018.

[10] Duseok Kang, Dong Hyun Kang, Jintaek Kang, Sungjoo Yoo, and Soonhoi Ha. Joint optimization of speed, accuracy, and energy for embedded image recognition systems. In *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE 2018*, volume 2018-January, pages 715–720. Institute of Electrical and Electronics Engineers Inc., apr 2018.

[11] Jaeha Kung, David Zhang, & Gooitzen Van Der Wal, Sek Chai, and Saibal Mukhopadhyay. Efficient Object Detection Using Embedded Binarized Neural Networks. 2018.

[12] Meng Jhe Li, An Hong Li, Yu Jung Huang, and Shao I. Chu. Implementation of deep reinforcement learning. In *ACM International Conference Proceeding Series*, volume Part F1483, pages 232–236. Association for Computing Machinery, 2019.

[13] Sheng Lin, Ning Liu, Mahdi Nazemi, Hongjia Li, Caiwen Ding, Yanzhi Wang, and Massoud Pedram. FFT-based deep learning deployment in embedded systems. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, volume 2018-Janua, pages 1045–1050. IEEE, mar 2018.

[14] Zhongyi Lin, Matthew Yih, Jeffrey M. Ota, John D. Owens, and Pinar Muyan-Ozcelik. Benchmarking Deep Learning Frameworks and Investigating FPGA Deployment for Traffic Sign Classification and Detection. *IEEE Transactions on Intelligent Vehicles*, 4(3):385–395, sep 2019.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9905 LNCS, pages 21–37. Springer Verlag, 2016.

[16] Sparsh Mittal. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform, aug 2019.

[17] Hiroki Nakahara and Tsutomu Sasao. A High-speed Low-power Deep Neural Network on an FPGA based on the Nested RNS: Applied to an Object Detector. In *Proceedings - IEEE International Symposium on Circuits and Systems*, volume 2018-May. Institute of Electrical and Electronics Engineers Inc., apr 2018.

[18] Razvan Nane, Vlad Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Fabrizio Ferrandi, Jason Anderson, and Koen Bertels. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(10):1591–1604, oct 2016.

[19] Bojan Nokovic and Shucai Yao. Image Enhancement by Jetson TX2 Embedded AI Computing Device. pages 1–4. Institute of Electrical and Electronics Engineers (IEEE), jul 2019.

[20] Alexandros Papakonstantinou, Karthik Gururaj, John A. Stratton, Deming Chen, Jason Cong, and Wen-Mei W. Hwu. FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs. In *2009 IEEE*

7th Symposium on Application Specific Processors, pages 35–42. IEEE, jul 2009.

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. jun 2015.

[22] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, jan 2018.

[23] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. sep 2014.

[24] Siddhartha S. Srinivasa, Patrick Lancaster, Johan Michalove, Matt Schmittle, Colin Summers, Matthew Rockett, Joshua R. Smith, Sanjiban Choudhury, Christoforos Mavrogiannis, and Fereshteh Sadeghi. MuSHR: A Low-Cost, Open-Source Robotic Racecar for Education and Research. aug 2019.

[25] Teixeira, Nogueira, Dalmedico, Santos, Arruda, Neves-Jr, Pipa, Ramos, and . Intelligent 3D Perception System for Semantic Description and Dynamic Interaction. *Sensors*, 19(17):3764, aug 2019.

[26] Marian Verhelst and Bert Moons. Embedded Deep Neural Network Processing: Algorithmic and Processor Techniques Bring Deep Learning to IoT and Edge Devices. *IEEE Solid-State Circuits Magazine*, 9(4):55–65, 2017.

[27] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, mar 2017.

[28] Dong Wang, Ke Xu, and Diankun Jiang. PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks. In *2017 International Conference on Field-Programmable Technology, ICFPT 2017*, volume 2018-January, pages 279–282. Institute of Electrical and Electronics Engineers Inc., feb 2018.

[29] Alexander Wong, Mahmoud Famuori, Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, and Jonathan Chung. YOLO Nano: a Highly Compact You Only Look Once Convolutional Neural Network for Object Detection. oct 2019.

[30] Xianchao Xu and Brian Liu. FCLNN: A Flexible Framework for Fast CNN Prototyping on FPGA with OpenCL and Caffe. In *Proceedings - 2018 International Conference on Field-Programmable Technology, FPT 2018*, pages 241–244. Institute of Electrical and Electronics Engineers Inc., dec 2018.

[31] Kailun Yang, Xinxin Hu, Hao Chen, Kaite Xiang, Kaiwei Wang, and Rainer Stiefelhagen. DS-PASS: Detail-Sensitive Panoramic Annular Semantic Segmentation through SwaftNet for Surrounding Sensing. Technical report, 2019.

[32] Hongwei Zhao, Weishan Zhang, Haoyun Sun, and Bing Xue. Embedded deep learning for ship detection and recognition. *Future Internet*, 11(2), 2019.

[33] Minghao Zhao, Chengquan Hu, Fenglin Wei, Kai Wang, Chong Wang, and Yu Jiang. Real-time underwater image recognition with FPGA embedded system for convolutional neural network. *Sensors (Switzerland)*, 19(2), jan 2019.