CHAPTER

Non-static TinyML for ad hoc networked devices 10

Evangelia Fragkou[b] **and Dimitrios Katsaros**

Department of Electrical and Computer Engineering, Volos, Greece

10.1 Introduction

Nowadays, the availability of data produced by edge devices is high, leading to an increase in techniques able to process these data, so new methods of machine learning/deep learning (ML/DL) have emerged. In the existing literature, ML/DL models are trained in devices without computational limitations like servers or the cloud and then the model can be deployed in any other device, running the known inference task. However, the tremendous success of the Internet of Things (IoT) networks [1] has solemnified the demands of real-time processing of data at the edge. Processing data at the core from which they originate provides data safety, since information remains at the edge device, and low latency, as data do not have to be transmitted to a server for processing, resulting in less total energy consumption.

However, taking into account the lack of resources, even inference tasks are not capable of running on devices with extremely restricted computational resources, like storage or energy consumption limitations, for instance microcontroller units (MCU) allotted with some megabytes of flash memory. This bottleneck in ML/DL deployment led to the development of *tiny machine learning* (*TinyML*) [1–3]. TinyML aims at producing compact models that can satisfactorily run inference tasks in the aforementioned devices. The proposed method includes off-line training of a model, and then, using a suitable tool, e.g., TensorFlow Lite [4], an interpreter is used to convert the pretrained model into a small memory-efficient one, applicable to an MCU, in which it is further applied.

Although progress has been made, taking into account that inference tasks can now be deployed on extremely resource-limited devices in some cases, inference-based methodologies place constraints on the efforts, so the really successful models can surface. This occurs since these methods are not able to adjust to new incoming data. The reason is that in inference tasks we can only modify the classifier of the network, not the core of the model responsible for the final classification (*static TinyML*). So, as mentioned in [5], the processing of data at the edge necessitates

[b] The research work is supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 5631).

the introduction of *reformable TinyML* (non-static TinyML) techniques, or in other words, the development of techniques which reduce the size of neural networks and thus the number of parameters during training, leading to better generalization of the processed data and further performance enhancement. In [6], the techniques called *pruning* and *sparsification* are extensively demonstrated, which are among the most widely known methods of accelerating the learning process of the model by eliminating unnecessary weights/connections/filters.

Despite the fact that non-static TinyML seems to be a productive means of successfully addressing the problem of proper raw data integration into the model, it demands both abundance of data and enormous execution time in order for the model to converge, conditions that are not in line with the TinyML device specifications. This problem is tackled to a great extent by *transfer learning* (TL) [7]. TL is a promising technique in which knowledge of a trained model on a task A (usually on a computationally efficient machine with data availability, like servers) can be used as preknowledge and further applied to an aforementioned device, e.g., an MCU, to learn its own task B. It is deduced in [8] that pretrained weights used for neural network initialization can lead to faster convergence of the model, irrespective of the task they were produced by.

However, spurring the deployment of training procedures at the edge and making endeavors to perform effectually learning "on the fly," it is significant to design and thus construct suitable decentralized neural network topologies (*backbones*) that can support cooperative training (*federated learning* [FL]). Edge devices lack not only computational resources, but also availability of data, making FL, in which a global model is trained by many devices, sending each other, e.g., gradients, instead of the data themselves, a promising method for successful cooperative training. There is a lot of work regarding centralized FL schemes [9], but semi/hybrid-decentralized ones [10–12] or stipulated purely decentralized schemes, like the ones edge learning concepts demand, are still in their infancy [13–17]. The cost overhead and reliability issues incurred by all-to-all nodes communication in an ad hoc network are major challenges that have to be further examined.

The rest of the chapter is organized as follows. In Section 10.2, ad hoc network topology construction and existing hardware capabilities are proposed. In Section 10.3, efficient pruning techniques are demonstrated. In Section 10.4, TL methodologies are introduced and analyzed. In Section 10.5, purely distributed FL methodologies are described. Section 10.6 introduces the main "bottlenecks" of TinyML and FL in peer-to-peer networks. Finally, Section 10.7 concludes this chapter.

10.2 Backbones and TinyML boards

10.2.1 Backbone construction

The communication reliability in edge distributed environments is of a great importance, since the research community focuses on the exploration of edge data processing and hence device independence. In *ad hoc* or *peer-to-peer* or *device-to-device* networks, all devices belonging to this network can communicate among each other without requiring a coordination of, e.g., a server node. However, these types of networks lack energy resources, since they consist of devices like sensors and hence vast amounts of information diffused to the network can cause resource depletion. So, this situation necessitates the construction of trustworthy ad hoc *backbones*, with the aim of reducing communication overhead by controlling information routing among the nodes of different local area networks (LANs) or subnetworks they need to interconnect. For instance, in [18], the combination of a *LoRa* low-power wide area network (LPWAN) single-hop star-like network topology and a mesh network architecture is used in order to implement cooperative learning in IoT networks. They take advantage of the LoRaWAN architecture protocol and the benefits that an LPWAN network topology offers, like long-range communication, low power consumption, or even low data rates, in conjunction with all-to-all communication that a mesh network provides to perform FL (see Section 10.5). However, the communication cost remains high, since every node can communicate with one another. Although there is a lot of existing literature [19], referring to smart routing protocols in distributed networks, there is little contribution regarding multi-layer resource-scarce networks. The main goal is to constrain routing of information, for instance as the distributed state-of-the art algorithms introduced in [20,21], by constructing backbones with small *cardinality*, meaning that every node is urged to transfer information between the minimum nodes, required in order for the message to be forwarded through the whole ad hoc network. By reducing unnecessary information exchange between the nodes, the energy consumption, required for every device to collaborate is reduced too, making these topologies “hospitable” to support collaborative training techniques (like FL methods) that will be further discussed in the next sections.

10.2.2 TinyML boards and embedded systems

Despite the endeavor for developing robust decentralized topology configurations, it is significant to take into account the nature of TinyML (see Section 10.3) edge devices that are part of the aforementioned networks. Specifically, TinyML embedded systems consist of MB or kB of flash memory, while they do not support energy-consuming applications, which provokes difficulty in deploying conventional approaches of ML/DL. TinyML devices, like the Raspberry Pi devices (*Raspberry Pi 3B+, Raspberry Pi 4*), based on Cortex A53 and A72 processors, respectively, belong to the family of low-power devices and have less than 2 GB of RAM [22,23].

Going further to the ultralow-power devices, like *Arduino Nano 33 BLE Sense*, the availability of flash memory is approximately 1 MB to 2 MB. This series of Arduino models features a Cortex-M4 processor (especially designed for ultralow-power devices and used for general purposes) [24,25], which reinforces DL/ML tasks. The series of STM32 microcontrollers, mentioned in [22,26,27], are also based on the family of Cortex processors. For example, the high-performance MCU *STM32F7* features a Cortex-M7 microprocessor with approximately 512 kB of SRAM and 2 MB of flash, while the general-purpose MCU *STM32F401* consists only of 96 kB of SRAM and 128 kB to 512 kB of flash memory. Similar performance is gained by the *STM NUCLEO L496ZG* (L4 – ultralow-power MCU), *STM DISCO F496NI* (F4 – over-balanced MCU), and *STM NUCLEO F767ZI* (F7 – high-performance MCU).

Furthermore, in [28], [29], PULP (Parallel Ultra-Low-Power Processing Platform) and Risc-V-based microprocessors, namely, *GAP8*, *VEGA*, and *Mr.Wolf*, are used for supporting parallel processing and multi-caching.

10.3 Neural network model reduction

ML/DL methods have high computational demands, since they need both sufficient data and much training execution time, in order to learn these data and converge. Optimization efforts like multicore parallel execution regarding both CPUs and GPUs, deep learning caching techniques [30], etc., try to bridge the gap between the efficient training/inference of a model and the enormous execution time/computational cost needed by such large models, e.g., GPT-x [31], having billions of connections among its neurons. However, talking about resource-constrained devices/environments, e.g., the Internet of Things (IoT), this is not feasible, taking into account that these devices lack computational power and memory (approximately MB or kB of RAM for example in sensors, embedded systems, etc., as described in Section 10.2.2) and the existence of a tremendous number of parameters, i.e., weights impact both inference time and training time. Conventional TinyML approaches support inference in these devices by compressing the ML/DL code and then deploying it in them. However, this technique does not enable adaptation to new raw data and thus it cannot provide satisfactory results in the future. In this work, we are going to demonstrate efficient methods (TinyML [32]) of minimizing computational cost and hence memory requirements of ML/DL algorithms during the training phase, so as to be capable of running on the aforementioned devices. A promising and extensively demonstrated concept of reducing computational demands of ML/DL methods is to reduce the size of the neural network we want to train by making it more "sparse," meaning that we can eliminate elements of the network (pruning) that do not or minimally contribute to its performance. We can classify pruning techniques into two main categories: one in which neurons are removed (e.g., dropout [33]) and one in which synapses (connections among neurons) are removed. Moreover, pruning is classified as static pruning, in which we prune the network in its initialization phase and fine-tune it in order to converge, or dynamic pruning, in which we prune and regrow the network iteratively in every epoch. The crucial part is to determine which pruning criteria work better in every case, regarding the nature of the network we use. It was observed in [34] and [6] that random pruning techniques are more efficient when they are applied for weight removal rather than neuron/filter removal in the network initialization phase. However, even in this case randomly eliminating information from the network may severely reduce network performance.

There exist straightforward techniques for pruning (removing unnecessary weights) after the training of a neural network and before inference, so as to accelerate the operation of the network. However, the real challenge is to do this during training. So how can neural model reduction be performed?

10.3.1 The lottery ticket hypothesis

Contemporary experience shows that there exists a small part of a dense neural network that can achieve (almost) the same test accuracy as the whole network. This observation was initially tested on fully connected multilayer perceptrons and convolutional neural networks (CNNs) in vision tasks and was summarized in the famous *lottery ticket hypothesis* described in [35] as follows:

A randomly-initialized, dense neural network contains a subnetwork (winning ticket) that is initialized such that – when trained in isolation – it can match the test accuracy of the original network after training for at most the same number of iterations.

In order to identify a winning ticket, an appropriate algorithm initializes and trains the whole network and after the completion of training removes the connections that correspond to the weights with the smallest values. After that pruning step, the algorithm restores the weights of the surviving connections to the values they got at the initialization step! The training and pruning could take place once at the end of the training (one-shot) or every time after a specific number of iterations, eliminating only a percentage of the surviving connections (iterative pruning).

A surge of works followed the initial article that generalized this technique to the *multi-prize lottery ticket hypothesis* [36] or applied the initial ideas to spiking neural networks [26,37,38], in few-shot learning [39], and so on.

10.3.2 Topology sparsification prospects derived from network science disciplines

The challenge in ML and the reason of introducing TinyML is that the learning process has shifted to the edge (IoT networks, sensors, etc.), meaning that every device has to be able to learn its own data, without being in need of, e.g., a server which reinforces it. Since the aforementioned devices (see Section 10.2.2) do not have enough computing power, the need to create more compact neural networks that still maintain their high accuracy was the main trigger.

The technique called *sparsification* is promising since it is aimed at reducing model size and hence memory requirements by dropping out unnecessary – regarding the training procedure – nodes/connections of the network. Additionally, there is great inspiration in the structure of the synapses in the human brain and hence the way the human brain processes incoming information. More specifically, the human brain creates thousands of synapses every day as it is exposed to new data (and consequently produces information), while it has the ability to make a decision of which of them to keep, since they are not all equally useful. At the same time, network science introduces the concepts of scale-free networks and based on them describes a variety of networks that we use in everyday life, such as the World Wide Web (web). More meticulously, in scale-free networks, the nodes follow a power law distribution, i.e., the nodes where most synapses end up are considered the most popular and consequently they are the nodes that transfer information between the nodes of the network.

So, inspired by the theory mentioned above, an idea was to remove the connections between the nodes of a bipartite fully connected feedforward *multilayer perceptron* (MLP) neuron network that do not actually contribute to the network (in other words, the weight values of these connections tend to be zero), while concurrently adding as many weights as we removed. The whole process of sparsification takes place during the training phase, in which we iteratively both remove and regrow (*dynamic sparsity during training*, according to [6]) a specific amount of connections (approximately 30%), with the aim of reorganizing the structure of the neural network. The work described in [40] (SET) was the first to correlate the functionality of scale-free networks with that of deep neural networks, but the initialization of synapses in the first untrained network is done in a random way (Erdos–Renyi type network), while only the final network is structured, following a scale-free law. In [41], an optimized version of it was described. In [42], both the initial and the final network are reconstructed after every epoch of training not in a random way, but following network theories such as scale-free or small-world networks, with the aim of strengthening the nodes with the most attached connections (and thus those responsible for the guaranteed dissemination of information in the network) before and after every epoch. Furthermore, taking into account that the weight of a pruned connection is considered to be acquired knowledge [43], an experiment with keeping the weight value before connection removal and reassigning it to the new one was conducted. In other words, the weight values of the connection being restored are either random or the sum of the previous weights being deleted. Although the pretrained weights are some kind of preknowledge, after many connection modifications and hence weight replacements, the first task, for which weights were trained, tends to be "forgotten" (catastrophic forgetting [44] – one of the crucial problems in deep learning methodologies, attracting great scientific interest), and hence the methodology of keeping those weights tends to increase complexity of the algorithm rather than reinforcing model performance regarding accuracy levels in the long run (see the exact results in [42]).

So, practically, five algorithms were implemented, which differ in the way both the initial and the final network are redefined (either variants of scale-free [SF], like the one described in [45,46], or small world [SW], introduced by Watts and Strogatz in 1998 [47]), namely, *SF2SFrand*, *SF2SFba*, *SF2SW*, and *SW2SW*, respectively, with SF2SFrand being the one outperforming the other variants introduced. Regarding SW2SW implementations, we experimented with two values ($p = 0.02$ and $p = 0.075$), given to the rewiring probability of every node in the network, making the network configuration either denser or more random, respectively (see Table 10.2 for further details, in which specifically SF2SFrand seems to retain high orders of accuracy, while reducing training time approximately to 50% in some cases). It is worth highlighting that reinforcing the strongest nodes of every layer, the performance of the network is proportionately enhanced too. However, small world-based implementations fail to make the model generalize efficiently the new data to which it is exposed. The results are not satisfying not only with respect to accuracy levels ($\leq 70\%$), but also with respect to training time needed (hours of training). This occurs due to the fact that this type of network tends to be less strictly structured (increasing randomness, as probability p increases) as the probability of a node connecting to others in a layer increases and therefore the network is more similar to an Erdos–Renyi network. The results regarding the datasets mentioned in Table 10.1 are illustrated in Figs. 10.1 and 10.2 and Table 10.3. For the experimental evaluation, a three-layered MLP neural network is used, consisting of 1000 neurons each and the widely known ReLU activation function (defined as $ReLU(x) = (x > 0)?x : 0$). In order to prevent misleading performance results in favor of a dataset, overfitting or L1 or L2 regularization is applied during the training phase.

There was further examination regarding the case where the percentage of connections, both pruned and restored, is not stable but varies (drawing inspiration from the ideas presented in [48]), linearly (the linear decreasing variation [*LDV*] method), exponentially (the exponential decay [*EXD*] method), or following the cosine rule (oscillating variation [*OSV*]) at each epoch, since from the beginning the pruned ones were insignificant for the process of training. The results [49] show that when the percentage of close-to-zero synapses is modified exponentially or oscillates following a cosine rule function, the accuracy of the model remains approximately 90%, as depicted in Fig. 10.3, and the time needed is also at the same level as for the baseline methods (MLP as bs1 and SET as bs2), since the technique used requires enough calculations, even if the number of connections removed is larger (see Fig. 10.4). The most impressive result, making the aforementioned methods congruous for TinyML devices, is that the memory footprint is further reduced (around 95%) (while tested on the fashion-mnist dataset and four different kinds of neural network topologies; see Table 10.1), as more superfluous connections were removed from the network (see Table 10.4).

10.4 Transfer learning

TL is a widely known technique, which aims at boosting ML model performance and learning ability by bequeathing already acquired knowledge from a source device to a target device [7,57–59]. Specifically, source devices like servers train ML algorithms asynchronously, without computational limitations, while they have plenty of data available, in order to accomplish successfully the training task. On the other hand, the target device (e.g., sensors, microcontrollers, etc.) lacks computational resources and storage capacity, making the deployment of ML algorithms unfeasible. So, according to the TL method, the model initially trained on a source device can be deployed to the target device by "freezing" some of its layers while retraining the remaining ones, using its weights as preknowledge. The specific approach is a promising technique,

Table 10.1 Characteristics of some of the datasets.

Dataset	Instances		Features	Classes
lung (.mat) [50]	203		3312	5
COIL20 (.mat) [51]	1440		1024	20
Fashion-Mnist [52]	train samples 60,000	test samples 10,000	28×28	10
CIFAR-10 [53]	train samples 50,000	test samples 10,000	32×32	10
CIFAR-100 [53]	train samples 50,000	test samples 10,000	32×32	100
ImageNet [54]	train samples 1,281,167	test samples 100,000	224×224	1000
Intel Classification Dataset [55]	train samples 14,000	test samples 3000	150×150	6
Kaggle [56] - temperature of five Chinese cities for a time period of five years (2010–2015).	52,585 measurements of real temperatures in degrees Celsius.			

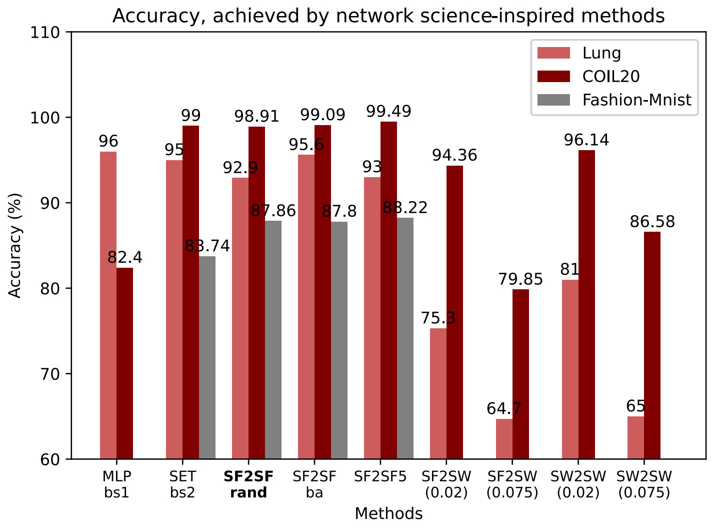

FIGURE 10.1

Accuracy gained during the training of network science-inspired algorithms with datasets mentioned in Table 10.1, described in Section 10.3.

as it combines not only predefined “knowledge,” but also on-device fine-tuning of the network, while simultaneously drastically reducing the time of online training, since it reduces the number of layers involved and therefore the total number of weights processed in the training phase.

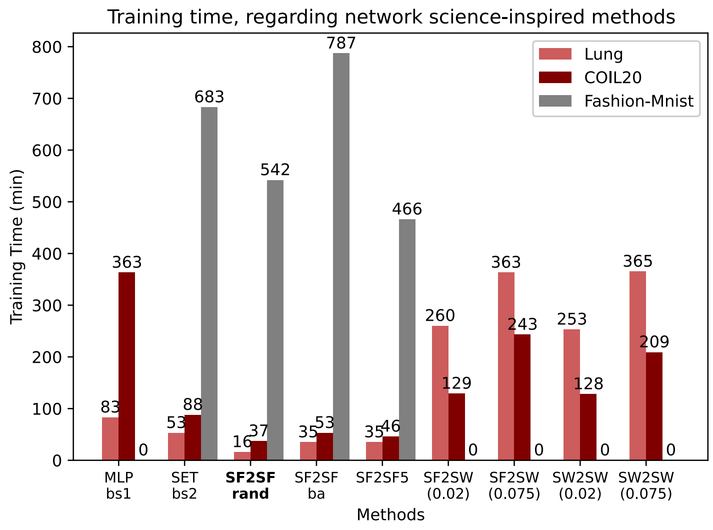

FIGURE 10.2

Training time (in minutes) of network science-inspired algorithms with datasets mentioned in Table 10.1, described in Section 10.3.

Table 10.2 Processes followed by the methods described in Section 10.3.

Method	Process followed
SF2SFbrand	We start from an initial non-arbitrarily connected node topology and conclude with a same type topology through training after linkage removal.
SF2SFba	We start from a Barabasi–Albert-based topology and conclude with a same type topology through training after linkage removal.
SF2SW (either $p = 0.02$ or $p = 0.075$)	We initialize a scale-free-like topology and reconfigure the final network topology into a small world-like network either with rewiring probability $p = 0.02$ (more structured topology) or $p = 0.075$ (more arbitrarily connected nodes) through training after linkage removal.
SW2SW (either $p = 0.02$ or $p = 0.075$)	We firstly initialize a small world-based network and redefine the final network topology into a small world-like network either with rewiring probability $p = 0.02$ (more structured topology) or $p = 0.075$ (more arbitrarily connected nodes) through training after linkage removal.

The most well-known case is the one in which only the last fully connected layer is fine-tuned, reinforcing the neural network model to adapt to the new data it is exposed to in the target device (raw data), as described in [57]. However, examining a classification task (which is one of the most frequently used tasks in ML/DL), we observe that fine-tuning only the last fully connected layer of a CNN, which is responsible for the final categorization of the training data, we can only modify the categories that the network recognizes, not somehow attain the ability of "adding" new categories (non-static training). In other words, in an image classification task,

Table 10.3 Summary of best results (Section 10.3).

Algorithm	**MLP-FC (*baseline1*)**	**SET (*baseline2*)**	**SF2SFrand**	**SF2SFba**	**SF2SF(5)**
Memory reduction (%) (*compared to MLP-FC*)	–	75.2%	76.31%	73.36%	74.00%
Performance on Lung.mat	Accur/time 99.52%/ 1 h 23 min	Accur/time 92.71%/ 38 min	Accur/time 92.9%/ 16 min	Accur/time 95.6%/ 32 min	Accur/time 92.8%/ 35 min
Performance on COIL20.mat	Accur/time 62.5%/ 6 h 3 min	Accur/time 99.37%/ 1 h 28 min	Accur/time 99.16%/ 1 h 42 min	Accur/time 99.16%/ 53 min	Accur/time 99.79%/ 1 h 13 min
Performance on Fashion-Mnist	Accur/time 49.94%/ 2 d 13 h 5 min	Accur/time 83.74%/ 11 h 22 min	Accur/time 87.86%/ 9 h 2 min	Accur/time 87.8%/ 13 h 7 min	Accur/time 88.22%/ 7 h 46 min

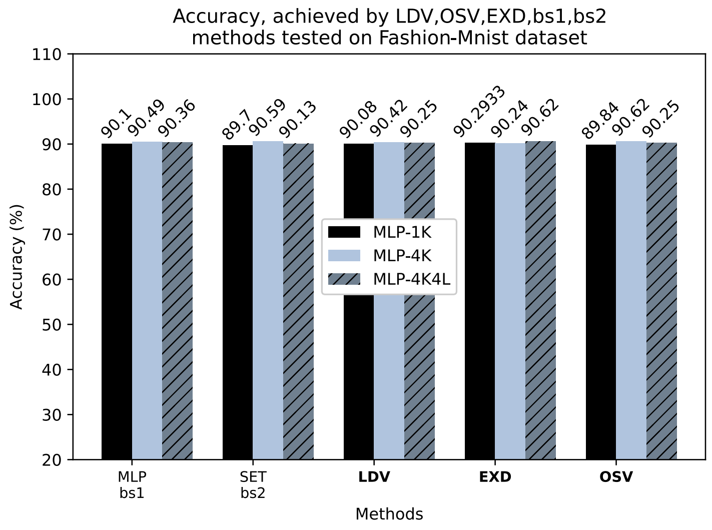

FIGURE 10.3

Accuracy gained during the training of different neural network topologies with LDV, EXD, and OSV algorithms and their baselines, described elaborately in Section 10.3.

the first layers of a CNN network (from whose functionality the main motivation of work [60] is derived) acquaint with the basic features of an input data image, and then this knowledge is transmitted among the next layers so as for the final layers to be able to “understand” the higher levels of the hierarchical structure of the image. This means that the final convolution layer is only responsible for categorizing the data processed by the network. So when the network is exposed to new data, it is of great importance to update the way that leads to the categorization of images and

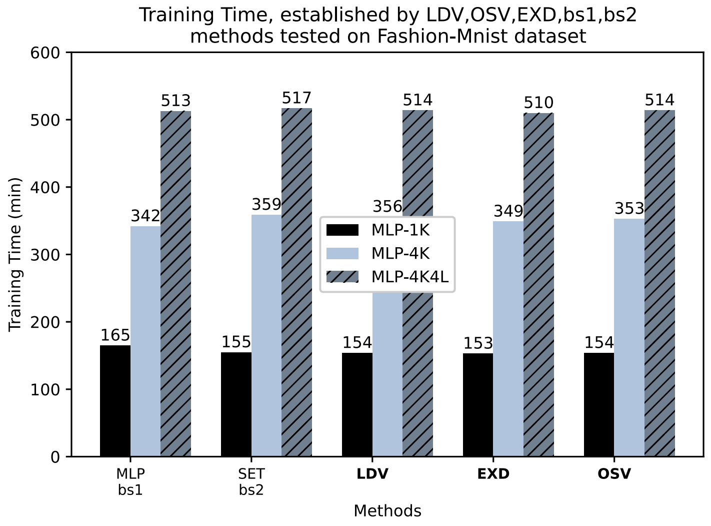

FIGURE 10.4

Training time (in minutes) of different neural network topologies using LDV, EXD, and OSV algorithms and their baselines, described elaborately in Section 10.3.

Table 10.4 Sparse MLP topologies, compared to dense ones, regarding memory footprint when using a non-stable number of pruned connections (Section 10.3).

Neural network architecture (# of hidden layers)	Size before sparsification	Size after sparsification	Reduction rate (%)
1000-1000-1000 (MLP-1K)	2.797.010	120.000	95.7%
4000-1000-4000 (MLP-4K)	11.185.010	350.000	96.8%
4000-2000-2000-1000 (MLP-4K4L)	17.155.010	380.000	97.7%

not merely the classification of the already formed categories. So, we delve into the existing research and try to tackle this problem by considering whether model performance can be improved if more than the last layers participate during the online training phase and more specifically how much its performance would improve if we retrain one or more of the previous (convolution) layers.

So, experiments were conducted [60] in which three different types of networks (Small CNN with 550,000 trainable parameters, EfficientNetB0 with 4,049,571 trainable parameters, and EfficientNetB2 with 7,768,569 trainable parameters) are used, in which more than the last fully connected layers are retrained. We call these approaches F_xC_y, where x and y denote the numbers of both fully connected and convolution layers retrained, respectively. The results, obtained by the experiments that are depicted in Figs. 10.5 to 10.8 and Table 10.5, show that the retraining of the last convolution layer can enhance model performance regarding accuracy, presenting a trade-off in the energy consumption required in order for the convolutional opera-

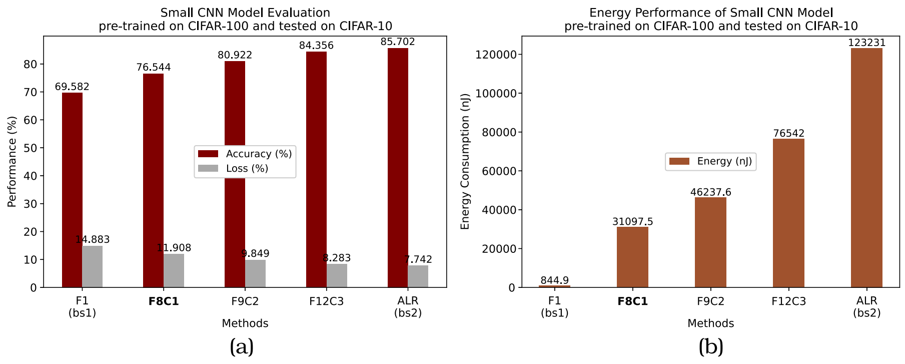

FIGURE 10.5

Performance evaluation (a) and energy consumption (b) using Small CNN neural network and transfer learning algorithms, described in Section 10.4.

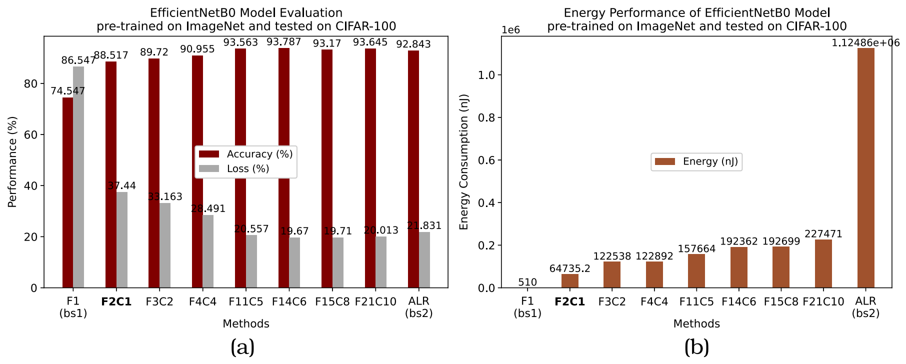

FIGURE 10.6

Performance evaluation (a) and energy consumption (b) using the EfficientNetB0 neural network and transfer learning algorithms, described in Section 10.4.

tion to be executed. In detail, the proposed method, namely, $F_x C_1$, outperforms both baseline methods (either reconfiguring the latter layer of a CNN network [$F1$-$bs1$] or including all the layers in the learning process[ALR-$bs2$]), attaining approximately 19% growth in model accuracy and about 61% faster convergence. To evaluate algorithms, accuracy, energy consumption in joules, and a categorical cross-entropy loss function were selected. Regarding energy consumption, we calculated the FLOPS an algorithm executes in every epoch of training; bearing in mind that a MAC operation in a 45 nm CMOS processor is tantamount to two FLOPS, we proportionately calculate the FLOPS consumed by our proposed approaches. The results are shown in Figs. 10.5 to 10.7.

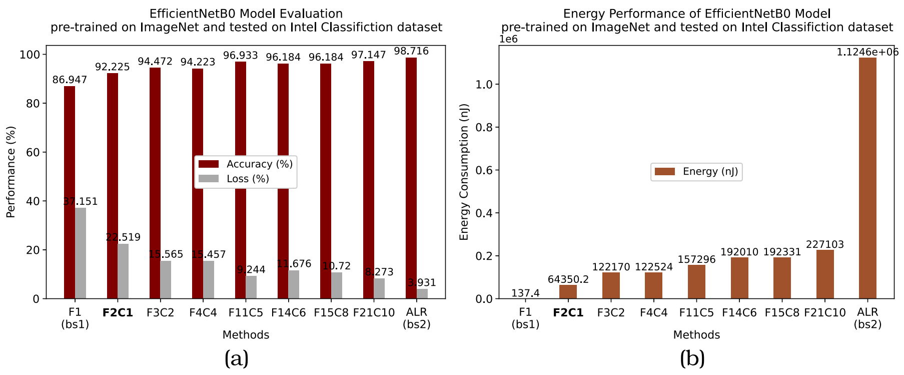

FIGURE 10.7

Performance evaluation (a) and energy consumption (b) using the EfficientNetB0 neural network and transfer learning algorithms, described in Section 10.4.

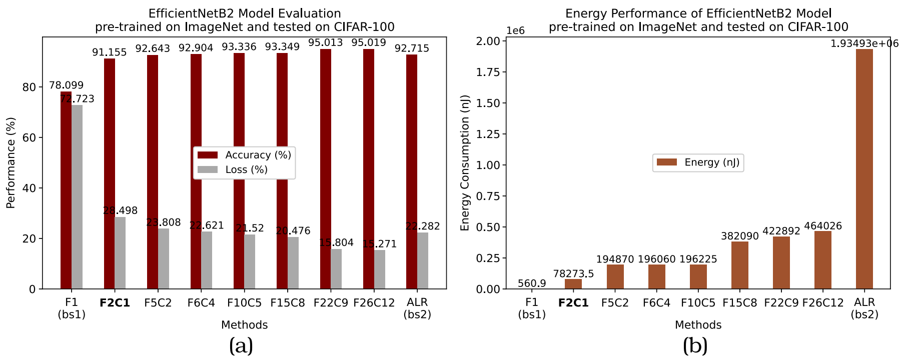

FIGURE 10.8

Performance evaluation (a) and energy consumption (b) using the EfficientNetB2 neural network and transfer learning algorithms, described in Section 10.4.

Table 10.5 Comparison of the proposed $F_x C_1$ method with its main competitor, the *baseline-1* method (Section 10.4).

Model used	Datasets for pretraining/testing	Increase in accuracy (%)	Increase in convergence rate (%)
Small CNN	CIFAR-100/CIFAR-10	10.01	19.98
EfficientNetB0	ImageNet/CIFAR-100	18.79	56.74
EfficientNetB0	ImageNet/Intel Classification	6.09	39.38
EfficientNetB2	ImageNet/CIFAR-100	16.79	60.81

10.5 Purely distributed federated learning

As far as device independence in data processing procedures is concerned, plenty of ways of reducing model parameters and hence the memory footprint (see Section 10.3) were developed, but the lack of data availability and energy constraints regarding data processing are still major shortcomings that affect the training phase of these devices; therefore, the *federated learning* (FL) method has been developed [61].

The FL mechanism aims at enhancing the neural network model learning ability by collaboratively training this model among cooperating edge connected devices, while preserving their data privacy. In other words, every device that participates in training trains the same model with its own data, and after some epochs of training, this device sends information like gradients produced during training (not the data themselves) either to a server, which then aggregates the sent model parameters, updates the model, and further broadcasts the global updated model to the rest of the participating devices (*centralized approach of FL*), or among the other participating devices (*decentralized – purely distributed FL* [PDFL]), which complete the parameter aggregation and hence the update of the global model among one another. In this way, FL takes advantage of data produced in different TinyML devices without incurring both learning process and memory overhead [62], due to plenty of data needed for the process and concurrently keeping data privacy of resource-constrained devices, like sensors, MCUs, etc. (see Section 10.2.2).

However, centralized FL approaches fail to keep up with the requirements an ad hoc network demands, since in a centralized FL scheme there is a suitable network infrastructure that can support communication between the device and the server, while the server has the demanding energy requirements to perform the training process. For these reasons, the transition from a centralized scheme to a decentralized one [9] has begun, taking into consideration the major problem that all-to-all communication costs result in. There are some efforts, mentioned in the existing literature, for example, [13–17,63,64], in which semi- or fully decentralized approaches are presented; however, they are not necessarily geared towards communication alleviating methods, as the following proposed methods. Since resource-scarce ad hoc environments needs decentralized FL solutions that can deal with the high communication cost, selecting which node participates is a critical issue and therefore the concept of *network clustering* is at the forefront of decentralized FL research [11,65].

10.5.1 Similarity-based selection of participating devices in PDFL

A practical way of mitigating information diffusion among the network of participating devices in FL is to fit them into groups (clusters) and only the representative node (device) of every cluster (clusterhead [CH]) is responsible for communicating with the other representative nodes. In this section, the configuration of the clusters is implemented using the MAX-MIN-D algorithm [66] and each CH is selected according to the well-known betweenness centrality algorithm [67], so as for the selected CH to be in the "center" of the cluster, or else to be equidistant from each node belonging to the cluster.

Moreover, an FL heuristic focusing on communication overhead reduction is demonstrated, in which the fact that some devices produce similar data is leveraged (for instance, connected devices, in 1-hop distance apart) and hence not all nodes are necessary in the training procedure. Taking into account the case of dealing with time-series data, the proposed way of seeking data similarity is based on the *discrete Fourier transform* (DFT). DFT mechanisms transform time-series data from the time domain to the frequency domain, easing even the comparison among time series, shifted through time. Furthermore, according to the Parseval's theorem, Euclidean distance can be utilized, even using only some of the first coefficients of the converted-to-frequency-space points, in order to calculate how similar the data of two different devices are. The first few coefficients are an adequate and representative sample, since the larger quantity of energy is gathered in them. The CH makes appropriate comparisons between all the nodes belonging to its cluster and hence eliminates a node or more which contain the same data as a previous node in the same cluster at the same epoch of training. It is worth highlighting that the DFT method for recognizing data similarity can be used not only for time-series data, but also for streaming data or image data, as described in [68,69].

We evaluated the aforementioned mechanism by conducting experiments using real time-series data produced by Kaggle (see Table 10.1). A part of the results is depicted in Figs. 10.9 and 10.10, in which either all nodes participate in the federation ($FEDLp2p\_ANP$) or a similarity-based criterion is used for selecting participating nodes ($FEDLp2p\_SNP$), respectively. In these figures, the performance of a representative node after the global model update phase is demonstrated, since after the global model update, all nodes have exactly the same model parameters and thus this node is an adequate sample in order to evaluate both the accuracy and the loss metric of the model. In the experiment in Fig. 10.10, either the first 5 or the first 10 coefficients of the DFT are leveraged in order to calculate the data similarity among two adjacent nodes. As we can observe, using a similarity-based criterion with 5 coefficients selected, the neural network converges faster than $FEDLp2p\_ANP$ and especially from the 15th round, the loss value remains stable and close to zero, while the accuracy attained by the model is approximately 99%. When running $FEDLp2p\_SNP$ with 10 coefficients, the results are more promising, since the accuracy values do not fluctuate, meaning that the model converges and the loss function is stably close to zero.

Finally, we can conclude that the participation of specific nodes in the federation can reduce the time complexity of algorithms, as the number of rounds of training is reduced due to faster convergence and the amount of data transmitted in the network is lower, reducing communication costs.

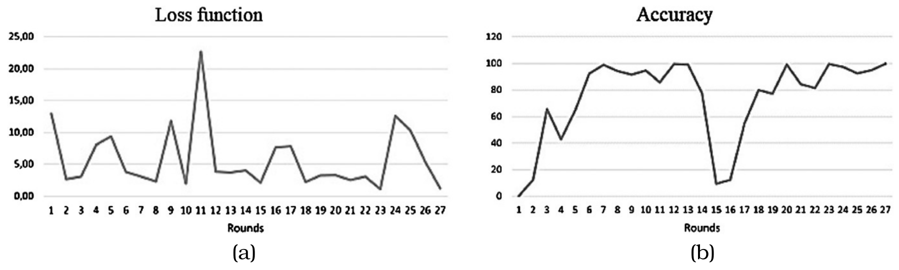

FIGURE 10.9

Demonstration of the performance (loss (a) and accuracy (b)) of $FEDLp2p\_ANP$ when running on a 20-node network, described in Section 10.5.

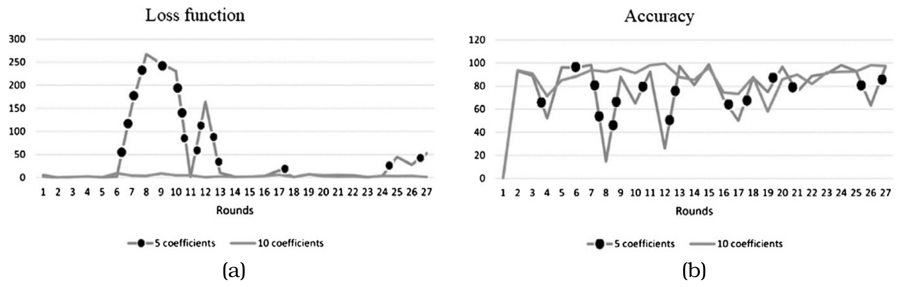

FIGURE 10.10

Demonstration of the performance (loss (a) and accuracy (b)) of $FEDLp2p\_SNP$ when running on a 20-node network, described in Section 10.5.

10.6 Challenges

- **Concept drift.** Taking into account that we are associated with IoT devices, like sensors or MCUs, after preinstalling an initially trained model on them, retraining it using data gathered by the device itself can make these devices converge faster. However, the problem that hinders the efficacy of this technique is that the streaming (raw) data of the target device may be of a different dimensionality (belonging to different feature spaces) compared to the ones inherited by the source device (heterogeneity of data); this phenomenon is called *concept drift*. Additionally, another form of concept drift is *class imbalance*, which is also a major problem in classification modeling approaches. In this case, the quantity of data every class consists of is not equal (or almost equal) between all the classes formed, since raw data preprocessing is not feasible in real-time TinyML algorithm deployment. So, incoming data are considered as biased or skewed incidental to this situation, leading to model performance degradation as the model is not adequately trained to make right predictions. Although there is a lot of work presented in the literature to prevent concept drift [70–72], there is still much room for improvement, since the complexity of tasks assigned to these kinds of devices increases.

- **Catastrophic forgetting**
  Making efforts to adjust incoming raw data to the existing knowledge, leading to the recognition of multiple tasks, in target devices can cause another problem, namely, *catastrophic forgetting*. Unlike humans' ability to learn more than one task and keep this knowledge in their memory, deep neural models execute some kind of "weight replacement," trying to fit to the new data they are exposed to. In [44], an approach to figure this problem out is demonstrated, based on recognizing which weights are the most significant ones from a probabilistic view. Furthermore, other approaches were also included, like in [73–77], specialized in either a specific network category, e.g., CNNs, or a type of learning scheme, like unsupervised learning. So, although the aforementioned annotated bibliography includes significant works for addressing this phenomenon, the actual challenge is to go further in finding ways for discovering efficient methods implementing few calculations to address this problem, since we are using resource-constrained devices, processing stream data in real-time scenarios.
- **Attacks.** ML/DL software methodologies suffer from attacks, aiming either at taking control over the neural network model and thus getting access to sensitive data or at *poisoning* the network – one of the most common attacks – by injecting deceitful data in it, in order for the model to export misleading results (white box attacks, backhaul attacks, etc.). Many approaches (adversarial ML) are already introduced in the existing literature, attempting to fortify DL algorithms, making the devices less susceptible to these attacks in [78]. From a hardware perspective, there are also vulnerabilities, according to [30], allowing Trojan viruses to invade the system. So, taking into consideration that the process of training takes place on the edge, there is still a lot of work that has to be done, so that DL models are secure and trustworthy, regarding the task they execute.
- **Benchmarking.** The lack of datasets, frameworks, and tasks adjusted to TinyML technology specifications in order to compare TinyML implementation feasibility for deployment is a serious deficiency that has to be mitigated, since this situation holds the scientific community back in releasing complete reliable TinyML systems.
- **Compatible-to-TinyML ad hoc decentralized topologies**
  As processing of data is shifting towards the edge, we anticipate the design of purely decentralized topologies, making the participating devices fully autonomous. Edge processing of data (either collaboratively among one another or with each device performing its own task) reduces the latency of data transmission (from device to server/cloud and vice versa), while data safety is ensured as data themselves do not circulate in the network. However, fully decentralized schemes demand secure, efficient, and trustworthy communication among the nodes of the network.
- **Communication cost in decentralized FL**
  In the literature, there is a plethora of FL algorithms, mainly in centralized environments, in which resource-limited devices are able to cooperatively train a global model, exchanging parameters, e.g., gradients, among each other. By shift-

ing to ad hoc peer-to-peer decentralized networks, there is not a central server coordinating the communication of devices, but devices themselves are responsible for keeping each other up-to-date, integrating the updated global model. So, the main "bottleneck" is the communication cost. Is it necessary to implement all-to-all communication? Is there an efficient way of electing which node can take place in global training? In Section 10.5, we briefly summarized some insights and demonstrated specific results regarding node participation protocols based on data similarity. However, the exploration of new efficient ways of solving this shortcoming in this field has yet to complete. Finally, is there a solution to reduce information diffused to the network, even more simple and less memory-expensive than model gradients? Purely decentralized learning schemes are an upsurging field, which the scientific community has to attach importance to.

10.7 Conclusions

All things considered, the necessity of developing new ML/DL techniques, applicable to edge resource-scarce devices, e.g., sensors, MCUs, etc., in networks like IoT, led to *TinyML* methodology, which reinforces device independence, regarding the processing of their own data (*decentralization of data*). In the existing literature, some static TinyML approaches have emerged, proposing inference-based methodologies, using ultracompact ML/DL models (see Section 10.3) so as to fit to these devices. However, the inference policy is not able to address the problem of heterogeneity of raw data, and this is the reason why *non-static TinyML* was introduced, which enables real-time training on the aforementioned devices. In order to perform on-device training, both the number of model parameters and the memory footprint of the neural network models have to be reduced during training, so *pruning* methods are used. The present chapter emphasizes the *sparsification* mechanism of a deep neural network, using network science concepts, like scale-free networks. Since the *transfer learning* (TL) mechanism bridges the gap between the random weight initialization and the all-layers training of a neural network, a new efficient TL method is also introduced (see Section 10.4).

Moreover, purely decentralized ML/DL methods have already appeared, in which node clustering protocols, *federated learning* (FL) schemes, or both are proposed in order to meet the resource specifications of this type of network. As extensively introduced in Section 10.5, an FL mechanism is proposed, accompanied by a promising node participation protocol in which nodes with similar data are discarded using DFT during the federation process. As a result, decentralized topologies necessitate the development of suitable reliable communication network infrastructures or else the construction of *backbones* as described in Section 10.2.

Finally, despite the fact that existing ML/DL or decentralized FL methodologies have paved the way for efficient training on edge devices, there are shortcomings that have yet to be solved. Some of the most crucial challenges in both TinyML and ad hoc network circumference are described in Section 10.6, which the scientific community has to cope with, so as to keep up with the requirements mandated by the new era of technological evolution.

References

[1] Armin Moin, Moharram Challenger, Atta Badii, Stephan Günnemann, Supporting AI engineering on the IoT edge through model-driven TinyML, in: IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), 2022, pp. 884–893.
[2] Lachit Dutta, Swapna Bharali, TinyML meets IoT: A comprehensive survey, Internet of Things 16 (2021) 100461.
[3] Partha Pratim Ray, A review on TinyML: State-of-the-art and prospects, Journal of King Saud University: Computer and Information Sciences 34 (2021) 1595–1623.
[4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, Software available from tensorflow.org, 2015.
[5] Visal Rajapakse, Ishan Karunanayake, Nadeem Ahmed, Intelligence at the extreme edge: A survey on reformable TinyML, ACM Computing Surveys 55 (13s) (2022) 282.
[6] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, Alexandra Peste, Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, Journal of Machine Learning Research 22 (2021) 241:1–241:124.
[7] Sinno Jialin Pan, Qiang Yang, A survey on transfer learning, IEEE Transactions on Knowledge and Data Engineering 22 (10) (2010) 1345–1359.
[8] Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson, How transferable are features in deep neural networks?, in: NeurIPS, 2014.
[9] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, Mohsen Guizani, A survey on federated learning: The journey from centralized to distributed on-site learning and beyond, IEEE Internet of Things Journal 8 (2021) 5476–5497.
[10] Frank Po-Chen Lin, Seyyedali Hosseinalipour, Sheikh Shams Azam, Christopher G. Brinton, Nicolò Michelusi, Semi-decentralized federated learning with cooperative D2D local model aggregations, IEEE Journal on Selected Areas in Communications 39 (2021) 3851–3869.
[11] S. Hosseinalipour, S.-S. Azam, C.G. Brinton, V. Michelusi, N. abd Aggrawal, D.J. Love, H. Dai, Multi-stage hybrid federated learning over large-scale D2D-enabled fog networks, IEEE/ACM Transactions on Networking 30 (4) (2022) 1569–1584.
[12] Yuchang Sun, Jiawei Shao, Yuyi Mao, Jessie Hui Wang, Jun Zhang, Semi-decentralized federated edge learning with data and device heterogeneity, IEEE Transactions on Network and Service Management 20 (2) (2021) 1487–1501.
[13] Mohammad Mohammadi Amiri, Deniz Gündüz, Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air, in: IEEE International Symposium on Information Theory (ISIT), 2019, pp. 1432–1436.

[14] S. Sundhar Ram, Angelia Nedić, Venugopal V. Veeravalli, Distributed stochastic subgradient projection algorithms for convex optimization, Journal of Optimization Theory and Applications 147 (2008) 516–545.

[15] Hong Xing, Osvaldo Simeone, Suzhi Bi, Decentralized federated learning via SGD over wireless D2D networks, in: IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), 2020, pp. 1–5.

[16] Stefano Savazzi, Monica Nicoli, Vittorio Rampa, Federated learning with cooperating devices: A consensus approach for massive IoT networks, IEEE Internet of Things Journal 7 (5) (2020) 4641–4654.

[17] Chenghao Hu, Jingyan Jiang, Zhi Wang, Decentralized federated learning: A segmented gossip approach, ArXiv:1908.07782 [abs], 2019.

[18] Nil Llisterri Giménez, Joan Miquel Solé, Felix Freitag, Embedded federated learning over a LoRa mesh network, Pervasive and Mobile Computing 93 (C) (2023).

[19] Nico Saputro, Kemal Akkaya, Suleyman Uludag, A survey of routing protocols for smart grid communications, Computer Networks 56 (2012) 2742–2771.

[20] Dimitrios Papakostas, Theodoros Kasidakis, Evangelia Fragkou, Dimitrios Katsaros, Backbones for internet of battlefield things, in: IEEE/IFIP 16th Annual Conference on Wireless On-demand Network Systems and Services Conference (WONS), 2021, pp. 1–8.

[21] Evangelia Fragkou, Dimitrios Papakostas, Theodoros Kasidakis, Dimitrios Katsaros, Multilayer backbones for internet of battlefield things, Future Internet 14 (2002) 186.

[22] Simone Disabato, Manuel Roveri, Incremental on-device tiny machine learning, in: Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things, 2020.

[23] Ranya Aloufi, Hamed Haddadi, David Boyle, Emotion filtering at the edge, in: Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems, 2019.

[24] Lennart Heim, Andreas Biri, Zhongnan Qu, Lothar Thiele, Measuring what really matters: Optimizing neural networks for TinyML, ArXiv:2104.10645 [abs], 2021.

[25] Jingtao Li, Runcong Kuang, Split federated learning on micro-controllers: A keyword spotting showcase, ArXiv:2210.01961 [abs], 2022.

[26] Josen Daniel De Leon, Rowel Atienza, Depth pruning with auxiliary networks for TinyML, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022, pp. 3963–3967.

[27] Colby R. Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, Paul N. Whatmough, MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers, ArXiv:2010.11267 [abs], 2021.

[28] Miguel de Prado, Manuele Rusci, Alessandro Capotondi, Romain Donze, Luca Benini, Nuria Pazos, Robustifying the deployment of TinyML models for autonomous mini-vehicles, Sensors 21 (4) (2021).

[29] Antonio Pullini, Davide Rossi, Igor Loi, Giuseppe Tagliavini, Luca Benini, Mr.Wolf: An energy-precision scalable parallel ultra low power soc for IoT edge processing, IEEE Journal of Solid-State Circuits 54 (2019) 1970–1981.

[30] Taiwo Samuel Ajani, Agbotiname Lucky Imoize, Aderemi A. Atayero, An overview of machine learning within embedded and mobile devices–optimizations and applications, Sensors 21 (2021) 4412.

[31] OpenAI, et al., GPT-4 Technical Report, ArXiv:2303.08774 [abs], 2021.

[32] Youssef Abadade, Anas Temouden, Hatim Bamoumen, Nabil Benamar, Yousra Chtouki, Abdelhakim Senhaji Hafid, A comprehensive survey on TinyML, IEEE Access 11 (july 2023) 96922–96922.

[33] P. Baldi, P. Sadowski, The dropout learning algorithm, Artificial Intelligence 210 (2014) 78–122.

[34] Aidan N. Gomez, Ivan Zhang, Kevin Swersky, Yarin Gal, Geoffrey E. Hinton, Learning sparse networks using targeted dropout, ArXiv:1905.13678 [abs], 2019.

[35] J. Frankle, M. Carbin, The lottery ticker hypothesis: Finding sparse, trainable neural networks, in: Proceedings of the International Conference on Learning Representations (ICLR), 2019.

[36] J. Diffenderfer, B. Kailkhura, Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted networks, in: Proceedings of the International Conference on Learning Representations (ICLR), 2021.

[37] Y. Kim, Y. Li, H. Park, Y. Venkatesha, R. Yin, P. Panda, Lottery ticket hypothesis for spiking neural networks, in: Proceedings of the European Conference on Computer Vision (ECCV), 2022.

[38] M. Yao, Y. Chou, G. Zhao, X. Zheng, Y. Tian, B. Xu, G. Li, Probabilistic modeling: Proving the lottery ticket hypothesis in spiking neural network, https://arxiv.org/pdf/2305.12148, 2023.

[39] Yu Xie, Qiang Sun, Yanwei Fu, Exploring lottery ticket hypothesis in few-shot learning, Neurocomputing 550 (2023).

[40] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, Antonio Liotta, Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science, Nature Communications 9 (1) (Jun 2018).

[41] Xiaolong Ma, Minghai Qin, Fei Sun, Zejiang Hou, Kun Yuan, Yi Xu, Yanzhi Wang, Yen kuang Chen, Rong Jin, Yuan Xie, Effective model sparsification by scheduled grow-and-prune methods, ArXiv:2106.09857 [abs], 2021.

[42] Evangelia Fragkou, Marianna Koultouki, Dimitrios Katsaros, Model reduction of feed forward neural networks for resource-constrained devices, Applied Intelligence 53 (2023) 14102–14127.

[43] Luciana Cavallaro, Ovidiu Bagdasar, Pasquale De Meo, Giacomo Fiumara, Antonio Liotta, Artificial neural networks training acceleration through network science strategies, Soft Computing 24 (2020) 17787–17795.

[44] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, Raia Hadsell, Overcoming catastrophic forgetting in neural networks, Proceedings of the National Academy of Sciences 114 (13) (Mar 2017) 3521–3526.

[45] A.-L. Barabasi, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.

[46] A.-L. Barabasi, Network Science, Cambridge University Press, 2016.

[47] Duncan J. Watts, Steven H. Strogatz, Collective dynamics of ‘small-world’ networks, Nature 393 (1998) 440–442.

[48] Q. Abbas, F. Ahmad, M. Imran, Variable learning rate based modification in backpropagation algorithm (MBPA) of artificial neural network for data classification, Science International 28 (3) (2016) 2369–2380.

[49] Andreas Chouliaras, Evangelia Fragkou, Dimitrios Katsaros, Feed forward neural network sparsification with dynamic pruning, in: Proceedings of the 25th Pan-Hellenic Conference on Informatics, 2021, pp. 12–17.

[50] Zi-Quan Hong, Jing-Yu Yang, Optimal discriminant plane for a small number of samples and design method of classifier on the plane, Pattern Recognition 24 (1991) 317–324.

[51] S.A. Nene, S.K. Nayar, H. Murase, Columbia Object Image Library (COIL-20), Technical Report CUCS-006-96, Columbia University, 1996.

[52] Han Xiao, Kashif Rasul, Roland Vollgraf, Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms, ArXiv:1708.07747 [abs], 2017.

[53] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton, Learning multiple layers of features from tiny images, Technical Report, Computer Science Department, University of Toronto, 2009.

[54] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, Fei-Fei Li, ImageNet: A large-scale hierarchical image database, in: IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255.

[55] Puneet Bansal, Image classification, Intel, https://www.kaggle.com/datasets/puneet6060/intel-image-classification, 2019.

[56] Song Chen, Beijing PM2.5, UCI, Machine Learning Repository, https://doi.org/10.24432/C5JS49, 2017.

[57] Kavya Kopparapu, Eric Lin, TinyFedTL: Federated transfer learning on tiny devices, ArXiv:2110.01107 [abs], 2021.

[58] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, Qing He, A comprehensive survey on transfer learning, Proceedings of the IEEE 109 (1) (2021) 43–76.

[59] Oscar Day, Taghi M. Khoshgoftaar, A survey on heterogeneous transfer learning, Journal of Big Data 4 (2017) 1–42.

[60] Evangelia Fragkou, Vasileios Lygnos, Dimitrios Katsaros, Transfer learning for convolutional neural networks in tiny deep learning environments, in: Proceedings of the 26th Pan-Hellenic Conference on Informatics, 2022, pp. 145–150.

[61] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Aguera y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), 2017, pp. 1273–1282.

[62] M. Ficco, A. Guerriero, E. Milite, F. Palmieri, R. Pietrantuono, S. Russo, Federated learning for IoT devices: Enhancing TinyML with on-board training, Information Fusion 104 (2023).

[63] J. Wang, A.K. Sahu, Z. Yang, G. Joshi, S. Kar, MATCHA: Speeding up decentralized SGD via matching decomposition sampling, in: Proceedings of the Indian Control Conference (ICC), 2019, pp. 299–300.

[64] A. Taya, T. Nishio, M. Morikura, K. Yamamoto, Decentralized and model-free federated learning: Consensus-based distillation in function space, IEEE Transactions on Signal and Information Processing over Networks 8 (2022) 799–814.

[65] F.P.-C. Lin, S. Hosseinalipour, S.S. Azam, Federated learning beyond the star: Local D2D model consensus with global cluster sampling, in: Proceedings of the IEEE Global Communications Conference (GLOBECOM), 2021.

[66] A.D. Amis, R. Prakash, T.H.P. Vuong, D.T. Huynh, Max-min *d*-cluster formation in wireless ad hoc networks, in: Proceedings of the IEEE Computer Communications Conference (INFOCOM), 2000, pp. 32–41.

[67] Meghana Nasre, Matteo Pontecorvi, Vijaya Ramachandran, Betweenness centrality - Incremental and faster, ArXiv:1311.2147 [abs], 2013.

[68] R. Agrawal, C. Faloutsos, A.N. Swami, Efficient similarity search in sequence databases, in: Proceedings of the International Conference on Foundations of Data Organization and Algorithms (FODO), 1993.

[69] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, Fast subsequence matching in time-series databases, Technical Report CS-TR-3190, University of Maryland, 1993.

[70] Yujing Chen, Zheng Chai, Yue Cheng, Huzefa Rangwala, Asynchronous federated learning for sensor data with concept drift, in: IEEE International Conference on Big Data (Big Data), 2021, pp. 4822–4831.

[71] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, Guangquan Zhang, Learning under concept drift: A review, IEEE Transactions on Knowledge and Data Engineering 31 (12) (2019) 2346–2363.

[72] Simone Disabato, Manuel Roveri, Tiny machine learning for concept drift, ArXiv:2107.14759 [abs], 2021.

[73] Peter Jedlicka, Matús Tomko, Anthony V. Robins, Wickliffe C. Abraham, Contributions by metaplasticity to solving the catastrophic forgetting problem, Trends in Neurosciences 45 (2022) 656–666.

[74] Kibok Lee, Kimin Lee, Jinwoo Shin, Honglak Lee, Overcoming catastrophic forgetting with unlabeled data in the wild, in: IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 312–321.

[75] Abel S. Zacarias, Luís A. Alexandre, Overcoming catastrophic forgetting in convolutional neural networks by selective network augmentation, in: IAPR International Workshop on Artificial Neural Networks in Pattern Recognition, 2018.

[76] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, Byoung-Tak Zhang, Overcoming catastrophic forgetting by incremental moment matching, ArXiv:1703.08475 [abs], 2017.

[77] Irene Muñoz-Martín, Stefano Bianchi, Giacomo Pedretti, Octavian Melnic, Stefano Ambrogio, Daniele Ielmini, Unsupervised learning to overcome catastrophic forgetting in neural networks, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits 5 (2019) 58–66.

[78] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, Debdeep Mukhopadhyay, Adversarial attacks and defences: A survey, ArXiv:1810.00069 [abs], 2018.