Deep Learning for Mobile Multimedia: A Survey

KAORU OTA, Muroran Institute of Technology
MINH SON DAO, Universiti Teknologi Brunei
VASILEIOS MEZARIS, ITI-Centre for Research and Technology Hellas
FRANCESCO G. B. DE NATALE, DISI-University of Trento

Deep Learning (DL) has become a crucial technology for multimedia computing. It offers a powerful instrument to automatically produce high-level abstractions of complex multimedia data, which can be exploited in a number of applications, including object detection and recognition, speech-to- text, media retrieval, multimodal data analysis, and so on. The availability of affordable large-scale parallel processing architectures, and the sharing of effective open-source codes implementing the basic learning algorithms, caused a rapid diffusion of DL methodologies, bringing a number of new technologies and applications that outperform, in most cases, traditional machine learning technologies. In recent years, the possibility of implementing DL technologies on mobile devices has attracted significant attention. Thanks to this technology, portable devices may become smart objects capable of learning and acting. The path toward these exciting future scenarios, however, entangles a number of important research challenges. DL architectures and algorithms are hardly adapted to the storage and computation resources of a mobile device. Therefore, there is a need for new generations of mobile processors and chipsets, small footprint learning and inference algorithms, new models of collaborative and distributed processing, and a number of other fundamental building blocks. This survey reports the state of the art in this exciting research area, looking back to the evolution of neural networks, and arriving to the most recent results in terms of methodologies, technologies, and applications for mobile environments.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies** → **Neural networks**; • **Information systems** → *Mobile information processing systems*; *Multimedia information systems;*

Additional Key Words and Phrases: Deep learning, deep neural networks, mobile multimedia computing

ACM Reference Format:
Kaoru Ota, Minh Son Dao, Vasileios Mezaris, and Francesco G. B. De Natale. 2017. Deep learning for mobile multimedia: A survey. ACM Trans. Multimedia Comput. Commun. Appl. 13, 3s, Article 34 (June 2017), 22 pages.
DOI: http://dx.doi.org/10.1145/3092831

1. INTRODUCTION

Deep Neural Networks (DNN), also known as Deep Learning (DL), have recently attracted significant attention from both industry and academia, as they provide powerful instruments to automatically produce high-level abstractions of complex multi-modal data. We are already experiencing the power of DNNs in our daily life, for example, through the on-line recommendation services used by Amazon or Netflix, the voice

Authors' addresses: K. Ota, Muroran Institute of Technology, 27-1, Mizumoto-cho, Muroran, Hokkaido, 050-8585, Japan; email: ota@csse.muroran-it.ac.jp; M. S. Dao, Universiti Teknologi Brunei, Jalan Tungku Link, Mukim Gadong A, BE1410, Bandar Seri Begawan, Brunei; email: minh.son@utb.edu.bn; V. Mezaris, Information Technologies Institute, Centre for Research and Technology Hellas, 6th Km. Charilaou-Thermi Road, P.O. Box 60361, 57001 Thermi-Thessaloniki, Greece; email: bmezaris@iti.gr; F. G. B. De Natale, University of Trento, Via Sommarive, 14-38050 Trento, Italy; email: denatale@ing.unitn.it.

DOI: http://dx.doi.org/10.1145/3092831

and image recognition tools adopted by Google, or the face recognition features in Facebook. DNN technologies are rapidly spreading thanks to the convergence of two key factors: the unprecedented interest in Big Data analytics, and the advances in hardware technologies. The former is creating the need for powerful tools capable of processing enormous volumes of low-structured data and extract significant information. In fact, traditional machine learning techniques are, in most cases, unsuitable for this purpose, because they require a lot of work to define and calculate suitable data descriptions (feature extraction) and are usually unable to generalize and scale to large, unstructured, heterogeneous data. DNNs overcome this issue by allowing computers to easily and automatically extract features from unstructured data without human intervention. The latter is enabling the deployment of DNN technologies at a large scale. At the same time, advances in parallel computing architectures make DNNs more and more feasible. Specifically, in recent years powerful and compact GPUs have been released at affordable prices, which allow accelerating the computation of the weights of DNNs. Such units provide massive parallel processing, speeding-up by several orders of magnitude the execution of both learning and inference algorithms, as compared to conventional CPUs. In addition, the availability of effective open-source libraries and frameworks for implementing basic learning algorithms (see, e.g., Chainer [3], Caffe [53], Torch [10], Theano [14], Tensorflow [12]) made DL methodologies easily available to anyone, causing their rapid diffusion not only within the research community but also for commercial purposes.

Among the many potential application areas, DL brings great opportunities in multimedia computing, where it can greatly enhance the performance of various components, such as object detection and recognition, speech-to-text translation, media information retrieval, multi-modal data analysis, and so on. DL is particularly suited to this domain, as multimedia data are intrinsically associated to severe computational and storage problems. Furthermore, the possibility of introducing smart multimedia applications in mobile environments is gaining more and more attention, due the rapid spreading of smart portable devices. As a consequence, there is an increasing interest on the possibility of applying DNNs to mobile environments [61]. DL not only can boost the performance of mobile multimedia applications available nowadays, but could also pave the way toward more sophisticated uses of mobile devices. Many such devices, including smartphones, smart cameras, pads, and so on, hold some sensing and processing capability that can potentially make them smart objects capable of learning and acting, either stand-alone or interconnected with other intelligent objects. As an example, a remote health-care system may use wearable devices to produce a huge amount of sensor data such as pulse rate, blood pressure, images of face and body, and even audio and video of the patient and the environment. All those data may be used to monitor the patient's condition but require an efficient on-the-fly processing to produce a compact stream of significant information.

Despite the great potential of mobile DNNs, it is not always straightforward to match the requirements of the neural architectures with the constraints imposed by mobile and wireless environments. There are a number of important problems to be solved, starting from the fundamental DNN technologies, to arrive to network architectures, training and inference algorithms, footprint reduction, and so on. For example, as the network connections among mobile devices become unstable and unreliable, existing DNN algorithms need more efficient fault tolerance and security technologies. Meanwhile, unlike traditional high-performance servers, mobile multimedia devices such as wireless sensors and smartphones usually have limited resources in terms of energy, computing power, memory, network bandwidth, and so on. This brings the need of more efficient DNN technologies, which can cope with the constraints of mobile multimedia. Furthermore, because of the dynamic scale and network topologies, mobile multimedia computing needs more scalable DNN support than traditional data center environments.

A possible solution to these problems is to use distributed processing facilities such as cloud computing, where powerful servers can be used to handle heavy DNN processes with plenty of memory space, at the cost of an increased data traffic. Nevertheless, the possibility of directly running DNNs on mobile devices is being investigated as a valuable alternative. This implies either increasing the capabilities of mobile processors and chipsets, or reducing the footprint of algorithms, or both. Additionally, new models of collaborative and distributed processing among mobile devices are considered promising directions to deal with higher complexity tasks. These and other fundamental instruments are expected to provide in the near future reliable, efficient, real-time DNNs for mobile multimedia computing, communications, and applications, but there is a critical need for extensive research from software to hardware and from theory to practice supporting DNN technologies for mobile multimedia.

In this article, we provide a comprehensive survey on state-of-the-art research in DNNs for mobile multimedia, and we highlight some critical issues for future research in this area. The rest of the article is structured as follows. In Section 2, we first present an overview of DNNs and report the software and hardware frameworks so far proposed for efficient DNN implementation. Section 3 focuses on DNNs for mobile environments, and first presents the current results on reduced complexity algorithms for mobile DL, and then we review the latest research on software frameworks and special hardware platforms for mobile DNNs. In Section 4, we introduce a selection of mobile multimedia applications already proposed in different areas, including object detection, human activity monitoring, ambient sensing, cyber-security, and resource optimization. Finally, in Section 5 we close the article with some conclusions and a summary of open research directions.

2. BASIC CONCEPTS ON DL

2.1. A Brief History of DNNs

The history of Artificial Neural Networks (ANN) dates back to the middle 1900s, with the introduction of the first biological-inspired models of shallow ANN [68]. These networks were unable to learn, but this capacity was introduced only a few years later, initially with the introduction of unsupervised learning [46] and subsequently with the first supervised training algorithms [75, 84, 85, 103].

The first wave of popularity of ANNs came a bit later, between 1970 and ’80, with the introduction of efficient back-propagation (BP) learning algorithms [38, 102], and reached its peak in the middle ’80s with the work by Rumelhart et al. [86], which first hypothesized the possibility of obtaining useful representations of input data in the hidden layers of a NN. Since the end of the ’80s, though, the initial enthusiasm within the research community went down when it became clear that BP-trained ANNs could not be the universal solution to any machine learning problem. In the very same years, however, a series of seminal works were published that paved the way toward modern DNNs. In 1987, Ballard proposed the use of unsupervised auto-encoder hierarchies for pre-training of networks [19]. In 1989, LeCun applied the BP to convolutional NN with adaptive connections, one of the key elements of modern deep learners [65]. In its architecture, LeCun organized the network in two types of layers, convolutional and subsampling, each one showing a topographic structure. A couple of years later, Hochreiter was the first to identify the fundamental DL problem, also known as the “long time lag problem” [48], which motivated the following works of Bengio et al. toward a viable solution for deep network learning [21, 49]. More or less in the same years, the Cresceptron model introduced the use of max-pooling (MP) layers in the neural architecture [101], a concept that will be later widely adopted in modern DL.

At that time, however, the attention of the ML community was monopolized by non-neural methods, such as SVMs [96], whose use was widespread in both research and practical applications. It is only at the end of the first decade of 2000 that DL as we currently intend it showed up with the invention of Deep Belief Networks [47], a derivation of Restricted Boltzman Machines, which immediately achieved astonishing performance in MNIST handwritten pattern classification and attracted a new wave of attention on ANNs. In the same years, the use of unsupervised multiple-stage auto-encoders to substitute the pre-training phase before fine-tuning, gained increasing attention [20, 39, 97].

In parallel, the availability of powerful GPUs at affordable prices allowed implementing effective parallel algorithms to speed up the training process. The combination of complex network models, effective algorithms, and ever-increasing computational power paved the way toward the rapid diffusion of DL technologies over the last decade. The first GPU implementations of CNNs allowed speeding-up the learning process by a factor 4–5 over traditional CPU architectures [24], but the joint progress of GPU architectures and optimized algorithms eventually brought gains of some orders of magnitude [82]. The work in Ref. [32] introduced a flexible GPU-implementation of convolutional NN integrating convolutional and max-pooling layers, with a final stage of fully connected layers, which inspired a number of efficient implementations, successfully used to solve various pattern recognition contests. In particular, in 2012, Alex Krizhevsky et al. proposed in Ref. [56] a combination of GPU-MPCNNs that obtained the best results at the ImageNet benchmarking initiative, the so-called AlexNet. The network was made up of five convolutional layers, max-pooling layers, dropout layers, and three fully-connected layers and was designed to classify 1,000 categories. Starting from the idea of AlexNet, in the following years new architectures were proposed with ever increasing performance. In 2013, M. Zeiler and R. Fergus presented the ZF Net [108], which introduced a new way of visualizing the feature maps to gain better knowledge of the inner mechanisms of the network. A further big step ahead in the design of efficient networks was the Inception architecture, which led to the so-called GoogleNet [93]. Inception introduced the intuition that the CNN layers could be organized in new ways, without the need of being layered sequentially. In the same year, Microsoft introduced ResNet, a very deep network with 152 layers that won the 2015 ILSVRC competition with an error rate as low as 3.6% [45]. Among the most interesting results of recent years, it is also worth mentioning the concept of generative adversarial networks, first introduced by Ian Goodfellow et al. in Ref. [43]. In this model, two networks compete in a game where a generator is trying to learn the input data distribution to fool a classifier (the discriminator), while the latter tries not to be fooled. At convergence, the samples created by the generator are indistinguishable from the training ones.

2.2. Software Frameworks

Software frameworks that provide the necessary basis and the building blocks for implementing deep network architectures without coding from scratch are a very important part of the DL ecosystem and a facilitator of the rapid advancements that we witness in DL. A number of such frameworks have appeared in the past five years; these include cuda-convnet [56], caffe [53], mxnet [6], chainer [3], neon [7], torch [10], theano [14], and tensorflow [12].

The most important characteristics of such frameworks are: (i) optimized implementation and exploitation of multi-core computing capabilities (especially GPUs), to allow for fast learning and inference (be reminded that the amount of computations required for training a deep network is one of the two factors that prohibited the practical use of DL for decades!); and (ii) expressibility and extensibility, allowing developers to easily re-use standard network structures such as convolutional, pooling and fully-connected layers, and to extend or modify such structures and the overall network architecture to investigate new ideas, such as the introduction of non-typical nonlinearities.

Cuda-convnet [56] and its more recent version, cuda-convnet2, are the archetypical DL frameworks. They are fast C++/CUDA implementations of convolutional or, more generally, feed-forward neural networks that can be used for modeling arbitrary layer connectivity and network depths; and, they rely on the back-propagation algorithm for training the resulting network. Caffe [53] is a BSD-licensed C++ library with Python and MATLAB bindings for training and deploying convolutional neural networks and other deep models. Caffe treats separately the network representation and its actual implementation, thus allowing its user to easily switch between different hardware architectures (e.g., CPUs and NVidia's CUDA-capable GPUs) for the same network (e.g., training it on a GPU and then performing inference on a CPU-based platform). MXNet [6] is another framework that supports multiple programming languages (C++, Python, R, and more), multiple hardware architectures (CPUs, GPUs, and distributed architectures of multiple CPU/GPUs) and different network architectures, including LSTM-based recurrent networks. Support for the latter (and other network architectures) is a characteristic of most DL frameworks, including Caffe (discussed above) and many of the frameworks discussed in the sequel. Being, by design, extensible, most of these frameworks are continuously extended with implementations of the latest ideas in the field. Chainer [3], for instance, is a Python library that supports feed-forward nets, convnets, recurrent and recursive nets, as well as batch optimization architectures. Neon [7], backed by leading chip manufacturer Intel, is another Python library that also provides a multitude of different types of network layers, activation functions, and cost functions as building blocks for implementing deep networks architectures. Torch [10] is a framework that makes use of the scripting language LuaJIT for simplifying the building of network architectures; similarly to the previous frameworks, it provides a multitude of network building blocks. A beta version of PyTorch [8], which uses Python instead of LuaJIT, has also been released recently. Theano [14] also tries to simplify the modeling of complex mathematical problems, including deep network training. It is a highly extensible compiler for mathematical expressions in Python that translates high-level NumPy-like code into machine language, for efficient CPU and GPU computation. TensorFlow [12], backed by Google, is one of the most recent additions in the DL frameworks ecosystem. Similar to Theano, it is a more general library for numerical computations, in this case using data flow graphs, which can be used not only for realizing DL networks but also in other scientific computing problems. In terms of DL, it includes a set of community-contributed (as for most frameworks) network models and individual building blocks, as well as utilities such as a converter for converting an existing Caffe model to TensorFlow. Add-on interfaces (wrappers) for further reducing the human effort and time of building and training a network architecture using libraries such as the above include Lasagne [5] for Theano, and Keras [4] for Theano and TensorFlow.

Extensions of DL frameworks for distributed DL on big-data clusters have also been developed. Such extensions include CaffeOnSpark [2] and TensorFlowOnSpark [9], based on Caffe and TensorFlow, respectively, which allow performing leaning and inference on Spark and Hadoop clusters.

2.3. Hardware Frameworks

Hardware accelerated technology is an important research area for DL. Special hardware design or architectures will significantly increase the efficiency of DNNs for different applications. In Table I, we list the major hardware acceleration solution categories for DNNs. We will briefly analyze each solution in terms of performance and energy consumption. In the next subsections, we will discuss the use of GPUs,

Table I. Different Hardware Acceleration Solutions for DNNs

Specific hardware or architecture	Solutions	Performance	Energy Consumption	Cost
GPU	CUDA [78], cuDNN [29]	High	High	Low
FPGA	nn-X [41]	Medium	Low	Medium
ASIC	DianNao [25], Eyeriss [28]	High	Low	High
VPU	Myriad 2 [74]	Medium	Low	Medium
Distributed Computing	Adam [30], GeePS [35]	High	High	High
High Performance Computing	COTS [33], DaDianNao [27]	High	High	High

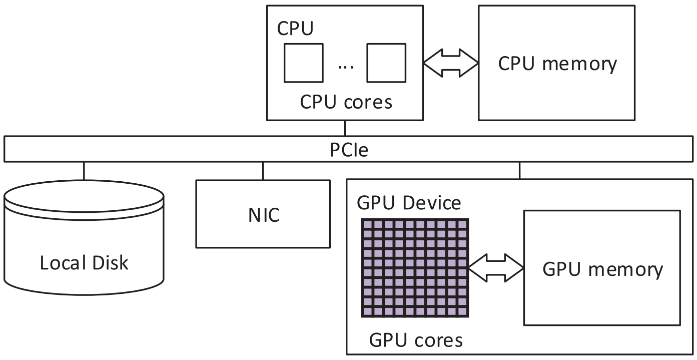

Fig. 1. GPU device in a machine.

distributed computing, and high performance computing. In the following Section 3.3, we will introduce FPGA, ASIC, and VPU-based solutions, which are characterized by a lower energy consumption and are therefore more suited to mobile applications.

2.3.1. GPU. GPUs play an important role in DL, because their highly parallel processing structure is very effective for both learning and inference algorithms. As shown in Figure 1, the typical application requires a host computer with a GPU board installed, where the basic algorithms of the DNN are moved during processing. A GPU device is a single instruction, multiple data (SIMD) device, which has much more cores than a traditional CPU, thus allowing extensive parallelization [87]. Each GPU core includes tens of arithmetic logic units (ALUs), which are the smallest computing units for simple arithmetic or logical computing operations. As each CPU core usually has one ALU, GPUs have better parallel performance than CPUs on specific applications. In DNN, large amounts of neurons will be processed by the same instructions at each layer, which can take full advantage of a SIMD architecture.

There are two major GPU manufacturers, AMD and NVIDIA, which propose computing platforms on their GPUs. NVIDIA also proposed cuDNN [29], which is the primary GPU-accelerated DNN development tool on NVIDIA graphic devices. cuDNN provides spatial convolutions, pooling and activation functions for DL applications. In a cuDNN implementation, NVIDIA focuses on on-chip memory and processing, since off-chip memory is much expensive. The authors of Ref. [29] implement input fetching to hide the memory latency with the data transfer and lazily materializing the lowered matrix on-chip memory only during lower convolutions onto matrix multiplication. Because of the delicate on-chip optimization, experimental results show that cuDNN can perform nearly half of the peak floating point performance (FLOPs) on the NVIDIA graphic devices.

2.3.2. Distributed System. Since the performance of a single GPU is not sufficient to manage large-scale DL applications, it is quite common to parallelize processing tasks

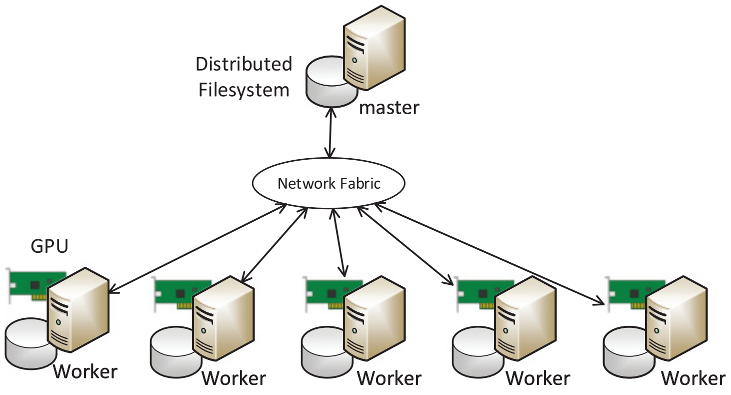

Fig. 2. Distributed DL system.

across multiple GPUs. Distributed computing is an efficient parallel solution to increase the DL performance by exploiting more distributed resources. Figure 2 shows an example of a distributed DL system. A distributed DL system usually consists of many workers and a master server [110]. The master server organizes the task scheduling and node management. A distributed filesystem is deployed for sharing training data, and the master server acts also as a meta-data server for the distributed filesystem, for example, the namenode in HDFS. The master server slices DL training into smaller tasks and distributes such tasks to workers for parallelized processing. Each worker is equipped with GPU devices to ensure adequate performance on processing DL tasks.

Adam [30] is a distributed system, which consists of commodity hardware for training large DNNs. In Adam architecture, the authors configured some servers for transformations from model training servers to improve data delivery throughput. In model training, the authors of Ref. [30] considered different aspects of the distributed system, such as multi-threads, fast weight updates, memory space, slow servers, and communications. They also implement a global parameter server to manage all training servers for throughput optimization, delayed persistence, fault tolerance, and communication isolation. Results show that Adam has good scalability and performance on training DL models.

GeePS [35] is another distributed DL system. It focuses on data movement overheads from training DNNs on distributed GPUs. GeePS introduces a new parameter server into the distributed GPU-based DL system. The parameter server supports parallel machine learning (including DL) applications, running on distributed GPUs. The authors of Ref. [35] design a memory management to organize memory and cache space in each CPU and GPU. In the current implementation, GeePS provides a set of APIs for machine learning applications. Also, it includes the necessary modules for parallel and distributed processing, such as data storage, data movement, and synchronization. Extensive experimental results show that GeePS achieves excellent scalability and performance in training machine learning and DL networks.

2.3.3. High Performance Computing. High-performance computing technology is another approach to organize large amounts of computing devices for parallel processing of various tasks. Coates et al. [33] proposed a high-performance computing (HPC) based DL system, which is able to execute complex training tasks on general HPC clusters. The authors set a cluster with a balanced number of GPUs and CPUs, which is essential for large-scale DL according to their observation. In their implementation, they optimized the GPU kernels for HPC infrastructure and built a communication model across multiple GPUs. From their experimental results, HPC systems may provide effective solutions for DL applications, with good scalability and performance.

Network-on-chip (NoC) is an emerging technology to reduce the latency in communications between different chips in DL applications. NoC is usually used for specific designed multiple-chips solutions. Choi et al. [31] proposed a hybrid NoC architecture that combines CPUs and GPUs on a single chip for DL. Meanwhile, the hybrid network-on-chip architecture also introduces wireless links in CPUs and GPUs communication. The authors designed a customized wireless NoC (WiNoC) to avoid the bandwidth bottlenecks typical of traditional mesh NoCs. They also optimized the network connectivity of the proposed WiNoC and the placement of wireless links. Experimental results show that their NoC architecture achieves a lower full-system energy-delay-product as compared to a mesh or a full wireline application-specific architecture.

Finally, DaDianNao [27] is another HPC-based DNN system, which consists of customized nodes named DianNao [25]. As compared to the classical computing nodes of a HPC architecture, DianNao nodes are specifically designed for DL. The authors introduced multiple chips based nodes into their DL computing cluster. In each node, they adopted eDRAM instead of SRAM to achieve a higher storage density, and they used a fat tree like high-speed internal network to connect all tiles for spreading neurons. Meanwhile, they introduced HyperTransport 2.0 and 2D mesh topology to interconnect the nodes and demonstrated that a carefully designed DaDianNao could outperform GPU devices with much better energy efficiency.

3. DNNS ON MOBILE

The unprecedented potential of DNNs in solving a number of complex machine learning problems has a clear appeal in the framework of mobile devices, where the availability of powerful pattern recognition tools could create great opportunities for a new generation of smart apps. Possible examples include augmented reality, automatic speech recognition and translation, object detection, ego-motion estimation, just to mention a few. The main problems to be solved, however, are concerned with the complexity and memory requirements of the relevant algorithmic solutions, and with the relevant energy consumption. In this section, we will first address the solutions that solely rely on the reduction of the algorithmic complexity of the DNN and then provide further insight on the specific software and hardware solutions for mobile DNNs.

3.1. Reduced Complexity Algorithms for Mobile DNNs

The research in this area is rather intensive, although recent. Since the learning component is typically conceived as an offline operation to be run once for all on a dedicated external hardware, most of the research is focused on the inference component. Nonetheless, several problems remain to be solved, mainly connected to the large number of parameters associated to the network layers and the relevant computation. Classical methodologies applied to reduce the algorithmic complexity involve different kinds of simplifications, including pruning and compression algorithms, as well as more basic approaches based on software optimization.

Pruning is a fundamental concept for the implementation of small footprint DNNs. It concerns the possibility of reducing the number of network connections, by removing the less relevant links. Pruning is a rather established complexity-reduction mechanism, already employed in the optimization of NNs since the ’90s (see Ref. [83] for a early survey on the subject). The first attempts to implement pruning strategies were based on heuristic assumptions, mainly related to the direct proportionality between the magnitude of the networks weights and their saliency. A significant step forward was made by Le Cun et al. in 1990 [36], with the proposal of a new theoretical framework based on the notion of optimal brain damage (OBD). According to the OBD theory, it is always possible to cut one half or more of the weights of a working NN, to obtain a new network that works equally or even better than the original. The selection of the weights to be pruned is not based on their magnitude, but rather on the optimization of a saliency measure, calculated as the second derivative of the objective function with respect to the parameters. Since the direct application of this method will be too complex, the authors proposed a computationally feasible implementation based on an iterative parameter selection strategy. Several works have been proposed in this direction, using different optimal pruning strategies. In Ref. [23], the node selection strategy is recast to an optimization problem and it is iteratively solved using least squares. The authors proposed also a heuristic criterion that allows choosing the units to be removed in a suboptimal way. In the same year, Stepniewski et al. [91] proposed the use of stochastic optimization techniques to solve the same problem, using some methods very popular at that time, such as genetic algorithms and simulated annealing.

More recently, with the increasing use of very deep networks in practical applications, the research on pruning methodologies has attracted new interest. In Ref. [34] the authors proposed a regularization-based approach to induce sparsity during the training of a Convolutional NN. The regularizer forces the learning algorithm to reduce the number of non-zero connections, thus dramatically reducing the memory and runtime requirements of the deployed network. An experimental result on AlexNet demonstrated that it is possible to reduce the memory consumption by a factor of four without significant performance loss. One of the problems of pruning concerns the sparsity of the resulting connections, which implies the need of additional information to represent the locations of preserved connections, and may have negative impacts on the computation. Works have been proposed to solve this problem by imposing some constraints to the locations of non-null parameters. These approaches are known as structured pruning techniques. An interesting example of structured pruning can be found in Ref. [17], where a particle filtering approach is used to decide on the network connections and paths to be removed and a retraining is performed to compensate the performance loss. In a successive work [100], an approach called Structured Sparsity Learning (SSL) is proposed that uses a regularization technique to derive a compact structure from a bigger DNN, while obtaining a hardware-friendly structured sparsity.

Coming to the methods based on compression, the main underlying idea is to shrink the network through typical coding strategies, such as scalar and vector quantization, simplification, hashing and other approaches. In Ref. [89], the authors propose to build compact DL pipelines, suitable for mobile devices due to their lower storage and power constraints. For this purpose, they introduce a unified framework to learn structured parameter matrices, that is, matrices characterized by a number of parameters significantly lower than the matrix size. Although the idea may be generalized, the authors adopt some Toepliz-like matrices, characterized by fast function and gradient evaluation. Results show that the proposed approach provides about $3.5\times$ compression as compared to standard networks, while ensuring nearly equal performance. A different approach is presented by J. Wu et al. in Ref. [104]. Here, to achieve complexity reduction as well as to limit storage requirements, the authors propose to quantize both filter kernels in convolutional layers and weighting matrices in fully connected layers. This technique allows achieving speed-ups in the order of 6, with a compression of about $20\times$ and negligible performance loss. The intuition of Ref. [26] to reduce the memory requirements of very deep networks is to introduce a so-called hashing trick. In their HashedNets, the authors introduce a low-cost hash function whose goal is to cluster connection weights into hash buckets that share the same parameter value. Thanks to the inherent redundancy of the network, this strategy allows saving substantial storage space without loosing in performance. In Ref. [90], a new idea for DNN compression is proposed that takes place during the training phase. To make the parameters more tractable, a regularization term is added to the cost function of the fully connected layers, to force the coefficients binarization. This allows applying a product quantization of the trained weights, as already investigated in Ref. [42], achieving a high compression gain. In Ref. [15], Alvarez and Petersson introduce DecomposeMe, a technique to learn the CNN features with simple 1D convolutions. They also demonstrate that this procedure not only provides a sharp reduction of computational and storage requirements but in some cases can also achieve better classification accuracy. Recently, some interesting proposals have been made to combine different mechanisms of simplification. For instance, Ref. [44] demonstrates the effectiveness of combining pruning with quantization and entropy coding. This method can reduce the storage requirements by a factor 35 on ImageNet, as compared to AlexNet, without loosing accuracy.

3.2. Software Frameworks for the Mobile

Although some of the software frameworks discussed in Section 2.2 do include a specific support for running on mobile platforms (e.g., MXNet [6], TensorFlow [12], and Android-versions of Caffe [1] and Torch [11]), their primary goal is to enable the definition and training of deep network architectures. As such, they are optimized for running on powerful GPUs, where it makes perfect sense to train a deep network, rather than performing inference on resource-constrained mobile devices. For the latter role, more specialized software libraries have been proposed, typically working on networks already trained on one of the aforementioned frameworks, in combination with powerful GPUs or computing clusters.

The DeepLearningKit [16] is a framework for using convolutional neural networks on iOS-running mobile devices that have a GPU (and GPU-powered tvOS devices, as well as conventional OS X computers). It supports performing inference using networks already trained with Caffe [53]. DeepX [59] is a software accelerator for optimizing at runtime the computation, memory and energy requirements of the inference stage of trained deep networks. It works by decomposing the network's computations into simpler blocks, which can be efficiently orchestrated and each of them can be executed on different device processors (e.g., GPUs, CPUs), thus achieving a good utilization of the mobile devices' computing capabilities and being able to control the balance between speed and consumption of the device's resources (i.e., energy). Experimental results show that DeepX allows sharp energy, memory and computation savings, with a very limited impact on the accuracy.

CNNdroid [64] is a GPU-accelerated library for the execution of trained deep CNNs on Android-based mobile devices. It can work with networks trained with Caffe [53], Torch [10], or Theano [14] and can make use of both GPU- and CPU-computing capabilities of the device.

DeepSense [51], a mobile GPU-based deep convolution neural network (CNN) framework, is designed to run CNNs on mobile devices that are equipped with GPUs. DeepSense is able to execute various CNN models for different applications such as image recognition, object detection, and face recognition in soft real time. To do that, DeepSense focuses on understanding the differences between server and mobile GPUs and studying the effectiveness of various optimization strategies. This application can help to develop console applications that run solely on mobile devices without connecting to servers.

Boda-RTC [73] is an open source system that can satisfy the requirements of competitive computational speed and portability towards running CNNs on mobile devices. This system can allow developers to rapidly develop new computational kernels for existing hardware targets and to tune existing computational kernels for new hardware

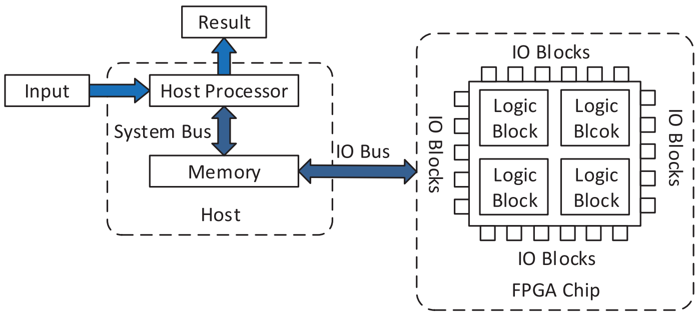

Fig. 3. FPGA-accelerated DL architecture.

targets. The foundation of this system is to use a code-generation approach to target the vendor-neutral OpenCL platforms.

3.3. Hardware for Mobile DNNs

New generations of mobile chipsets may bring high-performance GPUs into mobile devices, thus enabling the execution of DL algorithms in mobile environments. More and more works are appearing that focus on hardware solutions for mobile DNNs, trying to solve fundamental issues such as energy efficiency, integration of systems on a single chip (SoC), large-scale parallelization [98]. In the next paragraphs, we will analyze some of the most interesting solutions so far proposed.

3.3.1. FPGA. Because of the ease of development and affordable cost, Field-Programmable Gate Array (FPGA) is a very useful tool for supporting new hardware applications. FPGA is a reconfigurable chip for the development of particular applications. Users can deploy programs into FPGA to achieve hardware acceleration of specific applications, such as mobile DNNs. The performance of traditional CPU- or GPU-based computing is limited by the von Neumann architecture, in which instructions and data are fetched from an external memory. In DL, this problem becomes even more serious because of the memory bound techniques. FPGAs are not limited by the von Neumann architecture that implements the data and instruction path found in common logic functions. Meanwhile, FPGAs can explore distributed memory on the chip and large degrees of pipelines, which are intrinsically appropriate for the feed-forward nature DL [58].

FPGA manufacturers often provide various development tools for development and deployment. Meanwhile, developers can partially reconfigure FPGAs in a dynamic way to optimize large DL networks. In Figure 3, we show an example of a typical FPGA-accelerated DL architecture. In a FPGA chip, there are mainly three types of units: logic blocks, IO blocks, and programmable interconnects [18]. All units can be programmed for different purposes. Usually, a single FPGA chip cannot support the entire DL procedure. A host processor is used to manage the interaction with general data and applications. To manage the data exchange, the FPGA chip shares a piece of memory with the host processor. The latter pre-processes the input data and stores them back into the memory, from where the FPGA reads them and runs the DL acceleration. Then, the results are written back into the shared memory, from where the host processor can read them and classify the results.

Gokhale et al. [41] proposed a different embedded DL accelerator with a FPGA-based implementation. The authors try to reuse and concatenate data to reduce the limitations of memory access on custom hardware. Their work maximizes node-level parallelization in DNNs with available resources in the accelerator. Meanwhile, the accelerator transforms DNNs into operation codes for executing applications at the hardware level. The authors of Ref. [41] compared the performance of their accelerator with GPU and CPU in performance evaluation, showing that the performance per Watt is much better than that of GPU-based solutions.

3.3.2. ASIC. Specific chips usually perform better than FPGA-based solution, at the price of a higher cost and implementation complexity. As a customized chip for a specific usage, ASIC-based DL acceleration is a potential solution for mobile DNNs. ASIC solutions with dedicated design will typically perform better than FPGA-based solutions. Nevertheless, unlike FPGA solutions, ASIC chips usually need to be taped in the ASIC manufacturer, which brings much higher costs. A further drawback of ASIC with respect to FPGA is that they provide lower elasticity, as they cannot support dynamic partial reconfiguration to meet requirements of different applications.

DianNao [25] is a typical ASIC-based accelerator for machine learning applications. In specific chip design, an important problem is the memory usage. DianNao's design focuses on memory usage through an accelerator architecture and control. The authors also implement the accelerator for both small and large neural networks. In the design of DianNao, energy efficiency is a major issue, and the authors reduce the energy consumption through minimizing memory transfer, which is responsible of most energy consumption in DL processing.

Chen et al. [28] proposed a DL acceleration chip call Eyeriss, which achieves good energy-efficiency thanks to its delicate design and implementation. Eyeriss uses a processing element array with a four-level memory structure, for different steps in convolutional neural networks (CNN). The authors also designed a CNN dataflow for energy efficiency optimization, through reconfiguring the spatial architecture in mapping CNN shapes to computation. NoC is introduced in the chip design for both multicast and point-to-point data delivery, to support optimized data flow. For further improving energy efficiency, Eyeriss adopts run-length compression and processing element data gating to exploit zero-data in CNNs. Experimental results show that Eyeriss has significant potential for accelerating DL applications in a mobile environment.

3.3.3. Mobile Chip. Several manufacturers are currently providing mobile chips for accelerating DL-based applications. NVIDIA proposed the Jetson series, which are Tegra chipsets for general purpose computing [77]. NVIDIA tailored their desktop GPUs to meet the requirements of the mobile environments, and the Tegra chipsets can perform better than mobile processors on DL, as well as other accelerable applications. Since NVIDIA CUDA is the most successful general purpose GPU-based platform for DL, other companies also built NVIDIA chipsets on their mobile products [76]. As the biggest mobile processor vendor, Qualcomm also introduced specific chips into the newest Snapdragon system-on-chip (SoC), to accelerate DL applications [81]. Qualcomm chose a different way to support DL that builds upon Zeroth, a chipset designed for neuromorphic and cognitive computing, in their products. This solution is similar to the high performance meromorphic chip, IBM TrueNorth [52], which is designed for large scale DL clusters, and provides higher performance per Watt than general purpose GPU-based solutions.

Adding DL development tools to existing mobile chips, especially mobile vision processing unit (VPU), is another solution for accelerating DL in mobile devices. Movidius and Google brought a powerful Myriad 2 chip MA2450 with the relevant development tools into mobile devices [13]. The newly developed chip supports existing DNNs such as TensorFlow or Caffe, after translating DNN to Myriad 2 VPU [74]. The performance of the VPU-based solution is better than the on-chip GPUs available in commercial mobile devices.

4. APPLICATIONS

Many emerging mobile application could greatly benefit of artificial intelligence capabilities to locally close the loop between sensing and action. For instance, the ability of using the sensors installed on a portable device to acquire and recognize voice, sounds, images, video, as well as other indirect information (location, orientation, shaking, etc.), would allow extending the power of the device in a number of different ways. Nevertheless, these applications have to cope with the complexity of operating in the wild, with unknown environments, noisy data sources, and unconstrained scenarios, where traditional machine learning tools often provide very limited accuracy. DL has been proven to be one of the most promising approaches to overcome such limits and achieve more robust and reliable inference [60]. However, although DL techniques have been already applied with success in many fields and with different information sources, just a few DL-based mobile applications have been produced untill now, mainly due to the barrier of low-power computational resources offered by mobile devices [62].

Two major models are used to deploy DL applications on the mobile: (1) the client-server model and (2) the client-only model. The former uses the mobile device as a sensor (without data preprocessing), or a smart sensor (with some data preprocessing), whose data are sent to a server or simply the cloud, which runs the DL engine and sends back the results to the client. In this framework, maintaining a good connectivity is the utmost requirement to effectively operate the whole system. The latter model foresees running DL directly on the mobile device. This solution can operate without the need of a network connection and may produce faster results, but has to deal with the lack of resources of the device, thus requiring suitable software and hardware solutions. Both models have been considered in current proposals.

Mobile DL tools proposed so far rely on different capabilities such as object detection, human activity analysis, sound and speech recognition, environmental sensing, and others. These capabilities can be then integrated into different application domains that range from e-health, to security, well-being, resource optimization, games and entertainment, and so on. In the next subsections, we will review some of the most interesting mobile applications proposed so far that utilize DL, addressing some of the above categories.

4.1. DNNs for Health & Wellbeing

Many interesting mobile apps have been recently proposed in the area of health and well-being. In Ref. [57], the user may calculate the amount of calories eaten by taking pictures of the food he eats. The application runs in client-server mode. Whenever the user takes a picture of his/her meal, the system provides an estimate of the relevant amount of calories. The app is based on a DNN that classifies food images. The smartphone is only used to take the picture (sensing) and transferring it to the cloud (transmission), where all image processing steps are performed. Additional data relevant to position and orientation are directly extracted from the embedded phone sensors to improve detection performance. Another app with similar functionalities is *Snap, Eat, RepEat*, described in Ref. [70]. It runs again in client/server mode and informs the user about the amount of calories of a meal, but it provides two operation modes: (i) context-aware, where it uses GPS information and/or restaurant names tagged by users to provide a-priori information on the food, or (ii) in-the-wild, where it uses no additional information to restrict the dictionary. DL is applied to distinguish food and non-food images (GoogleNet and ImageNet) and food item (AlexNet CNN). In Ref. [79], a distributed application classifies food objects in the plate and provides calorie information, using image processing techniques running directly on the smartphone, and DL running on the cloud. DeepFoodCam [95] is a completely client-mode application that runs on iOS and Android devices (both smartphones and tablets). This application implements the inference component of a deep convolutional neural network (DCNN) on the mobile; moreover, the multi-scale network-in-networks (NIN) is integrated into DeepFoodCam to allow users to adjust the trade-off between recognition time and accuracy. Also other sources of information can be used to detect the food consumption. For instance, iHearFood [40] is an application that uses off-the-shelf bluetooth headsets and smartphones to unobtrusively monitor and detect users' eating episodes by analyzing the chewing sound. Four categories of food are defined: liquid (e.g., yogurt or soup), soft (e.g., pizza or fries), hard (e.g., steak or burger), and crispy (e.g., apple and chips). A Deep Boltzmann Machine–Deep Neural Network (DBM-DNN) model is used for recognition, due to its known performance in sound recognition. This application requires internet connectivity to manage the connection with the server.

Human activity monitoring is another interesting area in health-related applications. Several apps have been proposed for detecting the users' behavior, making use of heterogeneous sensors. For instance, Ref. [111] is an application that predicts personal daily activity budget using a wearable mobile sensor and DL. The energy expenditure is known as an important step in tracking personal activity and preventing chronic diseases. The application uses two different sensors to gather accelerometer data (mobile phone) and pulsation (heart-rate monitor). A multi-channel CNN operating in client-server mode is then utilized to learn and predict user' behaviors. The authors claim that DL can bring higher accuracy compared to other non-neural machine learning methods. DeepEar [63] is a mobile audio sensing framework that analyzes the acquired audio stream with a DNN, to infer a broad set of human behaviors and contexts. It is implemented to run on smartphones with effective energy consumption (only 6% of the smartphone's battery daily), using a cloud-free DSP-based implementation. The app provides an estimate of the level of stress and emotions and identifies speakers.

On a more specific medical domain, PhoneGap [55] is a smartphone e-health application designed to assist physicians in identifying thyroid lesions and diseases through images taken by a smartphone camera. PhoneGap analyzes images to detect thyroid cytopathology using different machine learning methods including convolutional neural networks (CNN). The results are linked to the semantic web to provide an explanation of the disease to the users. Reference [54] is an app that determines the heart rate of an individual during intensive motion (e.g., sports), by applying DL on smartphone accelerometer and PPG (photoplethysmography) signals recorded by a smartphone or a wearable device. The cloud connectivity of the device is requested to collect and classify selected PPG signals in real-time. These signals are then used to train a deep belief network with Restricted Boltzmann Machines (RBM) to estimate heart rate.

4.2. DNNs for Security

There are different areas of security where mobile applications can benefit from DL technologies. An area that received great attention is the detection of malware on mobile devices. DrodSec [106] tries to detect android malware by using deep belief networks with RBM. This application builds upon more than 200 features extracted from both static and dynamic analysis of Android apps, to create a malware-detection model capable of classifying malware in smartphones. DroidDetector [107] is also an application running on smartphones that can automatically detect if an app contains a malware. The core of this application is its online DL-based Android malware detection engine. Features that fall in required permissions, sensitive APIs, and dynamic behaviors categories are extracted to feed a Deep Belief Network (DBN) for detecting malware. DeepSign [37] is designed to accurately classify new malware variants using DBNs implemented with a deep stack of denoising autoencoders. This application is claimed to overcome the disadvantage of conventional signature- and token-based methods, which fail in detecting most of the new variants of malware. One of the advantages of this application is that it can generate a signature of a malware by training a related DNN with any raw input from a sandbox. Another advantage is that DeepSign is completely agnostic to the type of malware behavior that is logged. Deep4MalDroid [50] is an Android malware detection system using a Linux-kernel-system-call-graph-based DL framework. First, the code routines of each given Android application can be automatically executed by using the *component traversal*, a dynamic analysis method developed for Deep4MalDroid, instead of relying on user interactions or a random event generator. Then, the weighted directed graphs of system calls are constructed by using a list of Linux kernel system calls extracted by using the *component traversal*. Finally, Android malware is detected by using a DL architecture with Stacked AutoEncoders (SAEs) model.

DL can also be used for biometry. As an example, Ref. [109] describes an application to recognize the iris using mobile devices. Three image pre-processing algorithms are performed on the mobile to extract optimized ordinal measures (OM) features that are then encoded for local iris texture. Next, a CNN is run to learn pairwise features that will be used to measure the correlation between two irises. Finally, the score level is reported by fusing the selected OMs and the learned pairwise features. The application works in client-server mode.

Concerning the application of DNNs for security, it has to be mentioned that, despite their impressive results, recent research has shown that deep networks can be easily fooled by adversarial examples, namely, images/patterns that are slightly obfuscated to mislead classification by the addition of a barely-perceivable adversarial noise. This fact could have a large impact on the use of DNNs for security applications and life-critical tasks (e.g., self-driving cars). Some interesting works in this direction are starting to appear in the scientific literature [22, 72, 88, 94].

4.3. DNNs for Ambient Intelligence

Other interesting applications of DNNs can be found in the framework of smart environments. A nice example is DeepCamera [80], a smartphone application that recognizes the places-of-interest (PoIs) by using a deep convolution network, which is further compressed into a shallow net to fulfill real-time constraints. Both spatial and visual features are extracted and used to recognize the PoIs. SpotGarbage [71] uses the smartphone camera to automatically detect and localize garbage in the real world, using a deep architecture of fully connected CNNs, called GarNet. The optimization of the network leads to a reduction of 87.9% in memory usage, to allow direct processing on the smartphone. Both client-server and client-only modes are possible. GarNet is trained with GINI (Garbage IN Images) dataset, constructed by crowd annotations. SpotGarbage can also get feedback from users to further update the GarNet model.

It is possible to use mobile sensing also to detect anomalous events in the surrounding environment. For instance, MobiEar [67] is designed to assist deaf people, alarming them in the presence of dangerous situations with the help of visual signs (e.g., a flashing light) or vibrations. To accurately recognize acoustic events from various environments with diverse ambient noise patterns, MobiEar uses a CNN to extract the generic environment-free acoustic features, which are then input to an elastic event classification model. These two models allow the user to subscribe the events of interest, assigning them to unique models. The models are trained on the cloud and installed on smartphones. These models are also updated by using new information fed by the users' interaction processes.

Another interesting domain is user interaction. In Ref. [99], the authors develop a client-server mode application that utilizes Google's project Soli to construct a sensor for micro-interactions in mobile and wearable computing. The app is able to detect gestures in a fast, accurate and unobtrusive mode. To do that, an end-to-end trained combination of CNNs and recurrent neural networks (RCN) is built, receiving as input high temporal resolution data. High-frequency and short range radar input data, collected from Soli's sensors that records *sliding index finger over thumb* gestures, are pre-processed and sent to the NN, whose output will be fed to RCN to predict gestures. Reference [66] prevents users to be distracted by unimportant notifications being sent to a smartwatch from a smartphone. The app is trained to decide whether a notification to be relayed to the smartwatch is important or not, based on a DL-based binary classifier. This network receives as input a set of features extracted from the sensor data. The model is trained on a server, but runs in client-only mode.

4.4. DNNs for Translation and Speech Recognition

Google Translate is a well-known service that can run on mobile devices both in online and offline modes. This service uses *Phrase-Based Machine Translation* (PBMT) as the key algorithm. Later, *Neural Machine Translation* (NMT), based on DNN, was introduced to reduce the engineering complexity of previous phrase-based statistical translation systems [92]. Both Google and Microsoft are using from late 2016 similar NMT-based translation engines in their applications. The Google implementation [105], namely *Google's Neural Machine Translation System*(GNMT), is based on the principle of example-based machine translation and was trained with millions of examples. Google also introduces a personalized speech recognition system running on Nexus 5 Android smartphones [69]. This system utilizes quantized DNNs to create a system with fewer requirements of memory and computational footprint but acceptable accuracy to run on low-resource mobile devices. Microsoft developed a variant of NMT, applying it to all its speech translation services, including Microsoft Translator and Skype Translator. They also introduced a cloud-based API (the *Translator Speech API*),[1] offering speech translation services, using DL as the core algorithm. This service can be called through the REST open interface, so developers can integrate a speech recognition service into their applications easily. Nevertheless, mobile applications using this service can only work with internet connection (client-server mode). NMT technology was also released as open source by the Harvard NLP group with the OpenNMT package.[2]

Amazon Polly[3] is a Amazon AI text-to-speech service that applies advanced DL technologies to turn text into lifelike speech. By using this service, Android developers can quickly build speech-enabled apps that work in multiple languages. *Amazon Lex*[4] is a Amazon AI speech-to-text service that can be integrated to mobile apps (both iOS ans Android platforms) to enhance conversational interfaces for any applications using voice and text. This service is built upon DL techniques. Also IBM has recently introduced a speech-to-text and text-to-speech service API[5] that allows developers adding speech recognition capabilities to their applications. This service is built upon convolutional neural networks and works in client-server mode.

5. DISCUSSION, OPEN DIRECTIONS, AND CONCLUSIONS

In this article, we went through the basics of DL for multimedia, and focused on the main components of DL for mobile environments: the low-complexity DL algorithms,

[1]https://www.microsoft.com/en-us/translator/speech.aspx.
[2]www.opennmt.net.
[3]https://aws.amazon.com/amazon-ai/.
[4]https://aws.amazon.com/amazon-ai/.
[5]https://www.ibm.com/watson/developercloud/doc/speech-to-text/.

the software frameworks that are optimized for mobile (or, in general, resource-constrained) environments, and the specialized hardware that is or can be part of mobile devices for supporting the computationally expensive processes of deep network training and inference. We also highlighted several applications of DL in the mobile, which give us an overview of the different possibilities for real-life usage of this technology.

Concerning the open issues at the core technology level, we argue that there is room and need for progress in all fronts. DL architectures for the desktop/server environment are becoming more and more complex, with the number of layers sometimes exploding from the 20 or so that we could find in state-of-the-art implementations just five years ago, to 1,000 or more; this brings improved performance but also new challenges in porting these advanced architectures and the performance that they can achieve in resource-constrained mobile platforms. Living up to this challenge calls for techniques for optimizing such architectures by "compressing" them (in amount of computations and in memory footprint) without loss of learning performance; developing new hardware that can optimally cater to the needs of these architectures (often going beyond general-purpose processors) and to the energy constraints that exist in the mobile world; and developing new or extending the existing software frameworks for the mobile, which are necessary for coupling the deep network architectures and the computing capabilities of the mobile devices in an optimal way.

Concerning the applications of DL in the mobile, we believe that so far we have seen only the tip of the iceberg. There are already several applications that can be found in the literature (some of them surveyed in Section 4, above), and these are sufficient for answering "Yes" to the question *Can DL Revolutionize Mobile Sensing?* that was raised in Ref. [62]. Multimedia processing and DL can indeed be integrated to work in mobile devices. The early and still popular approach of using the mobile devices just as sensor and actuator devices, while the main processing and data storage services for DL are located in servers, can indeed support some interesting application scenarios. Nevertheless, as mobile devices become more powerful, we will be seeing more and more applications running DL engines on the mobile, alleviating the burden of maintaining internet connectivity (which is also power-consuming) and complex server infrastructure. And this will open a new world of possibilities for mobile applications that have the potential to make the currently "smart" mobile devices look like the early cell-phones of the '90s, compared to the DL-enabled mobile devices of the near future. Stay tuned!

REFERENCES

[1] Caffe for Android. Retrieved from https://github.com/sh1r0/caffe-android-lib. Accessed: 2017-02-20.

[2] CaffeOnSpark. Retrieved from https://github.com/yahoo/CaffeOnSpark. Accessed: 2017-02-20.

[3] Chainer. Retrieved from http://chainer.org. Accessed: 2017-02-20.

[4] Keras. Retrieved from https://keras.io/. Accessed: 2017-02-20.

[5] Lasagne. Retrieved from https://lasagne.readthedocs.io. Accessed: 2017-02-20.

[6] MXNet. Retrieved from http://mxnet.io/. Accessed: 2017-02-20.

[7] Neon. Retrieved from http://neon.nervanasys.com/index.html/index.html. Accessed: 2017-02-20.

[8] PyTorch. Retrieved from http://pytorch.org/. Accessed: 2017-02-20.

[9] TensorFlowOnSpark. Retrieved from https://github.com/yahoo/TensorFlowOnSpark. Accessed: 2017-02-20.

[10] Torch. Retrieved from http://torch.ch/. Accessed: 2017-02-20.

[11] Torch for Android. Retrieved from https://github.com/soumith/torch-android. Accessed: 2017-02-20.

[12] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, et al. 2016. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *CoRR* abs/1603.04467 (2016). Retrieved from http://arxiv.org/abs/1603.04467.

[13] Richard Adhikari. 2016. Google, Movidius to Bring Deep Learning to Mobile Devices. Retrieved from http://www.technewsworld.com/story/83052.html (Jan 2016).

[14] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermüller, Dzmitry Bahdanau, Nicolas Ballas, et al. 2016. Theano: A python framework for fast computation of mathematical expressions. *CoRR* abs/1605.02688 (2016). Retrieved from http://arxiv.org/abs/1605.02688.

[15] Jose M. Alvarez and Lars Petersson. 2016. DecomposeMe: Simplifying ConvNets for end-to-end learning. *CoRR* abs/1606.05426 (2016). Retrieved from http://arxiv.org/abs/1606.05426.

[16] Torbjrn Morland Amund Tveit and Thomas Brox Rst. DeepLearningKit—An Open Source Deep Learning Framework for Apple's iOS, OS X, and tvOS developed in Metal and Swift. Retrieved from https://arxiv.org/abs/1605.04614"⟩https://arxiv.org/abs/1605.04614. Accessed: 2017-02-20.

[17] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.* 13, 3 (2017), 32:1–32:18.

[18] David Bacon, Rodric Rabbah, and Sunil Shukla. 2013. FPGA programming for the masses. *Queue* 11, 2, Article 40 (Feb. 2013), 13 pages. DOI:http://dx.doi.org/10.1145/2436696.2443836

[19] Dana H. Ballard. Modular learning in neural networks. In *Proceedings of the 6th National Conference on Artificial Intelligence*, K. Forbus and H. Shrobe (Eds.). Morgan Kaufmann, San Francisco, CA. 279–284.

[20] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19*, P. B. Schölkopf, J. C. Platt, and T. Hoffman (Eds.). MIT Press, 153–160.

[21] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.* 5, 2 (1994), 157–166.

[22] B. Biggio, G. Fumera, and F. Roli. 2014. Security evaluation of pattern classifiers under attack. *IEEE Trans. Knowl. Data Eng.* 26(4) (2014), 984–996.

[23] G. Castellano, A. M. Fanelli, and M. Pelillo. 1997. An iterative pruning algorithm for feedforward neural networks. *IEEE Trans. Neural Netw.* 8, 3 (1997), 519–531.

[24] Kumar Chellapilla, Sidd Puri, and Patrice Simard. 2006. High performance convolutional neural networks for document processing. In *Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition*. Suvisoft.

[25] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. ACM, New York, 269–284. DOI:http://dx.doi.org/10.1145/2541940.2541967

[26] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*. 2285–2294.

[27] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. 2014. DaDianNao: A machine-learning supercomputer. In *Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 609–622. DOI:http://dx.doi.org/10.1109/MICRO.2014.58

[28] Y. H. Chen, T. Krishna, J. Emer, and V. Sze. 2016. 14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC'16)*. 262–263. DOI:http://dx.doi.org/10.1109/ISSCC.2016.7418007

[29] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. CUDNN: Efficient primitives for deep learning. arXiv:1410.0759 (2014).

[30] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an efficient and scalable deep learning training system. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Broomfield, CO, 571–582.

[31] Wonje Choi, Karthi Duraisamy, Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Radu Marculescu, and Diana Marculescu. 2016. Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'16)*. ACM, New York, Article 13, 10 pages. DOI:http://dx.doi.org/10.1145/2968455.2968510

[32] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. 2011. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence—Volume Two (IJCAI'11)*. 1237–1242.

[33] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. 2013. Deep learning with COTS HPC systems. In *Proceedings of the 30th International Conference on Machine*

Learning (ICML'13), Sanjoy Dasgupta and David Mcallester (Eds.), Vol. 28. JMLR Workshop and Conference Proceedings, 1337–1345.

[34] Maxwell D. Collins and Pushmeet Kohli. 2014. Memory bounded deep convolutional networks. *CoRR* abs/1412.1442 (2014).

[35] Henggang Cui, Hao Zhang, Gregory R. Ganger, Phillip B. Gibbons, and Eric P. Xing. 2016. GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys'16)*. ACM, New York, Article 4, 16 pages. DOI:http://dx.doi.org/10.1145/2901318.2901323

[36] Yann Le Cun, John S. Denker, and Sara A. Solla. 1990. Optimal brain damage. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 598–605.

[37] O. E. David and N. S. Netanyahu. 2015. DeepSign: Deep learning for automatic malware signature generation and classification. In *Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN'15)*. 1–8. DOI:http://dx.doi.org/10.1109/IJCNN.2015.7280815

[38] S. Dreyfus. 1973. The computational solution of optimal control problems with time lag. *IEEE Trans. Automat. Control* 18, 4 (1973), 383–385.

[39] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.* 11 (2010), 625–660.

[40] Y. Gao, N. Zhang, H. Wang, X. Ding, X. Ye, G. Chen, and Y. Cao. 2016. iHear food: Eating detection using commodity bluetooth headsets. In *Proceedings of the 2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE'16)*. 163–172. DOI:http://dx.doi.org/10.1109/CHASE.2016.14

[41] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello. 2014. A 240 G-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 696–701. DOI:http://dx.doi.org/10.1109/CVPRW.2014.106

[42] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *CoRR* abs/1412.6115 (2014). Retrieved from http://arxiv.org/abs/1412.6115.

[43] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680.

[44] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR* abs/1510.00149 (2015). http://arxiv.org/abs/1510.00149

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*.

[46] Donald O. Hebb. 1949. *The Organization of Behavior: A Neuropsychological Theory*. Wiley.

[47] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Comput.* 18, 7 (2006), 1527–1554.

[48] S. Hochreiter. 1991. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München (1991).

[49] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jrgen Schmidhuber. 2001. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies (2001).

[50] S. Hou, A. Saas, L. Chen, and Y. Ye. 2016. Deep4MalDroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In *Proceedings of the 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW'16)*. 104–111. DOI:http://dx.doi.org/10.1109/WIW.2016.040

[51] Loc Nguyen Huynh, Rajesh Krishna Balan, and Youngki Lee. 2016. DeepSense: A GPU-based deep convolutional neural network framework on commodity mobile devices. In *Proceedings of the 2016 Workshop on Wearable Systems and Applications (WearSys'16)*. ACM, New York, 25–30. DOI:http://dx.doi.org/10.1145/2935643.2935650

[52] IBM Corporation. 2016. Introducing A Brain-inspired Computer and an End-to-End Ecosystem that Could Revolutionize Computing. Retrieved from http://www.research.ibm.com/articles/Brain-chip.shtml (2016).

[53] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM'14)*. ACM, New York, 675–678. DOI:http://dx.doi.org/10.1145/2647868.2654889

[54] V. Jindal. 2016. Integrating mobile and cloud for PPG signal selection to monitor heart rate during intensive physical exercise. In *Proceedings of the 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft'16)*. 36–37. DOI:http://dx.doi.org/10.1109/MobileSoft.2016.027

[55] Edward Kim, Miguel Corte-Real, and Zubair Baloch. 2016. A deep semantic mobile application for thyroid cytopathology (2016). DOI:http://dx.doi.org/10.1117/12.2216468

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, P. Bartlett, F.c.n. Pereira, C.j.c. Burges, L. Bottou, and K.q. Weinberger (Eds.). 1106–1114. Retrieved from http://books.nips.cc/papers/files/nips25/NIPS2012_0534.pdf.

[57] P. Kuhad, A. Yassine, and S. Shimohammadi. 2015. Using distance estimation and deep learning to simplify calibration in food calorie measurement. In *Proceedings of the 2015 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA'15)*. 1–6. DOI:http://dx.doi.org/10.1109/CIVEMSA.2015.7158594

[58] G. Lacey, G. W. Taylor, and S. Areibi. 2016. Deep learning on FPGAs: Past, present, and future. *ArXiv e-prints* (Feb. 2016). arXiv:cs.DC/1602.04283.

[59] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. 2016. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'16)*. 1–12. DOI:http://dx.doi.org/10.1109/IPSN.2016.7460664

[60] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2015. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In *Proceedings of the 2015 International Workshop on Internet of Things Towards Applications (IoT-App'15)*. ACM, New York, 7–12. DOI:http://dx.doi.org/10.1145/2820975.2820980

[61] Nicholas D. Lane and Petko Georgiev. 2015a. Can deep learning revolutionize mobile sensing? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile'15)*. 117–122. Retrieved from http://doi.acm.org/10.1145/2699343.2699349.

[62] Nicholas D. Lane and Petko Georgiev. 2015b. Can deep learning revolutionize mobile sensing? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile'15)*. ACM, New York, 117–122. DOI:http://dx.doi.org/10.1145/2699343.2699349

[63] Nicholas D. Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'15)*. ACM, New York, 283–294. DOI:http://dx.doi.org/10.1145/2750858.2804262

[64] Seyyed Salar Latifi Oskouei, Hossein Golestani, Matin Hashemi, and Soheil Ghiasi. 2016. CNNdroid: GPU-accelerated execution of trained deep convolutional neural networks on android. In *Proceedings of the 2016 ACM on Multimedia Conference (MM'16)*. ACM, New York, 1201–1205. DOI:http://dx.doi.org/10.1145/2964284.2973801

[65] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1, 4 (1989), 541–551.

[66] Jemin Lee, Jinse Kwon, and Hyungshin Kim. 2016. Reducing distraction of smartwatch users with deep learning. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct (MobileHCI'16)*. ACM, New York, 948–953. DOI:http://dx.doi.org/10.1145/2957265.2962662

[67] Sicong Liu and Junzhao Du. 2016. Poster: MobiEar-building an environment-independent acoustic sensing platform for the deaf using deep learning. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion (MobiSys'16 Companion)*. ACM, New York, 50–50. DOI:http://dx.doi.org/10.1145/2938559.2948831

[68] Warren S. McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* 5, 4 (1943), 115–133.

[69] Ian McGraw, Rohit Prabhavalkar, Raziel Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Hasim Sak, Alexander Gruenstein, Franoise Beaufays, and Carolina Parada. 2016. Personalized speech recognition on mobile devices. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP'16)*.

[70] Michele Merler, Hui Wu, Rosario Uceda-Sosa, Quoc-Bao Nguyen, and John R. Smith. 2016. Snap, eat, RepEat: A food recognition engine for dietary logging. In *Proceedings of the 2Nd International Workshop on Multimedia Assisted Dietary Management (MADiMa'16)*. ACM, New York, 31–40. DOI:http://dx.doi.org/10.1145/2986035.2986036

[71] Gaurav Mittal, Kaushal B. Yagnik, Mohit Garg, and Narayanan C. Krishnan. 2016. SpotGarbage: Smartphone app to detect garbage using deep learning. In *Proceedings of the 2016 ACM International*

Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16). ACM, New York, 940–945. DOI:http://dx.doi.org/10.1145/2971648.2971731

[72] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. 2016. Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2574–2582.

[73] Matthew W. Moskewicz, Forrest N. Iandola, and Kurt Keutzer. 2016. Boda-RTC: Productive generation of portable, efficient code for convolutional neural networks on mobile computing platforms. *CoRR* abs/1606.00094 (2016). Retrieved from http://arxiv.org/abs/1606.00094.

[74] Movidius. 2017. Embedded Neural Network Compute Framework: Fathom. Retrieved from https://www.movidius.com/solutions/machine-vision-algorithms/machine-learning (2017).

[75] Kumpati S. Narendra, Senior Member, and M. A. L. Thathachar. 1974. Learning automata—A survey. *IEEE Trans. Syst. Man. Cybernet.* (1974), 323–334.

[76] NVIDIA. 2017. Next-Gen Smartphones, Tablets, Devices. Retrieved from http://www.nvidia.com/object/tegra-phones-tablet s.html (2017).

[77] NVIDIA Corporation. 2017. Embedded Systems Developer Kits & Modules. Retrieved from http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html (2017).

[78] Nvidia, CUDA. 2010. Programming guide (2010).

[79] Sri Vijay Bharat Peddi, Pallavi Kuhad, Abdulsalam Yassine, Parisa Pouladzadeh, Shervin Shirmohammadi, and Ali Asghar Nazari Shirehjini. 2017. An intelligent cloud-based data processing broker for mobile e-health multimedia applications. *Future Gen. Comput. Syst.* 66 (2017), 71–86. DOI:http://dx.doi.org/10.1016/j.future.2016.03.019

[80] Pai Peng, Hongxiang Chen, Lidan Shou, Ke Chen, Gang Chen, and Chang Xu. 2015. DeepCamera: A unified framework for recognizing places-of-interest based on deep ConvNets. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM'15)*. ACM, New York, 1891–1894. DOI:http://dx.doi.org/10.1145/2806416.2806620

[81] Qualcomm Technologies, Inc. 2016. Qualcomm Helps Make Your Mobile Devices Smarter With New Snapdragon Machine Learning Software Development Kit. Retrieved from https://www.qualcomm.com/news/releases/2016/05/02/qualcomm-helps-make-your-mobile-devices-smarter-new-snapdragon-machine (2016).

[82] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. 2009. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*. 873–880.

[83] R. Reed. Pruning algorithms-A survey. *Trans. Neur. Netw.* 4, 5, 740–747.

[84] F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* (1958), 65–386.

[85] F. Rosenblatt. 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books.

[86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, David E. Rumelhart and James L. Mcclelland (Eds.). MIT Press, Cambridge, MA, 318–362.

[87] Jason Sanders and Edward Kandrot. 2010. *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Addison-Wesley Professional.

[88] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. 2016. Accessorize to a crime: Real and stealthy attacks on state-ofthe-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1528–1540.

[89] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. 2015. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 3088–3096.

[90] Guillaume Soulié, Vincent Gripon, and Maëlys Robert. 2016. *Compression of Deep Neural Networks on the Fly*. Springer International Publishing, 153–160.

[91] Slawomir W. Stepniewski and Andy J. Keane. 1997. Pruning backpropagation neural networks using modern stochastic optimisation techniques. *Neural Comput. Applic.* 5, 2 (1997), 76–98.

[92] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)*. MIT Press, Cambridge, MA, 3104–3112. Retrieved from http://dl.acm.org/citation.cfm?id=2969033.2969173

[93] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'15)*.

[94] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations*.

[95] Ryosuke Tanno, Koichi Okamoto, and Keiji Yanai. 2016. DeepFoodCam: A DCNN-based real-time mobile food recognition system. In *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management (MADiMa'16)*. ACM, New York, 89–89. DOI:http://dx.doi.org/10.1145/2986035.2986044

[96] Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York.

[97] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning (ICML'08)*. 1096–1103.

[98] Joel Emer Vivienne Sze. 2016. Chip could bring deep learning to mobile devices. Retrieved from http://www.eurekalert.org/pub_releases/2016-02/m iot-ccb020316.php (2016).

[99] Saiwen Wang, Jie Song, Jaime Lien, Ivan Poupyrev, and Otmar Hilliges. 2016. Interacting with soli: Exploring fine-grained dynamic gesture recognition in the radio-frequency spectrum. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST'16)*. ACM, New York, 851–860. DOI:http://dx.doi.org/10.1145/2984511.2984565

[100] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. *CoRR* abs/1608.03665 (2016).

[101] J. Weng, N. Ahuja, and T. S. Huang. 1992. Cresceptron: A self-organizing neural network which grows adaptively. In *Proceedings of the 1992 IJCNN International Joint Conference on Neural Networks*, Vol. 1. 576–581.

[102] Paul J. Werbos. 1982. *Applications of Advances in Nonlinear Sensitivity Analysis*. Springer, Berlin . 762–770.

[103] Bernard Widrow and Marcian E. Hoff. 1962. *Associative Storage and Retrieval of Digital Information in Networks of Adaptive "Neurons."* Springer, Boston, MA, 160–160.

[104] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016a. Quantized convolutional neural networks for mobile devices. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 4820–4828.

[105] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, and et al. 2016b. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR* abs/1609.08144 (2016). Retrieved from http://arxiv.org/abs/1609.08144.

[106] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: Deep learning in android malware detection. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 371–372. DOI:http://dx.doi.org/10.1145/2740070.2631434

[107] Z. Yuan, Y. Lu, and Y. Xue. 2016. Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Sci. Technol.* 21, 1 (Feb. 2016), 114–123. DOI:http://dx.doi.org/10.1109/TST.2016.7399288

[108] Matthew D. Zeiler and Rob Fergus. 2014. *Visualizing and Understanding Convolutional Networks*. Springer International Publishing, 818–833.

[109] Q. Zhang, H. Li, Z. Sun, Z. He, and T. Tan. 2016. Exploring complementary features for iris recognition on mobile devices. In *Proceedings of the 2016 International Conference on Biometrics (ICB'16)*. 1–8. DOI:http://dx.doi.org/10.1109/ICB.2016.7550079

[110] Sixin Zhang, Anna E. Choromanska, and Yann LeCun. 2015. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 685–693.

[111] J. Zhu, A. Pande, P. Mohapatra, and J. J. Han. 2015. Using deep learning for energy expenditure estimation with wearable sensors. In *Proceedings of the 2015 17th International Conference on E-health Networking, Application Services (HealthCom'15)*. 501–506. DOI:http://dx.doi.org/10.1109/HealthCom.2015.7454554

Received March 2017; revised April 2017; accepted April 2017