applied sciences

MDPI

Article

Load Classification: A Case Study for Applying Neural Networks in Hyper-Constrained Embedded Devices

Andrea Agiollo [1,2,*] and Andrea Omicini [1]

[1] Department of Computer Science and Engineering (DISI), Alma Mater Studiorum—Università di Bologna, 47522 Cesena, Italy; andrea.omicini@unibo.it
[2] The Research Hub by Electrolux Professional S.p.A., 33170 Pordenone, Italy
* Correspondence: andrea.agiollo@unibo.it or andrea.agiollo@electroluxprofessional.com

Abstract: The application of Artificial Intelligence to the industrial world and its appliances has recently grown in popularity. Indeed, AI techniques are now becoming the *de-facto* technology for the resolution of complex tasks concerning computer vision, natural language processing and many other areas. In the last years, most of the the research community efforts have focused on increasing the performance of most common AI techniques—e.g., Neural Networks, etc.—at the expenses of their complexity. Indeed, many works in the AI field identify and propose hyper-efficient techniques, targeting high-end devices. However, the application of such AI techniques to devices and appliances which are characterised by limited computational capabilities, remains an open research issue. In the industrial world, this problem heavily targets low-end appliances, which are developed focusing on saving costs and relying on—computationally—constrained components. While some efforts have been made in this area through the proposal of AI-simplification and AI-compression techniques, it is still relevant to study which available AI techniques can be used in modern constrained devices. Therefore, in this paper we propose a load classification task as a case study to analyse which state-of-the-art NN solutions can be embedded successfully into constrained industrial devices. The presented case study is tested on a simple microcontroller, characterised by very poor computational performances—i.e., FLOPS –, to mirror faithfully the design process of low-end appliances. A handful of NN models are tested, showing positive outcomes and possible limitations, and highlighting the complexity of AI embedding.

Keywords: load classification; Neural Networks; embedding; hyper-constrained devices

Citation: Agiollo, A.; Omicini, A. Load Classification: A Case Study for Applying Neural Networks in Hyper-Constrained Embedded Devices. *Appl. Sci.* **2021**, *11*, 11957. https://doi.org/10.3390/app112411957

Academic Editor: Krzysztof Koszela

Received: 15 November 2021
Accepted: 13 December 2021
Published: 15 December 2021

1. Introduction

The emerging trend of products smartness is now becoming a minimum requirement that each commercial appliance should satisfy. Users who once were surprised by simple smart features of commercial products are now expecting devices to perform highly complex tasks such as object detection [1], language processing [2], and many others [3]. Therefore, it is a common trend of industries and manufacturers to introduce processors and controllers in modern commercial products, aiming at satisfying customers requests and transforming once-dumb devices into smart appliances. However, the ever-increasing complexity of the smart tasks to tackle does not cope well with the limited computational capabilities of such commercial appliances. On the one hand, state-of-the-art solutions for complex tasks rely on Artificial Intelligence (AI) approaches, which require high computational powers devices. Indeed, Neural Networks (NNs) and Transformers are the de-facto approaches to tackle such tasks in AI, both of which require GPUs, if not swarm of GPUs and TPUs to train and run. Moreover, most available AI technologies focus solely on the final performance, rather than considering minimization of computational requirements, resulting in requirements of large amounts of energy to run. On the other hand, most commercial products are designed focusing heavily on resource wastes and cost competitiveness. The introduction of costly components like GPUs or powerful processors just to

tackle intelligent tasks would produce an undesirable increase of the final product cost, leading to possible negative outcomes on the market.

It is worth mentioning that there exist possible workarounds to the aforementioned issue. Indeed, most appliances have also become interconnected, enabling the Internet of Things (IoT) paradigm [4], where devices sense an environment, updating a remote user and actuating user decisions. It is possible to leverage such device interconnection to perform smart decisions remotely, removing the computational burden from edge devices. However, this workaround raises concerns in terms of security and privacy, as well as introducing communication bottlenecks and inability for smart devices to work offline. While some solutions for security—e.g., intrusion detection [5], malware analysis [6], etc.—and privacy—e.g., federated learning [7], etc.—can be implemented, it is impossible to fix the offline dumbness of IoT devices as well as their unreliability for real-time or sensible applications. Indeed, devices required to work in real-time scenarios should be designed avoiding IoT solutions, as they are intrinsically bounded by poor connectivity and long-waited responses. Therefore, it would be strongly preferable to design smart devices characterized by local intelligence—i.e., Artificial Intelligence stored directly in edge devices.

Understanding the relevance of enabling small and powerful local AI solutions, emerging research trends focus on the study and proposal of resource-friendly AI techniques: either focusing on the proposal of novel architectures [8,9], or studying NNs deployability in embedded scenarios [10]. However, many works on this field vastly underestimate the resource constraints of devices used in low-end commercial appliances, leveraging rather powerful devices such as Nvidia Jetsons [11,12] or Raspberry Pis [13]. To avoid such fallacies, throughout our work we rely on the selection of hyper-constrained embedded devices, which are common choices for the design of low-end commercial appliances. Such choice distance ourselves from common research approach where device selection is made to satisfy resource requirements of AI techniques, hindering results fidelity. Moreover, while tackling the resource-hungriness issue of common AI solution is a fundamental topic, it is also relevant to analyse what are the existing possibilities to embed AI techniques into constrained devices for commercial appliances. Therefore, in our work we present a relevant case study which aims at analysing what are the available solutions to inject smartness into embedded devices, presenting some successes and undeniable limitations. The relevance of our work is two-folded:

(i) we present a realistic case study for embedding AI techniques into constrained devices;
(ii) we apply such case study over hyper-constrained devices, analysing how available state-of-the-art techniques can be leveraged on such devices.

2. Case Study

In this section we present the proposed case study. First we introduce the characteristics of the hyper-constrained device used in our experiments, comparing it with others commonly-used devices (Section 2.1). Then we present the features of the framework for our case study in Section 2.2.

2.1. Hardware

The design process of low-end commercial appliances is often characterised by a huge focus on costs saving. Indeed, even the introduction of a single hardware component could turn out to be critical to the design of such devices: e.g., adding a costly component to the hardware needed to build the final appliance may substantially increase its consumer price and hinder its chances of commercial success. In order to reproduce as faithfully as possible the cost-saving process, in the following we adopt a hyper-constrained embedding device as the testing device to run AI techniques. More in details, throughout our experiments we leverage an OpenMV Cam H7 Plus (https://openmv.io/products/openmv-cam-h7-plus, accessed on 14 November 2021) microcontroller board. The selected board contains an

ARM Cortex M7 processor which runs at low clock speed (480 MHz) to obtain low power consumptions, enabling deployment on edge and battery-powered devices. Moreover, the selected board stores 32MBs of SDRAM, 1MB of SRAM, and 2 MB of internal flash, for general purpose storage: such a small amount of memory represents the first obstacle against using the simple embedded device for AI applications. Indeed, most of the available state-of-the-art NNs have footprints greater than the memory availability of such device—see Table 1.

Table 1. Footprints of available state-of-the-art NNs. The number of parameters of each model is expressed in millions (M). The memory size is expressed in MBs and is computed as four bytes times the number of parameters, since each parameter is a floating point variable. FLOPs are the number of floating points operations required to run a single instance of a given model and are indicated in billions. The upper half refers to models targeting image classification, while the bottom half refers to models targeting object detection.

Model	#Parameters (M)	Footprint (MB)	#FLOPs (B)
1.0 MobileNet-224 [14]	3.3	13.2	0.28
EfficientNet-B0 [15]	5.3	21.2	0.39
DenseNet-169 [16]	14	56	3.5
Inception-v3 [17]	24	96	5.7
ResNet-50 [18]	26	104	4.1
VGG-16 [19]	138	552	16
SSD300-MobileNet [20]	6.8	27.2	1.2
EfficientDet-D0 [21]	3.9	15.6	2.5
FasterRCNN-MobileNet [22]	6.1	24.4	25.2
SSD300-Deeplab [20]	33.1	132.4	34.9
FasterRCNN-VGG [22]	138.5	554	64.3
YOLOv3 [23]	40.5	122	71

Running AI techniques does not depend on memory footprint only. Indeed, another limitation for running AI on that sort of board comes from the computational power of the device. In order to measure computational power of devices, and compare it with the NNs computational requirements, the FLOPS and FLOPs metrics are commonly used. FLOPS—FLoating point Operations Per Second—refers to the number of floating point operations that a device is capable to complete in one second. FLOPs—FLoating point OPerations–, on the other hand, represents the number of floating points operations required to run a single algorithm—e.g., NN inference. Knowing the number of cores of a device, its clock frequency, and the number of FLOPs per clock cycle that the device is capable to handle, it is possible to compute the theoretical FLOPS performance of a device by means of the following equation:

$$\text{FLOPS} = \text{cores} \times \text{clock frequency} \times \frac{\text{FLOPs}}{\text{cycle}} \tag{1}$$

The ARM Cortex-M7 at our disposal is a single-core microprocessor, which runs at 480 MHz, and capable of one FLOP per cycle. Therefore the board used in our experiments has a theoretical performance of 0.48×10^9 FLOPS. This measure, when compared with the FLOPs requirements of modern NNs, clearly indicates the strong limitations of that sort of devices for AI applications. Indeed, the ARM board is *theoretically* capable of running a single inference of a MobileNet (Table 1) in $t = \frac{\text{FLOPs}}{\text{FLOPS}} = 0.583\,\text{s}$. This would allow incoming inputs to be processed at a rate of 1.72 Frames Per Second (FPS), which is not enough for most real-time commercial solutions. Moreover, our preliminary tests—see Section 3.1—performed using standard pre-trained MobileNet do not even come close to those performances, thus increasing concerns on the applicability of AI techniques on that device. Bigger and more powerful NNs would bear even more burden to the board at hand, thus making it impossible to run close to real-time inference.

The selected device for our tests reflects the design pipeline of low-end commercial appliances. Indeed, the severe constraints of the device differ from most of the embedded solutions that are leveraged by popular works in the field of constrained AI. To show the significant differences between our solution and previous works, we now compare the ARM Cortex M7 processor with other popular solutions. Table 2 clearly points out that the device we selected for our experiments represents the most constrained device. First of all, from a bare computationally perspective, the board is the only one not leveraging any form of GPU assistance for complex computations. Then, given its focus on minimisation of power consumption, the selected board is the only single-core board, and the only device whose processor runs below the 1 GHz threshold. These factors combined make our board the least powerful, having at most one tenth of the computational power (FLOPS) of other boards. For instance, the most-commonly used device for applying AI solutions on embedded devices has a whopping 2083× performance advantage over our board. Furthermore, from the storage perspective, the ARM-based board is 32-times smaller than any other board available on the market which was used in AI-based projects. Overall, given the constrained nature of other devices, it is reasonable to consider the selected board to be a *hyper-constrained* device.

Table 2. Comparison between available embedded devices.

Device Name	RAM	Cores	Work Frequency	#FLOPS	GPU Availability
ARM Cortex M7 *(ours)*	32 MB	1	480 MHz	0.48×10^9	✗
Raspberry Pi-3B+ [24,25]	1 GB	4	1400 MHz	5.3×10^9	Broadcom VideoCore IV
Odroid Xu-4 [26]	2 GB	8	2000 MHz	6.25×10^9	Mali-T628 MP6
Latte Panda [26]	4 GB	4	1440 MHz	7.01×10^9	Intel HD Graphics
Raspberry Pi-4 [13]	8 GB	4	1500 MHz	13.5×10^9	Broadcom VideoCore VI
Nvidia Jetson-Nano [13,27]	4 GB	4	1400 MHz	472×10^9	128-Core NVIDIA Maxwell
Nvidia Jetson-TX1 [8,27,28]	4 GB	4	1700 MHz	1×10^{12}	256-Core NVIDIA Maxwell
Nvidia Jetson-TX2 [13]	8 GB	6	2000 MHz	1.3×10^{12}	256-Core NVIDIA Pascal
Nvidia Jetson-AGX Xavier [11,26,27]	16 GB	8	1900 MHz	11×10^{12}	512-Core NVIDIA Volta

2.2. Load Classification

State-of-the-art NN models are complex and require high computational resources to run, hindering their applicability to embedded devices, and putting at risk the deployment of AI-enabled low-end commercial appliances. However, it should be noticed that those models have been thought and designed so as to tackle complex and general tasks. Indeed, classification models presented in the upper section of Table 1 are trained to classify natural images available in complex datasets like ImageNet [29] or CIFAR-100 [30]. On the other hand, object detection models of Table 1 are trained over Microsoft COCO [31] or Pascal-VOC [32]—which are complex datasets containing the most diverse natural images. That level of complexity is, most of the times, not really required by commercial appliances, and, more often than not, represents an overkill with respect to the actual application needs. In fact, many low-end commercial appliances require the use of AI for simpler tasks, involving few entities—e.g., person detection, face detection, etc.—or simpler research spaces—e.g., smart-home devices, etc. Moreover, deploying general-purpose architectures to tackle more narrow-fielded tasks may hinder the success of the AI solution itself. Indeed, the more NNs are complex—i.e., more parameters –, the more they tend to overfit when presented with simple tasks.

Along this line, we now focus on a simple and representative case study, which aims at faithfully representing the level of smartness actually required by common low-end commercial devices. The case study we introduce deals with image classification, and considers the task of *load classification*. Given a commercial appliance, the simplest and most common task possibly required by the end user is to automatically identify the input provided to the appliance. Automatic recognition could ease many processes required for the proper operation of the appliance at hand, like identification of the process to run, detection of stopping criteria, reduction of energy consumption and wastes, and the like. For that sorts of case study, we only rely on the image classification task, as we consider it enough to satisfy the requirements of most low-end devices. We also limit the problem to the recognition of a fairly-limited amount of entities—i.e., classes of labels–, given that many devices target a fairly-small amount of objects. More specifically, here we consider classifying the load of a common dishwashing machine, identifying four different classes for the given load.

In order to collect the load classification dataset, we produce a prototype of a standard dishwashing machine, installing the same embedded device of Section 2.1 inside its loading cabinet. For each of the four classes of the load classification task we then proceed to take about 1000 images of common loads that belong to such class. It is worth noticing that that number of images represents quite a realistic estimate of the number of images to be collected for a preliminary analysis on the application of AI to commercial appliances: the image collection and labelling process is a costly and time-consuming one, and companies typically aim at reducing its costs to the bare minimum. Since input diversity ease the avoiding of overfitting issues for NNs, we aim at providing as much diversity as possible inside each class images. Images were collected having a 256 $\times$ 256 pixels resolution, following the OpenMV board settings.

The collected dataset represents an intellectual property of Electrolux Professional. Therefore, the collected images can not be made publicly available and no further information can be disclosed regarding the collected dataset. However, the relevance of the presented case study should not be affected by the non-disclosure agreement in place. Up to our knowledge, our work represents the first one combining implementation of AI techniques on a hyper-constrained embedded device with the proposal of a relevant and realistic load-classification task. A comparison between the collected dataset and commonly-used image-classification datasets is presented in Table 3. Thanks to its specificity—i.e., image content—and to the low number of classes involved, the proposed load-classification dataset suits the scenario of AI application to low-end commercial appliances more realistically than the other datasets.

Table 3. Image classification datasets comparison.

Dataset Name	Image Content	# Classes	# Images	Image Resolution	Dimensions
Ours	Dishwasher Loads	4	4K	256 × 256	51 MB
MNIST [33]	Handwritten Digits	10	70K	28 × 28	21 MB
Fashion-MNIST [34]	Clothes	10	70K	28 × 28	36 MB
CIFAR-10 [30]	General	10	60K	32 × 32	162 MB
CIFAR-100 [30]	General	10	60K	32 × 32	161 MB
EMNIST [35]	Handwritten Characters	62	814K	28 × 28	535 MB
Caltech-101 [36]	General	101	9K	variable	132 MB
TF-Flowers [1]	Flowers	5	4K	variable	221 MB
Stanford-Dogs [37]	Dogs	120	20K	variable	778 MB
Food-101 [38]	Food	101	101K	variable	4.65 GB
ImageNet [29]	General	20,000	14M	variable	155.8 GB

[1] http://download.tensorflow.org/example_images/flower_photos.tgz, accessed on 14 November 2021.

3. Experiments and Results

In this section we present our experiments on the case study proposed in Section 2, and the corresponding results. We first analyse possible solutions that leverage state-of-the-art NN architectures as they are (Section 3.1). New models that rely on state-of-the-art architectures are then tested in Section 3.2, while some custom-made models are implemented and compared to other NNs in Section 3.3. Finally, we present a brief study on the potential for generalisation of the trained NNs (Section 3.4).

3.1. State-of-the-Art Models

In order to measure the performance of state-of-the-art NN architectures and show their limitations on hyper-constrained devices, we start our experiments considering well-known NN architectures. More in details, we consider the following highly-popular NNs:

- *MobileNet-V2*—as one of the most light-weight NN architecture available, it was picked as the simplest baseline
- *ResNet-50*—being one of the most representative NN architecture, it was selected to show possible run-time issues
- *VGG-16*—since it represents one of the heaviest NN architecture, it was chosen so as to stress issues with embedded devices

The selected models were trained via standard Stochastic Gradient Descent over the train split of the dataset collected in Section 2.2. Training was completed on a general laptop machine. Computational limitations of the case-study board makes it infeasible to run the training procedure directly on the microcontroller. Trained NNs were then converted to *TensorFlowLite* (https://www.tensorflow.org/lite/microcontrollers, accessed on 14 November 2021) architectures and deployed on the ARM-based board. The conversion process is required, since the board runs the *MicroPython* operating system (http://micropython.org/, accessed on 14 November 2021) which itself supports *TensorFlowLite* architectures only. Inference over the testing dataset was run on the OpenMV microcontroller and performance—both from accuracy and speed perspective—are shown in Table 4.

Table 4. SOTA models performance when deployed on the OpenMV microcontroller.

Model	Memory Footprint	Accuracy	Inference Time
MobileNet-V2	8.6 MB	29.40%	12.35 s
ResNet-50	37.1 MB	30.66%	✗
VGG-16	115.3 MB	28.55%	✗

Interestingly enough, the ResNet-50 and VGG-16 models show the common issues arising when dealing with hyper-constrained devices. Inference for those models cannot terminate on the board due to memory issues. The reason behind the behaviour is to be found on their architectures, which rely on fairly big convolutional filters whose footprint exceeds the 32 MBs available on the microcontroller. Results in Table 4 also shows how state-of-the-art NNs overfit the learning task. This behaviour was expected and is highlighted by the poor performances that those models achieve over the testing set (testing set accuracy for ResNet-50 and VGG-16 was obtained on the training machine). Overfitting issues are due to the complexity of state-of-the-art models and the simplicity of the dataset at hand. Finally, results show that state-of-the-art models are not applicable to embedded hyper-constrained devices as they are characterised by unbearable inference times. Indeed, the only successfully running model is the MobileNet-V2 model which runs a single inference step in 12.35 s. Such measure is far from the theoretical value obtained in Section 2.1 and unsatisfactory for any applicative scenarios.

3.2. SOTA-Based Models

State-of-the-art models present issues mainly linked with their structural complexity. In order to solve them and tackle the case study at hand, we propose a handful of NN architectures that rely on the simplification of state-of-the-art models. We call such NNs *SOTA-based models*—as they rely on the concept of state-of-the-art models, but simplify their structures. More in details, we tackle the structural complexity of SOTA models via truncation of their architectures. SOTA-based models are built stacking N modules of operations that characterise a state-of-the-art NN, and appending a simple classification layer at the end. N is an hyper-parameter of the SOTA-based model and should be ideally kept small to avoid building complex NNs. Throughout our experiments N was selected to match the memory requirements of the OpenMV microcontroller.

As an example, consider the MobileNet-based model built from the MobileNet-V2 architecture. The building block of the MobileNet NN is the Inverted Mobile Bottleneck convolutional layer [14]. Therefore, the MobileNet-based model is obtained stacking together $N = 4$ Inverted Mobile Bottleneck convolutional layers, followed by a classification layer. This represents a trivial and well-known approach for the resolution of NNs complexity issue. However, given the simplicity of most real scenarios where AI is required, such an approach represents a good solution.

We build one SOTA-based model for each SOTA model considered in Section 3.1. Similarly to what was done for SOTA models, SOTA-based NNs were trained outside the microcontroller, converted and then embedded in the OpenMV board for testing. As shown in Table 5 a small ablation study was completed considering image scales. For each SOTA-based model, a new version of the NN was trained considering either $1\times$ or $0.5\times$ scaled images. $1\times$ scaled images correspond to the 256×256 pixels resolution, while the $0.5\times$ scaled images correspond to the 128×128 pixels resolution. This ablation study was undertaken to establish if small loss in image quality could bring inference speed-up, and ease the deployment of AI on hyper-constrained devices.

As Table 5 shows and similarly to what happened for SOTA models, the VGG-based NN applied to 256×256 pixels images could not terminate its inference mechanism due to memory issues. Indeed, the truncation technique we used does not solve memory issues linked with local structures—i.e., layers—of NNs. However, all other SOTA-based NNs run successfully on the microcontroller board, showing improved performances over their Section 3.1 counterparts. The simplification technique applied to NNs successfully solved the overfitting issue. Indeed, classification performances on testing set of such models are now satisfactory. Concerning inference time, such truncation technique brought a relevant speed-up, as it was expected. However, such inference times may not be satisfactory for real-time applications, yet. Finally, the ablation study shows that small reduction in image quality can boosts greatly the speed performance of NNs. Results show the existence of a linear dependency between image resolution and inference time. A $0.5\times$ scaled input image bear a quarter of the pixels to analyse, therefore requiring a quarter of the time to obtain a prediction.

Table 5. SOTA-based models performance when deployed on the OpenMV microcontroller. The subscript of each model represents N, the number of peculiar modules which are stacked to build the SOTA-based model. The $1\times$ or $0.5\times$ represent the rescaling factor of images.

Model	Memory Footprint	Accuracy	Inference Time
$1 \times \text{MobileNet}_4$	0.73 MB	98.72%	5.40 s
$0.5 \times \text{MobileNet}_4$	0.73 MB	98.72%	1.37 s
$1 \times \text{ResNet}_3$	1.70 MB	96.64%	41.30 s
$0.5 \times \text{ResNet}_3$	1.70 MB	95.03%	10.35 s
$1 \times \text{VGG}_2$	1.70 MB	98.15%	✗
$0.5 \times \text{VGG}_2$	1.70 MB	97.80%	47.11 s

3.3. Custom Models

In order to avoid focusing just on popular and ready available NNs, we design and test also few super-simple custom NNs. There, models are designed from scratch using well-known convolutional layers and keeping their complexity to the bare minimum. This approach allows exploring as many solutions as possible, avoiding incurring into wrong conclusion about the deployment of AI techniques on constrained devices. Indeed, we aim at avoiding AI limitations caused by complexity of popular design choices. The designed NN architectures are thought to be as simple as possible, concatenating as few layers as possible, and considering simple convolution techniques only. Such NN architectures are shown in Figure 1, and are presented in details below.

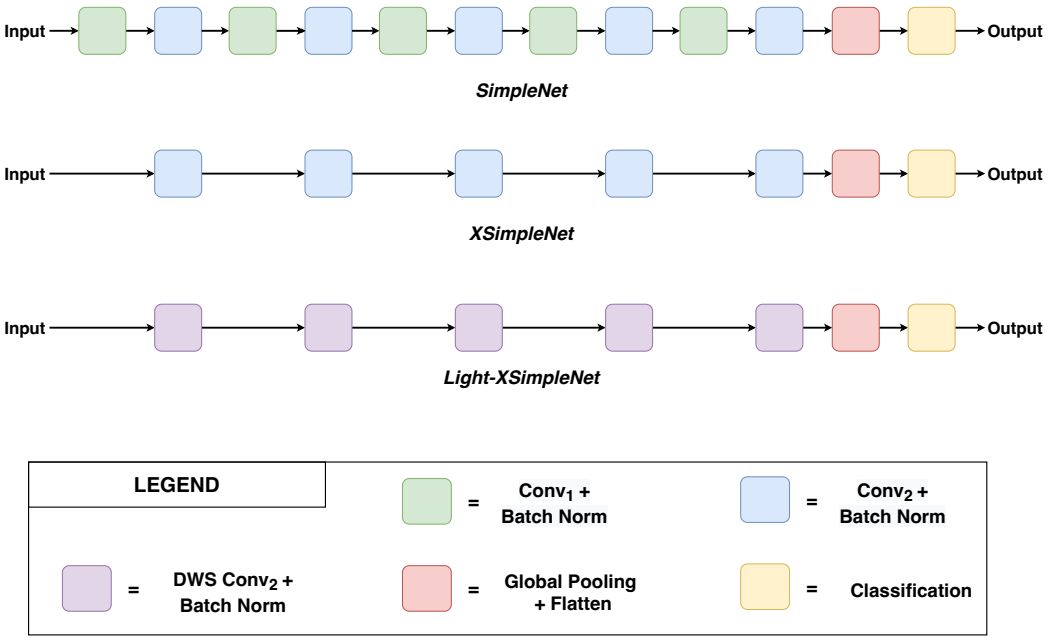

Figure 1. Custom-made NN architectures (better viewed in color).

3.3.1. SimpleNet

We first build a NN relying on the standard 2D-Convolution operation. Aiming at the simplest model possible, we construct its architecture by avoiding skip connections, pooling operations, etc. Therefore, we rely solely on stacking multiple convolutional layers together. More in details, convolutional layers are concatenated, alternating between stride-1 layers and stride-2 layers, up until an 8×8 pixels embedding is obtained. Stride-1 convolutional layers are straightforward convolutions that keep the resolution of the embedding untouched. On the other hand, stride-2 convolutional layers reduce embedding size by a factor of 2. The width of the NN—i.e., number of filters of convolutional layers—duplicates at each stride-2 layer, to avoid information compression issues. Finally, a global average pooling is applied to the 8×8 pixels embedding, and a classification layer is added to obtain the prediction over the input image. SimpleNet architecture is shown in the upper section of Figure 1.

3.3.2. XSimpleNet

SimpleNet complexity is comparable to the MobileNet-based NN. To build an even simpler NN architecture, we here rely solely on the stride-2 convolutional layers, getting rid of the stride-1 convolutions. The NN architecture obtained is an eXtra Simple NN (XSimpleNet), shallower than SimpleNet and relying solely on convolutions. More in details, XSimpleNet is built concatenating stride-2 convolutional layers up until an 16×16 pixels embedding is obtained. Similarly to SimpleNet, the width of the NN duplicates at each stride-2 layer, to avoid information compression issues. The removal of stride-1 convo-

lutions produces an architecture whose width increases at each layer. The aim of that approach is to reduce the NN depth (therefore complexity), while keeping its information handling capabilities more or less untouched. Finally, a global average pooling is applied to the 16 × 16 pixels embedding, and a classification layer is added to obtain the prediction over the input image. XSimpleNet architecture is shown in the middle section of Figure 1.

3.3.3. Light-XSimpleNet

We now aim at building an even smaller and simpler NN architecture than XSimpleNet. To do so, however, it would be risky to decrease the depth—i.e., number of layers—of such architecture. Indeed, XSimpleNet depth is set to the bare minimum needed to manipulate visual information. Therefore, to build a lighter version of the XSimpleNet architecture (Light-XSimpleNet), we here rely on depthwise separable convolution layers [39]. Previously used, standard convolutions perform the channel-wise and spatial-wise computation in one step. Conversely, depthwise separable convolution splits the computation into two steps: a single convolutional filter is applied for each input channel (depthwise), whereas a pointwise convolution is applied to create a linear combination of such output. The splitting allows for a smaller and faster convolutional layer. The Light-XSimpleNet architecture is shown in the bottom section of Figure 1.

Table 6 shows the ablation study and performance of the NNs designed from scratch. Remarkably, the complexity and performance of the SimpleNet architecture is comparable to the ones of the MobileNet-based NN. On the other hand, XSimpleNet and its lighter version outperform all SOTA-based models in terms of inference speed, while retaining a satisfactory accuracy on the test set at hand. Those results represent an encouraging answer to the application of AI into constrained devices issue. Indeed, many simple AI-based scenarios are similar to our case study, and the performance of super simple NN architectures (like XSimpleNet and its lighter version) prove their suitability for solving such tasks.

Table 6. Custom models performance when deployed on the OpenMV microcontroller. The 1× or 0.5× represent the rescaling factor of images.

Model	Memory Footprint	Accuracy	Inference Time
1 × SimpleNet	1.54 MB	97.57%	6.02 s
0.5 × SimpleNet	0.56 MB	95.49%	0.99 s
1 × XSimpleNet	0.38 MB	92.60%	1.84 s
0.5 × XSimpleNet	0.10 MB	92.25%	0.33 s
1 × Light-XSimpleNet	0.06 MB	92.60%	0.33 s
0.5 × Light-XSimpleNet	0.02 MB	89.36%	0.078 s

3.4. Generalisation Testing

While being satisfied by the performance obtained for both SOTA-based models and custom-made NNs, we deepen their analysis, focusing on their generalisation abilities. In particular, the best performing NNs were selected and tested over few novel images of dishwashing loads, which do not belong to the test set. The images are assumed to be different from images available in the dataset, while still representing dishwashing loads. Due to time consumption issues, the number of online images available are far less than the number of images belonging to the test set. However, although being limited in size, such preliminary generalisation test is meaningful as it shows whether trained NNs can be deployed safely in real-world scenarios.

Table 7 shows the generalisation capabilities of NNs, comparing SOTA-based and custom-made architectures. Results highlight how SOTA-based architectures generally bear higher generalisation capacity. This behaviour is understandable, and probably due to the SOTA-based model pretraining. SOTA-based models are built on a backbone—i.e., the SOTA modules—which is pre-trained on ImageNet. Therefore, those models start their learning procedure from a general knowledge, allowing them to better generalise concepts.

Table 7. Best performing models where selected and tested over online images, not belonging to the test set.

	Model	Accuracy	Average Accuracy
SOTA-based	$1\times$ MobileNet$_4$	83%	79.75%
	$0.5\times$ MobileNet$_4$	76%	
	$1\times$ ResNet$_3$	74%	
	$0.5\times$ ResNet$_3$	86%	
Custom	$1\times$ SimpleNet	75%	66.5%
	$1\times$ XSimpleNet	67%	
	$0.5\times$ XSimpleNet	73%	
	$1\times$ Light-XSimpleNet	50%	

4. Discussion

Results presented in Section 3 show the relevance of embedding AI into hyper-constrained devices, highlighting how this is still mostly an open issue. Currently, it is basically impossible to actually make state-of-the-art NNs work on hyper-constrained devices as they are. However, SOTA NNs are thought to be run on powerful machines and tackle complex general tasks. Many, possibly most, tasks to be tackled on embedded devices do not actually need to bear the complexity burden of general tasks. Our result in fact show that it is possible to embed simpler versions of state-of-the-art NNs and custom-built architectures.

The selected NNs show acceptable performance over a simple classification task that mirrors faithfully realistic scenarios. However, this represents only a partial success. Indeed, the performance of the embedded architectures is still far from ideal, as they are characterised by slow speed, hindering AI applicability. Out of the architectures presented, only three models were capable of producing a single prediction in less than 1 s. That value represents a strong limitation to any real-time application, which may require running multiple inferences in shorter time spans. Indeed, such NNs are reliably applicable only on those applications that require classification to be made one-off. Real-time applications requiring AI techniques should therefore consider less constrained devices, as the ones presented in Table 2. Those limitations represent an open issue of AI techniques, and much research effort is needed to tackle them.

Results also show that architectures obtained from state-of-the-art NNs are more stable to small, but relevant, variations of the input images. Those architectures benefit from the pretraining of state-of-the-art models, bearing more general information. When considering real scenarios, this feature is desirable. In fact, real-world inputs may differ from the ones available in the training and testing phase. Therefore, we would suggest to consider NN models derived from state-of-the-art, or at least models showing generalisation abilities, whenever it is possible. However, results point out the existence of a tradeoff between generalisation capability and inference speed. Indeed, SOTA-based model are the most capable of generalisation, while being also the slowest—compared to custom-built models. This stresses once more how the application of AI to embedded devices is still an open research issue. The trade-off between performance and speed is not always acceptable, and a simple-yet-powerful NN architecture capable of running on constrained devices is desirable and worth of research efforts.

5. Conclusions

In this paper we propose a case study on the applicability of NNs in hyper-constrained devices. We identify a relevant image classification task, which mirrors as faithfully as possible many real world scenarios where AI is required. We then consider a set of NN architectures—ranging from well-established heavy and general models, to custom-made small and specific models—and embed such NNs in a microcontroller. The chosen hardware is specifically selected to be as constrained as possible, mirroring the focus on costs saving that characterise the design process of low-end appliances.

The extracted results show that state-of-the-art NNs can not be deployed successfully on such constrained devices. However, SOTA-based and custom-made NNs can be embedded effectively on the microcontroller at hand, showing acceptable accuracy. While being mostly accurate, such models are still characterised by long inference times, which often do not cope well with real-time applications. On the other hand, those NNs suit well scenarios where a one-off prediction is required. The complexity and power-hungriness of NNs represent strong limitations of AI for low-end embedded hardware. Therefore, a research effort is required in the field of NNs simplification in order to enable the deployment of AI-based solutions on hyper-constrained devices.

Author Contributions: Conceptualization, A.A.; Methodology, A.A.; Software, A.A.; Supervision, A.O.; Writing—original draft, A.A.; Writing—review & editing, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially supported by the H2020 Project "StairwAI" (G.A. 101017142) and by Electrolux Professional (Italy).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.W.; Chen, J.; Liu, X.; Pietikäinen, M. Deep Learning for Generic Object Detection: A Survey. *Int. J. Comput. Vis.* **2020**, *128*, 261–318. [CrossRef]
2. Otter, D.W.; Medina, J.R.; Kalita, J.K. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 604–624. [CrossRef] [PubMed]
3. Malik, M.; Malik, M.K.; Mehmood, K.; Makhdoom, I. Automatic Speech Recognition: A Survey. *Multimed. Tools Appl.* **2021**, *80*, 9411–9457. [CrossRef]
4. Al-Fuqaha, A.I.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutorials* **2015**, *17*, 2347–2376. [CrossRef]
5. Agiollo, A.; Conti, M.; Kaliyar, P.; Lin, T.; Pajola, L. DETONAR: Detection of Routing Attacks in RPL-Based IoT. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1178–1190. [CrossRef]
6. Pajouh, H.H.; Dehghantanha, A.; Khayami, R.; Choo, K.R. A Deep Recurrent Neural Network based Approach for Internet of Things Malware Threat Hunting. *Future Gener. Comput. Syst.* **2018**, *85*, 88–96. [CrossRef]
7. Chamikara, M.A.P.; Bertók, P.; Khalil, I.; Liu, D.; Camtepe, S. Privacy Preserving Distributed Machine Learning with Federated Learning. *Comput. Commun.* **2021**, *171*, 112–125. [CrossRef]
8. Fang, W.; Wang, L.; Ren, P. Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments. *IEEE Access* **2020**, *8*, 1935–1944. [CrossRef]
9. Roth, W.; Schindler, G.; Zöhrer, M.; Pfeifenberger, L.; Peharz, R.; Tschiatschek, S.; Fröning, H.; Pernkopf, F.; Ghahramani, Z. Resource-Efficient Neural Networks for Embedded Systems. *arXiv* **2020**, arxiv:2001.03048.
10. Alippi, C.; Disabato, S.; Roveri, M. Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case. In Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN) 2018, Porto, Portugal, 11–13 April 2018; Mottola, L., Gao, J., Zhang, P., Eds.; IEEE: Piscataway, NJ, USA, 2018; pp. 212–223. [CrossRef]
11. Jiang, S.; Lin, Z.; Li, Y.; Shu, Y.; Liu, Y. Flexible High-Resolution Object Detection on Edge Devices with Tunable Latency. In Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom), New Orleans, LA, USA, 25–29 October 2021; Association for Computing Machinery (ACM): New York, NY, USA, 2021; pp. 559–572. [CrossRef]
12. Zhao, R.; Hu, Y.; Dotzel, J.; De Sa, C.; Zhang, Z. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. In Proceedings of the 36th International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 7543–7552. Available online: https://proceedings.mlr.press/v97/zhao19c.html (accessed on 14 November 2021).
13. Süzen, A.A.; Duman, B.; Şen, B. Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry Pi using Deep-CNN. In Proceedings of the 2nd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 26–27 June 2020; pp. 1–5. [CrossRef]
14. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
15. Tan, M.; Le, Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 6105–6114. Available online: https://proceedings.mlr.press/v97/tan19a.html (accessed on 14 November 2021).

16. Huang, G.; Liu, Z.; van der Maaten, L.; Weinberger, K.Q. Densely Connected Convolutional Networks. In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; IEEE Computer Society: Washington, DC, USA, 2017; pp. 2261–2269. [CrossRef]
17. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; IEEE Computer Society: Washington, DC, USA, 2016; pp. 2818–2826. [CrossRef]
18. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; IEEE Computer Society: Washington, DC, USA, 2016; pp. 770–778. [CrossRef]
19. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, 7–9 May 2015.
20. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.E.; Fu, C.; Berg, A.C. SSD: Single Shot MultiBox Detector. In Proceedings of the Computer Vision—ECCV 2016—14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part I; Lecture Notes in Computer Science; Leibe, B., Matas, J., Sebe, N., Welling, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9905, pp. 21–37. [CrossRef]
21. Tan, M.; Pang, R.; Le, Q.V. EfficientDet: Scalable and Efficient Object Detection. In Proceedings of the 33rd IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; Computer Vision Foundation/IEEE: Piscataway, NJ, USA, 2020; pp. 10778–10787. [CrossRef]
22. Ren, S.; He, K.; Girshick, R.B.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Proceedings of the Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 7–12 December 2015; pp. 91–99. Available online: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf (accessed on 14 November 2021).
23. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767.
24. Sajjad, M.; Nasir, M.; Muhammad, K.; Khan, S.; Jan, Z.; Sangaiah, A.K.; Elhoseny, M.; Baik, S.W. Raspberry Pi Assisted Face Recognition Framework for Enhanced Law-Enforcement Services in Smart Cities. *Future Gener. Comput. Syst.* **2020**, *108*, 995–1007. [CrossRef]
25. Curtin, B.H.; Matthews, S.J. Deep Learning for Inexpensive Image Classification of Wildlife on the Raspberry Pi. In Proceedings of the 10th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 10–12 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 82–87. [CrossRef]
26. Hossain, S.; Lee, D. Deep Learning-Based Real-Time Multiple-Object Detection and Tracking from Aerial Imagery via a Flying Robot with GPU-Based Embedded Devices. *Sensors* **2019**, *19*, 3371. [CrossRef] [PubMed]
27. Ullah, S.; Kim, D. Benchmarking Jetson Platform for 3D Point-Cloud and Hyper-Spectral Image Classification. In Proceedings of the IEEE International Conference on Big Data and Smart Computing (BigComp), Busan, South Korea, 19–22 February 2020; Lee, W., Chen, L., Moon, Y., Bourgeois, J., Bennis, M., Li, Y., Ha, Y., Kwon, H., Cuzzocrea, A., Eds.; IEEE: Piscataway, NJ, USA, 2020; pp. 477–482. [CrossRef]
28. Stamoulis, D.; Chin, T.R.; Prakash, A.K.; Fang, H.; Sajja, S.; Bognar, M.; Marculescu, D. Designing Adaptive Neural Networks for Energy-Constrained Image Classification. In Proceedings of the 37th International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 5–8 November 2018; Bahar, I., Ed.; Association for Computing Machinery (ACM): New York, NY, USA, 2018; pp. 1–8. [CrossRef]
29. Deng, J.; Dong, W.; Socher, R.; Li, L.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the 22nd IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Miami, FL, USA, 20–25 June 2009; IEEE Computer Society: Washington, DC, USA, 2009; pp. 248–255. [CrossRef]
30. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Technical Report; Department of Computer Science–University of Toronto: Toronto, ON, Canada 2009. Available online: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (accessed on 14 November 2021).
31. Lin, T.; Maire, M.; Belongie, S.J.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. In Proceedings of the Computer Vision—ECCV 2014—13th European Conference, Zurich, Switzerland, 6–12 September 2014; Proceedings, Part V; Lecture Notes in Computer Science; Fleet, D.J., Pajdla, T., Schiele, B., Tuytelaars, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8693, pp. 740–755._48. [CrossRef]
32. Everingham, M.; Gool, L.V.; Williams, C.K.I.; Winn, J.M.; Zisserman, A. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vis.* **2010**, *88*, 303–338. [CrossRef]
33. LeCun, Y.; Cortes, C.; Burges, C. MNIST Handwritten Digit Database. 2010. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 14 November 2021).
34. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv* **2017**, arXiv:1708.07747.
35. Cohen, G.; Afshar, S.; Tapson, J.; van Schaik, A. EMNIST: Extending MNIST to handwritten letters. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 2921–2926. [CrossRef]

36. Fei-Fei, L.; Fergus, R.; Perona, P. Learning Generative Visual Models From Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. *Comput. Vis. Image Underst.* **2007**, *106*, 59–70. [CrossRef]
37. Khosla, A.; Jayadevaprakash, N.; Yao, B.; Li, F.F. Novel Dataset for Fine-Grained Image Categorization. In Proceedings of the 1st Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Colorado Springs, CO, USA, 25 June 2011; Citeseer: Princeton, NJ, USA, 2011. Available online: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.255.6394 (accessed on 14 November 2021).
38. Bossard, L.; Guillaumin, M.; Gool, L.V. Food-101—Mining Discriminative Components with Random Forests. In Proceedings of the Computer Vision—ECCV 2014—13th European Conference, Zurich, Switzerland, 6–12 September 2014; Proceedings, Part VI; Lecture Notes in Computer Science; Fleet, D.J., Pajdla, T., Schiele, B., Tuytelaars, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8694, pp. 446–461._29. [CrossRef]
39. Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; IEEE Computer Society: Washington, DC, USA, 2017; pp. 1800–1807. [CrossRef]