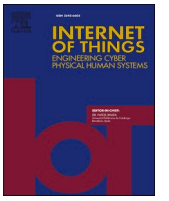
Internet of Things 16 (2021) 100461

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.sciencedirect.com/journal/internet-of-things



TinyML Meets IoT: A Comprehensive Survey

Dr. Lachit Dutta, Assistant Professor [1,*], Swapna Bharali, Assistant Professor [2]

[1] Dept. of ETE Barak Valley Engineering College, Nirala, Karimganj, Assam 788701, India
[2] Dept. of ETE, Barak Valley Engineering College, Nirala, Karimganj, Assam 788701, India

ARTICLE INFO

Index Terms:
Internet of Things (IoT)
tiny machine learning (TinyML)
hardware-software co-design

ABSTRACT

The rapid growth in miniaturization of low-power embedded devices and advancement in the optimization of machine learning (ML) algorithms have opened up a new prospect of the Internet of Things (IoT), tiny machine learning (TinyML), which calls for implementing the ML algorithm within the IoT device. TinyML framework in IoT is aimed to provide low latency, effective bandwidth utilization, strengthen data safety, enhance privacy, and reduce cost. Its ability to empower the IoT device to reliably function without consistent access to the cloud services while delivering accurate ML services makes it a promising option for IoT applications seeking cost-effective solutions. Especially in settings where inadequate connectivity is common, TinyML aims to provide on-premise analytics which will add substantial benefit to IoT services. In this article, we introduce the definition of TinyML and provide background information on diverse related technologies stating their strengths and weaknesses. We then show how TinyML-as-a-service is implemented through efficient hardware-software co-design. This article also introduces the role of 5G in TinyML-IoT scenario. Furthermore, it touches on the recent progress in TinyML research in both academia and industry along with future challenges and opportunities. We believe that this review will serve as an information cornerstone for the IoT research community and pave the way for further research in this direction.

1.1. INTRODUCTION

Over the decades there has been tremendous research focus dedicated to meliorate embedded technologies for use in resource-limited environments. The development of an affordable setting that incorporates a miniaturized embedded system is the key strategy that provides real-time solutions for many complicated real-world applications. In this regard, the microcontroller unit (MCU) based embedded systems have garnered tremendous attention, primarily due to the low power requirement and the recent development of tiny ant-sized microcontrollers [1]. Moreover, MCU possesses crucial performance and reliability traits such as safety, security, dependability, maintainability, adaptability, and acceptability [2]. MCUs have thus become ubiquitous and their influence on research is booming rapidly.

MCUs also play a central role as an extreme edge device across the IoT ecosystem from collecting sensor data and actuation, to transfer and reception of information [3]. The IoT device i.e. the MCU integrated with the sensors in the IoT network generally outsources the generated data to the cloud. Appending cloud in IoT is a delineated research area, in which the cloud acts as a brain to aggregate and draw insights from the data in order to provide services like ML to make intelligent decisions [4]. Outsourcing data to the cloud poses the following challenges, and hence remains an active area of research. Data processing at the edge is believed to be

* Corresponding author.
E-mail addresses: lachitdutta@gmail.com (Dr.L. Dutta), swapnabharalieng@gmail.com (S. Bharali).

https://doi.org/10.1016/j.iot.2021.100461
Received 10 July 2021; Received in revised form 27 September 2021; Accepted 27 September 2021
Available online 5 October 2021

more efficient than at the cloud because data is generated at the edge of the network [5]. Despite the enormous computing power on the cloud, executing the entire computing task on the cloud will demand large bandwidth, which is causative of IoT having large latency [4, 5]. Moreover, frequent access to the cloud increases the risk of privacy and also has serious concerns related to the autonomy of the edge device [4, 5]. On the contrary, the introduction of technologies such as cloudlet [6], micro datacenter [7], fog computing [8], and mobile edge computing [5] promises to handle data more efficiently, however, the IoT device still remains passive as decision making and executing ML algorithms are restricted to edge and cloud. Typically tiny MCUs find their use at the extreme edge of the IoT framework; these devices are integrated along with the sensors for laying the foundation of smart homes [9], smart agriculture [10], smart city [11], as well as safety-critical systems such as health care and vehicular technology [12, 13]. Furthermore, in an IoT framework, the MCUs are typically dumb and are endorsed only to transfer and receive information. The lack of an intelligent supporting mechanism near the sensor system is a computational bottleneck and inherent limitation of such a setting. Therefore, although tiny MCUs have galvanized IoT research, they are not yet exploited to realize its full potential.

During the last decade notable achievements were done in the field of ML: deep learning (DL) [14], neural networks (NNs) [15], and reinforced learning [16], etc., but their use was limited to GPUs, data-centers, and supercomputers only. Regrettably, the uses of these algorithms in MCUs were previously left unattended greatly due to their limited resources. A major factor restricting the integration of ML algorithms in MCUs is the difficulty in bridging between the large sizes of ML algorithms with those of smaller memory sizes of MCUs.

As the data generated by the IoT devices continues to rise, it is essential to identify and invest in IoT solutions that will alleviate the burden of frequent access and transfer of data to the cloud. A remedial measure to circumvent these drawbacks is to integrate TinyML on IoT devices [17]. TinyML in MCUs within the IoT device is a framework that obviates the need for frequently accessing the already overburdened cloud services. Therefore, in the current scenario ML services in IoT can be acquired by means of either *'serving ML to IoT devices'* or *'processing ML within IoT device'* [18]. The former approach is the traditional approach where the IoT based system relies on the edge and cloud to provide the ML services [18]. On the other hand, the latter method is a recent framework that uses TinyML to make IoT devices intelligent which implies that the MCU interfaced with the sensors can locally execute ML-algorithm to predict on sensor data. Although integrating ML in MCUs are relatively underexploited, early successes incorporating TinyML framework within IoT devices indicates that the field has significant potential to execute ML at the deepest IoT edge possible. TinyML is believed to be the next frontier of the hyper-digitization.

It is estimated that by 2023, the demand in the edge computing market will reach 1.12 trillion dollars globally [19]. To mitigate this burden, improvements in IoT layers along with a special emphasis on ML within IoT devices must be actively pursued. Moreover, the response of the global research community to the growing demand for intelligent devices has been remarkable, with companies like Ericsson already providing TinyML-as-a-Service solutions [18]. We believed that TinyML will play a major role in the near future in providing intelligent IoT solutions worldwide, with minimal requirement of access to cloud services. It will also revolutionize the use of IoT services in constrained environments with limited internet connectivity.

It is worth indicating the differences between this work and the work presented in [17]. The review work presented in [17] is carried out in the context of frugal objects where the detailed analysis of TinyML workflow is not considered. In contrast, in this paper, we provide the development history of TinyML and review the existing research work systematically to offer explicit guidance for researchers. Further, to fill this gap we have summarized the benefits of TinyML along with the wide contribution from academia and industry in hope of helping the scholars to provide a better understanding of the concepts of TinyML. The major contributions of this review article are to:

- Present the key performance indicators of TinyML framework along with its definition and overview; review of related technologies.
- Establish an intrinsic link between traditional ML implemented in MCUs with that of TinyML; detail the significance of hardware-software co-design to develop efficient TinyML frameworks.
- Review on TinyML research undertaken by various research groups in both academia and industry; discuss the feasibility of such systems in always-on battery-powered IoT devices, and finally, layout a roadmap for the future direction of research by citing out the various challenges and opportunities.
- Indicate the role of 5G in TinyML research.

1.2. TinyML Overview

The data processing and ML algorithms build on the cloud yield accurate prediction, nevertheless, they, in general, incur a high computational burden due to the need to handle a large amount of raw data generated by each IoT device [4]. Additionally, the speed of data transportation from the edge is also becoming one of the bottlenecks of cloud-based systems [5]. Therefore, the development of affordable and robust ML algorithms for use in resource-constrained IoT devices will outweigh the possible downside of cloud computing. Shifting ML algorithms to the IoT devices would create a plethora of opportunities, but along with it, new challenges will come into being. In this section, we present some affirmation why TinyML is more efficient than cloud computing for certain use-cases, and then we detail the foundation and our understating of TinyML.

1.2.1. Review of Related Technologies

Thus far, some specialized techniques have been introduced in the literature to address the challenges that are hailed from cloud-

based systems by adopting alternative mechanisms for data analysis. The techniques are mobile cloud computing (MCC), local cloud (LC), cloudlet, fog computing, and mobile edge computing (MEC).

The MCC is a resource management technique engineered to assimilate the advantages of mobile computing, cloud computing, and mobile internet [20-21]. With cloud servers located far away from end devices, the MCC structure tracks the requirement of resources and provides on-demand services. Although such systems can deliver high accuracy, they use a centralized approach similar to the cloud and possess the limitations of the cloud paradigm. LC is another tool commonly used in specialized local network centers together with cloud services for a group or institution [22]. The synchronization between LC and cloud server is achieved by dint of a software program running on the local server. LC is expected to provide better communication performance with high privacy features, however, due to sparse resources in the LC they suffer from computational limitations [23]. In contrast, cloudlet technology provides better services in computation intense environments. Cloudlet, characterized by a data-center positioned in form of a small-box, is essentially installed at a single wireless hop away from mobile devices [7]. This technology is successfully employed in public places like a community center, workplace, and shopping mall, etc. [7], where a number of multi-core computers are combined to form a cloudlet. Compared with cloud computing, the cloudlet based approach could attain overall ML services closer to the end device, which leads to low latency and reduced energy consumption. The main drawback of this approach is that each cloudlet module depends upon uninterrupted internet connectivity to provide its services, moreover, it has some serious concerns related to security and privacy [24]. Recently, fog (edge) computing has been used to transport the cloud services to the local area network where the intelligence is processed at the IoT gateway or a fog node that lies in the vicinity of the extreme edge device. In a fog network, each IoT devices is usually first registered onto a single gateway, through which multi-sensor data can be extracted and processed accordingly. Even if, fog computing provides low latency, the significant downside of this method is that the wireless connection between IoT device and gateway must always be active in order to perform complex operations [5]. Therefore, a radio access network (RAN) based method, namely, the MEC has been introduced, which aims to counteract this effect by means of pushing the computing and intelligence to the RAN [5]. Unlike fog computing, MEC provides built-in IT services locally through virtual servers using a part of a cellular network communication system [25]. Owing to its success, the MEC technology received an inordinate amount of attention from the IoT community [26-28]. However, the relevance of high computing virtual servers depends heavily on the availability of cellular network communication system to prepare and process information effectively. Standard edge hardware like edge TPU by Google, Jetson Nano by NVIDIA, Movidius by Intel provides a better solution by providing execution at the network edge [29]. However, edge computing suffers from privacy, latency, reliability, and energy efficiency.

A limitation to all the above approaches is their deployment of ML algorithms away from the IoT device. The contrasting features of the technologies described above are collocated in Table I.

1.2.2. Definition of TinyML

TinyML is an up-and-coming concept which deals with executing optimized ML models on ultra-low-power (<1mW) MCUs with minimal power consumption [30]. Particularly relevant for low-resource settings where access to traditional IoT network is limited, TinyML can become a valuable tool to enhance processing capabilities, especially as the data processing and ML services within the device bolsters the ability of the device.

Further, by adopting TinyML each IoT device becomes intelligent, it offers devices the ability to analyze data at the extreme edge, accelerating decision making and the paradigm can also be used in various use cases to provide standalone ML services, bypassing the need for IoT architecture. Due to its potential to execute ML models in a low-cost resource-constrained environment TinyML has grasped a lot of attention from both academia and industry. Research experts from big technology companies and academics have collaborated and are working towards the development of TinyML solutions. This also led to the initiation of the TinyML community in 2019 whose goal is to develop systems, hardware, algorithms, software, and applications for the TinyML framework [31]. Moreover, the community hosts the TinyML summit annually to showcase the whole stack of technological advancement in the particular field. The TinyML workflow in IoT environment is illustrated in Fig. 1.

It is worth mentioning, that the goal of TinyML is not to replace cloud computing completely but to equip the endpoint device with

TABLE I
Comparison Among Various Technologies.

Technologies	Structure	Latency	Bandwidth Requirement	Security and Privacy	Reliability	Computation Capability
Cloud/MCC	Fully Centralized	Very High	Very High	Very Low	Depends on the uninterrupted internet connectivity	Very High
Local Cloudlet	Partly Centralized	High	High	Low	Depend upon the local network connectivity	Limited
Cloudlet	Partly Centralized	High	High	Very Low	Depends on the robust and uninterrupted internet	Very High
Edge/Fog	Not Centralized	Low	Low	High	Depends on the always active wireless communication	High
MEC	Not Centralized	Low	Low	High	Depends upon the cellular connectivity	High
TinyML	May Operate Independently	Very Low (Near Zero)	Very Low to Nil	Very High	High as very less communication	Low

certain computing capabilities [32]. To achieve TinyML services, one can couple multiple IoT devices, where each device performs its own analytics separately as a component in a larger system and regulate the information to be passed down to the cloud. Another approach is to isolate the devices from the IoT ecosystem by assuming that each model works separately without the need for other IoT layers.

1.2.3. TinyML Benifits

In TinyML the computing is done at the proximity of sensors which can enhance or introduce new methods of data analysis previously unavailable in low-resource settings. The inherent features like versatility, cost-effectiveness, and simplicity of TinyML framework warrant attention for their potential to transform the whole IoT ecosystem. The key performance indicators that certify TinyML as an effective and essential tool can be characterized as follows [33, 34].

a. Change from dumb to intelligent IoT devices

The generation of huge amount of raw data by the sensor system has restricted alongside it the ability of the cloud computing to handle these data. Therefore, the majority of data are wasted at the edge without transmission to the cloud. TinyML provides an opportunity for analysis of these data in resource-limited setting where each IoT devices are made intelligent by incorporating ML algorithms within them. For instance, in Boeing 787 every second 5 GB of data is generated [35] likewise in a smart vehicle 1 GB of data is generated every second [36]. Therefore, instead of transmitting the whole raw data to the cloud, primary investigation on the raw data using ML algorithms at each IoT device to grasp the delicate information and obliterate the junk data would be more beneficial.

b. Network Bandwidth

The traditional IoT requires a sensor to gateway network, gateways, gateways to internet network, and cloud services for data ingestion, information processing, and running ML algorithms. However, innovative approaches to TinyML in resource-limited settings have the potential to redefine these requirements reducing the omnipresence of the cloud and making other services of IoT layers optional.

Compared to standard IoT services TinyML provides more independence allowing for low transmission and potentially less bandwidth. Further, the bandwidth requirement for transmission of raw data in a framework with dense IoT devices is significantly high. Therefore, prior analysis of the raw data at the edge and limiting the transmission to only important metadata would significantly reduce the bandwidth requirement. This can be further exemplified by the instances in Boeing 787 and smart vehicle put forward earlier.

c. Security and Privacy

Data security is a factor that negatively affects IoT acceptance, as large chunks of private data are transmitted to the cloud. It remains unclear to the end user where their personal information are stored and who owns the data when third party vendors are used for IoT services. Moreover, transmitting data opens up the possibility of intercept by malicious sources. Therefore, avoiding dataflow and keeping data confined within the device improves security and privacy. In TinyML data does not travel (or travels less) so they are less exposed to attack. As a result, in TinyML data security and privacy features are ingrained by-default and by-design.

d. Latency

The chain of events in an IoT system begins with transfer of sensor data from the IoT devices to the cloud servers and eventually ends on reception of decision (prediction) computed at the cloud by the IoT devices. It is clear from this sequence of events that this approach creates a huge latency, hence requires analysis close to the device. An effective approach to this problem is to use TinyML. Further, in safety critical systems such as health care (like robotic surgery) and autonomous vehicle waiting for the cloud to make a

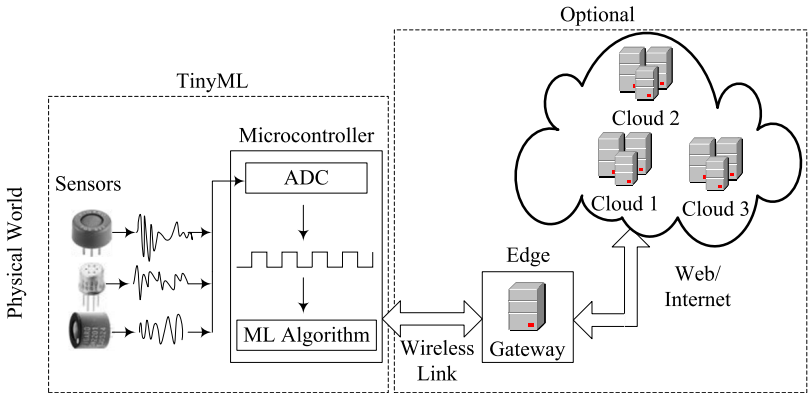


Fig. 1. TinyML workflow in IoT.

decision may have disastrous consequences. In such situations TinyML will serve as a backbone to provide reduced (almost near-zero) latency to provide ML services as there is lesser dependence on external communication.

e. Reliability

One key characteristic that is highly desirable of IoT is the ability to perform data-driven computation within the sensor network. TinyML has been identified as a solution to perform task on-premises in places like in open sea and rural areas where cellular connectivity/internet is very limited. This alteration in turn increases the reliability of IoT services.

f. Energy Efficiency

It is another important and popular indicator of TinyML in MCUs. In an IoT ecosystem most of the IoT devices are always-on and battery-operated. Typically these devices are powered by a coin battery (like CR2032), whereby IoT devices should perform its operation up to several months and sometimes years. Therefore, it is often required that the device mostly remains in sleep state, where the MCU can observe the data and wake up, based on its necessity. Moreover, in certain cases transmitting data may consume more energy than locally providing ML services. To address these challenges TinyML would prove to be a beneficial tool.

g. Potential Crash

Short time inoperability (STI) is a commonly problem that occurs due to interrupt in cloud services or crash in any IoT layer. This STI may lead to potential crash of the IoT structure. If acceptable level of inoperability is not achieved, the chance of the IoT surviving in risk-critical environment is very low, irrespective of their readiness. For instance, in use cases like smart agriculture and smart home STI might not have any severe effect while in risk-critical environment it is sufficient to bring about severe consequences. Using TinyML in IoT, prevention of STI would be possible by keeping analytics within the IoT device.

h. Data filtration

Intelligence in IoT device allows the designer to analyze the data and filter out the residues. Substantial advantage by designing of intelligent support system at the extreme edge can be provided by implementing in heavy traffic use cases in comparison with conventional systems. For example, a surveillance system engaged to detect anomalies consists of several cameras in which most of the information captured by the cameras are redundant. In such situations, filtering out the redundant images within the device rather than transmitting everything to the cloud would be more meaningful.

i. Low Cost

Reducing the data traffic reduces the bandwidth requirement which on the other hand reduces the cost.

To this end, we deduce that TinyML solutions combined with cloud technology can enhance the capacity of traditional cloud services, offer new avenues for data processing, or task-shift roles to MCUs to improve the aforementioned performance indicators. So, TinyML is the new trend and has been heralded by many as the technology that will empower the Industry 4.0 revolution or the digital revolution of industries [37].

1.2.4. Inception of TinyML

Unraveling the various hardware-based solutions used for implementing ML algorithms in the past is of paramount importance to understand the technological advancements that led to the development of TinyML. In the literature, an attempt has been made by various researchers to develop standalone embedded devices running ML algorithms (SEDRMA). Typically, the ML models are trained on a computer to determine the model parameters, which are then used for coding the ML model in the MCU. At first, research on the development of SEDRMA concentrated mainly on using digital ICs [38] and modular neural ring co-processor [39], but with little success as they wound up taking large space. These methods become less common in practice as it is almost impossible to design large NN architectures using them. Hence, MCUs which are programmable hardware platform has become a hotspot of ML research in embedded systems and a lot of related studies have been conducted.

While the ML model in PC will offer more computational resources, however, to make the system truly field-deployable, the ML models must be implemented in a resource-limited setting of embedded hardware (MCUs in particular). In [40], Farooq et al designed a hurdle avoidance system controller for a robotic car. In this system, feed-forward back propagation (FFBP) NN with a single hidden layer was implemented in AT89C52 MCU. Dutta et al. also integrated the FFBP structure in PIC 18F45K22 MCU to develop a field-type hand-held E-Nose for black tea classification [41]. Zhang et al. [42] introduced an ARM9 MCU-based PID controller to overcome the microbiological fermentation process using artificial NN. Further, there exists abundant research works in which ANN has been successfully tested in MCU for classification and prediction purposes [43]. Note that because these works are beyond the scope of this review work they are not extensively included. However, what the above investigational works all have in common is that they are specifically designed to execute ANN having a very small size. That is, these works do not have the explicit goal of deploying large ML models and are thus unable to offer target an appropriate output in certain cases.

Notable efforts in the recent past to improve the accuracy of ML algorithms have witnessed an exponential growth in their size. In this context, several DL, NN, reinforced learning, and clustering models, etc., have been proposed in the literature. These ML algorithms largely outperformed the traditional ML approaches in a wide variety of use cases [44]. In order to develop a more well-directed and efficient ML model, the primary requirements are high-fidelity, huge algorithm handling capability, and a large-scale well--annotated dataset. Unfortunately, due to the enormous size and complexity of the ML models, expensive GPUs are required to perform

the computation workload, as standard CPUs cannot handle huge ML architecture. However, the advent of cloud-based online platforms like software-as-a-service-instances (e.g. Google Colaboratory) and infrastructure-as-a-service (e.g. Amazon EC2 Instances), enabled the users' remote access to the computational power of the GPUs and to perform task-shifting. Cloud technology, in the last decade, has also been widely used in the context of the IoT framework [4]. However in the absence of intelligence at the IoT device, cloud-based methods present a significant risk to latency, power consumption, cost, data monitoring, analytics, and privacy [4, 5, 24-27]. Therefore, due to its safety, portability, and intelligence, MCUs executing ML have resurfaced as an intelligent edge device for IoT ecosystem in low resource settings. Another motivation behind this is due to the established efficacy of compressed ML algorithms used for the detection of audio wake words in smart-phones. Optimized ML algorithms originally engineered for keyword spotting in mobile-phones like '*Ok Google*' and '*Hey Siri*' have leveraged advances in ML for integration in MCUs [45]. Moreover, IoT devices, coupled with ML algorithms and proper training models, can change the landscape of IoT services in settings where the lack of network connectivity has created a bottleneck in IoT workflow. A major complicacy associated with implementing ML algorithms in MCUs is the high computational complexity of the algorithms. Further, implementing a huge ML model in a very small memory of MCU requires unprecedented changes in MCU hardware as well as the underlying ML structure.

1.2.5. *Hardware-Software Co-design*

The optimization of ML algorithm is a more challenging problem to overcome. Therefore, hardware and software needs to be appropriately tailored to develop frameworks that would alleviate these challenges. These two fields which were earlier not studied together have already caught many researchers' attention. Research and development of efficient hardware that can execute ML models effectively is of paramount importance but alongside it equal progress in software is required considering the limitation of resources in low power MCUs.

Due to the large complexity of the ML algorithm training the ML model will require a fast machine with an extremely large amount of memory. Further, the numerical precision of MCUs is very low, so it will be very difficult to build an accurate model. Although, some works in the literature have demonstrated the possibility of training the ML model in a resource-limited setting but training the model will consume very high power. Therefore, for battery-powered always-on devices (like IoT devices) the burden of training the network is generally left to the CPUs and cloud. Moreover, multiplication of tens and thousands of weights and execution of several non-linear functions (like log-sigmoid, tan-sigmoid), along with the need to maintain the fidelity of gradient computations will bring out additional challenges. Furthermore, frequent update in weights and structure of the ML algorithms occurs during the training phase, MCUs with limited memory will require longer duration for training the network and thereby consume enormous power. Thus, the tedious work of training the classifier is mostly done offline using a CPU or cloud services, and the exact mathematical model that represents the trained model is extracted. From the viewpoint of limited memory capacity in the MCU, the extracted model cannot be directly fed into it and therefore powerful optimization tools are used to trim down the precision of the ML model's internal representation to endow MCU with intelligence [46].

In high-resource settings, like cloud services, due to the on-demand availability of huge memory and computing power they can

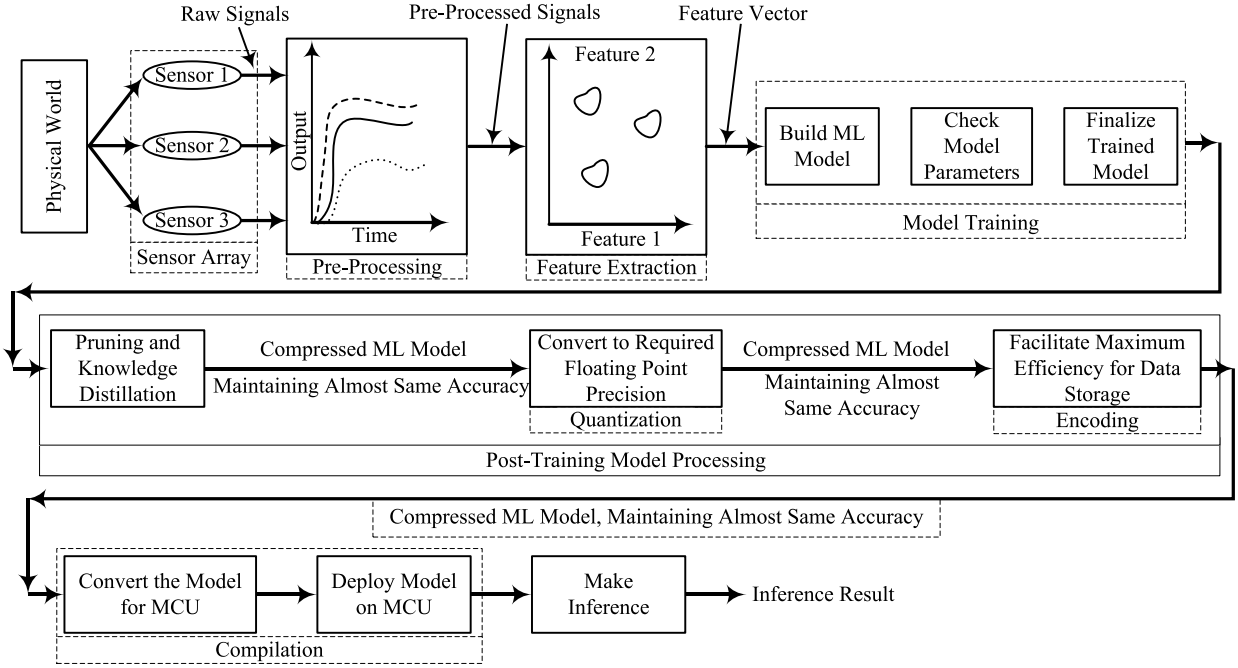


Fig. 2. TinyML architecture.

efficiently execute the massively parallel architecture of ML algorithms. In MCUs the resources are already limited, therefore deploying a ML model in a constrained environment presents significant challenges. Post-training the ML algorithm is crafted to create a model with a more compact representation to target MCUs using deep compression techniques. In doing so, the system developers have to sporadically make a decision whether to trade accuracy for low memory requirement or superior performance. As indicated in [47] trade-off is made amidst hardware to accuracy, model accuracy to connectivity, and connectivity to hardware. Whether the trade-off is within an acceptable limit depends on the accuracy of the targeted application, therefore, becomes complicated to develop a generalized optimization model. Several TinyML tools have been proposed to date including the following: deep quantization techniques [48], memory aware neural architecture searches [49], optimization frameworks (like TFL), and dedicated inference libraries [50, 51]. Among them, the deep compression method has become a hotspot of TinyML research and is elaborately discussed below. Fig. 2 shows the complete TinyML architecture, starting from data collection in the physical world to building the ML model in high-end processors and compressing the model to fit in MCUs for inference.

1.2.5.1. Methods and Procedures

In order to design a TinyML model the first step is data collection, depending upon the classification task data can be collected using sensors or we can use previously collected data from any standard database repositories. For instance, in [41] the authors have gathered the data from the output responses of 4 gas sensors to different variety of tea samples to develop a hand-held tea aroma estimation system; however, in [52] the authors have used an online database EEGS1 and EMGS2 in order to develop a microcontroller based system for early detection of ALS and Myopathy. Further, in image classification tasks preliminary investigation on benchmark datasets like ImageNet and CIFAR-100 can be conducted in order to test the efficacy of compressed data-driven algorithms. The raw signals or data thus acquired are pre-processed to extract features. The extracted features serve as inputs to the neural network for training and testing. The steps aforementioned are similar in any classification task. However, in TinyML, several compression techniques are employed in the network to create a model that can fit in MCUs. The steps that are required for deep compression of ML algorithms can be formulated as knowledge distillation, pruning, quantization, encoding and compilation.

Knowledge Distillation (KD): KD technique provides a unified framework to transfer knowledge acquired by a large pre-trained model (referred as teacher) to another model which is of very small size (student) [53]. The steps involved in KD are discussed here with the help of convolution neural network (CNN). Initially, a large CNN model (teacher) is trained until the desired accuracy level is obtained. Fig. 3 shows a typical CNN model with all the layers.

In the output layer of the CNN, the "softmax" activation function is used, which generates a probability distribution and assigns a probability for each class. The input to the softmax function is a vector Z (logits) consisting of K real numbers representing the number of classes, and it converts them into K probabilities. The standard softmax function σ: $***^K \rightarrow [0, 1]^K$ is represented as:

$$q_i = \sigma(Z)_i = \frac{e^{Z_i}}{\sum_{j=1}^{k} e^{Z_j}} \text{ for } i = 1, 2, \ldots\ldots, k \text{ and } Z = (Z_1, Z_2, \ldots\ldots Z_K) \in ***^K \tag{1}$$

The softmax produces one hot output label (hard target) or assigns the highest probability to a particular output, and other classes are assigned low values. For example, the MNIST dataset gives a probability distribution as [0.98, 0.02] to the output labels 2 and 7. It indicates the class label of the input to the CNN is the number 2; however, there is a possibility that the number is 7. The information that the image has some traits of number 7 is necessary to transfer to the student. But, to transfer the knowledge, the distribution should be more spread across the class labels. Making the distribution spread creates soft targets rather than one hot output. Hinton et al. [53] claimed that much of the information are retained in the soft targets rather than the hard targets. In order to demonstrate the efficacy of the soft targets, a new parameter temperature (T) is introduced in the softmax function as shown below [53]:

$$q_i = \frac{e^{Z_i/T}}{\sum_j e^{Z_j/T}} \tag{2}$$

When $T=1$, we get (1). Increasing T extends the spread of the class distribution. A high temperature is used in the teacher model to get a desirable spread from the modified softmax output. Next, a small-sized model is chosen as a student model (distilled model) and trained with soft targets at the same temperature.

At a specific temperature ($T>1$), estimate loss function using the Kullback-Leibler divergence function considering the probability

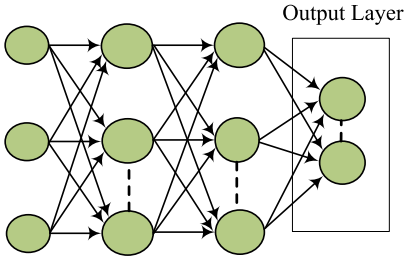


Fig. 3. CNN architecture.

distribution of the master (p_i) and the student (q_i) as [54]:

$$\text{Loss}_{\text{soft},T} = -\sum_i p_i \log(q_i) \tag{3}$$

Next, setting T=1 with the same logits and hard label y_j is obtained. The estimated cross-entropy for input x_j is [54]:

$$\text{Loss}_{\text{hard}} = -\log\left(q_{yj}, T = 1\right) \tag{4}$$

Using (3) and (4), determine the final loss as [54]:

$$\text{Loss}_{\alpha,T} = \alpha \text{Loss}_{\text{soft},T} + (1-\alpha)\text{Loss}_{\text{hard}} \quad \text{where, } 0 < \alpha < 1. \tag{5}$$

Where $\alpha = 1$ corresponds to training with only soft loss, conversely, $\alpha = 0$ means training performed on the true labels only. Training performed as above promises to provide a relatively small model that imitates a large model in terms of accuracy. More recently, various classification tasks successfully implemented KD. However, the introduction of a new hyper-parameter T increases the computational burden.

Pruning: The pruning technique is typically an iterative process (as illustrated in Fig. 4) used to convert a dense neural network into a sparse network. To create a pruned model, the network is initially trained using a conventional training approach. Subsequently, as suggested in [55] the important connections are determined by locating the weights which are above a certain threshold. Pruning is achieved by removing the number of weights that falls below the threshold limit. However, when testing is performed on the pruned network, it fails to deliver the same level of accuracy as the dense network.

The poor performance of the pruned network limits its applicability in classification tasks; however, retraining the remaining weights tends to recover the accuracy. Additionally, if pruning and retraining are done in an iterative fashion, the lost accuracy can be fully recovered even if 90% of the parameters are pruned.

It should be noted, that the Pruning technique not only helps in removing connections but also provides a systematic approach to prune neurons which has no input connections or zero output connections (Fig. 5) [55]. These neurons during retraining are automatically eliminated and hence termed as dead neurons.

It is demonstrated in [55] that pruning a network reduces the size of the network up to 13 times without loss of accuracy. Although pruning provides a compressed version of the original model, the time complexity in deducing the acceptable model is very high and labor-intensive.

Quantization: In pruning and KD, we get a fewer number of parameters because of which the model size reduces. To further downsize the model, quantization technique is applied through which the number of bits required to represent these parameters is reduced. Therefore after pruning and distillation, the next process is quantization in which the floating-point precision of the algorithm is reduced to match the architecture of the MCU [56]. For instance, weights of the ML models, which are typically of 32-bit or 64-bit, can be mapped down to 8-bit as well as others, to execute them on general purpose MCUs which are of 8-bit. To do so, the weights previously stored in the standard 32/64-bit floating point precision format $x \in [\alpha, \beta]$ are mapped to a lower bit-width (b-bit integers) $x_q \in [\alpha_q, \beta_q]$. *Int8* quantization is one of the popular methods in which 8-bit integers are used instead of floating-point numbers. Han et al. [57] demonstrated that a pruned network is further made small up to 31 times by implementing quantization without loss of accuracy. They also proposed a weight-sharing strategy using scalar quantization to reduce the memory footprint. In certain cases quantization is achieved by trading accuracy around 1-3% [58].

Encoding: It is an optional process that facilitates maximum efficiency for data storage and is at times implemented to further reduce the size of the ML model [57]. The most popular encoding scheme used in ML is the Huffman encoding, which is a variable-length lossless data compression technique. It encodes the more frequent weights with fewer bits and the infrequent weights by more number of bits. Huffman's encoding scheme can further reduce the network size up to 49 times maintaining the same level of accuracy. [57].

Compilation: Compilation is the final step, in this step an interpreter is used to convert the ML model designed using any language into a language (like embedded C) that is readily understandable to the MCU [45]. In general, interpreters like TFL are used to convert the scripts written on python into C++ output file, the generated output file is then burned in to the MCU using any cross-compiler.

As discussed earlier, KD [53] trains a compact network (the student model) to mimic the behavior of a high-capacity network (the teacher model). KD also incorporates a simple mechanism to reduce the network parameters compared to pruning [59]. Despite the significant performance, the student model requires a prior trained model to extract a distilled model. Furthermore, KD based method requires frequent human intervention and the task of matching students' performance at each level of iteration is difficult and time-consuming.

The extensions of the KD framework are briefly summarized herein. In [60], a super teacher-based KD was proposed. In this scheme, an ensemble of 16 CNN models were used as a teacher and the distillation model was extracted to design a shallow NN

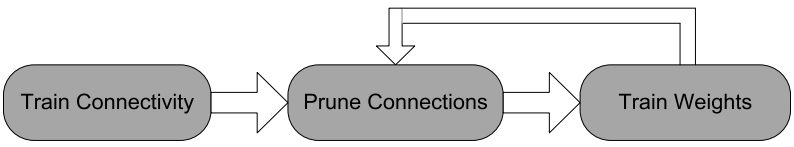


Fig. 4. Pruning process.

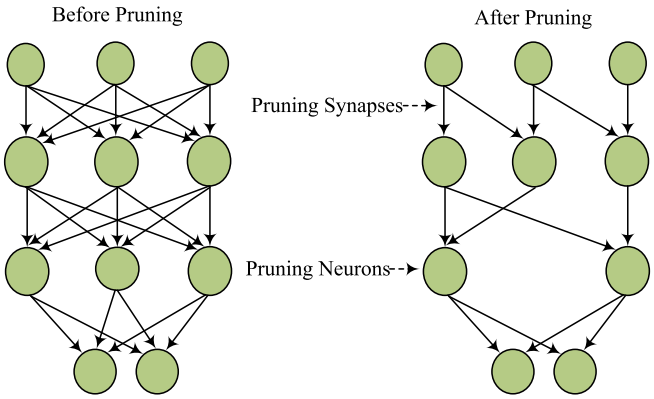


Fig. 5. Pruning synapses and neurons.

containing 1, 2, 3, 4, and 5 layers. However, the choice of a large student model is important so as to allow training to be successful. To preserve accuracy, the student model cannot be too shallow, too narrow, or miss any necessary unit [56]. In another work, Sobolev Training for the neural network was introduced [61]. This approach not only reduces the difference between the teacher prediction and the student prediction but also minimizes the derivatives of their respective loss function. Further, Furlanello et al. [62] proposed Born-Again Networks where a student having an identical architecture as the teacher is trained with two goals first to predict the correct output labels and second to match the output labels of the teacher. They observed that the student model significantly outperforms the teacher network. However, model compression is not achieved in their work.

Pioneering works on pruning [63, 64] based on the second derivative was proposed a few decades back. Han et al. [55] reintroduced this concept to reduce the size of image recognition NN. However, accuracy is achieved at the cost of higher training time as each retraining process involves several iterations. More recently, there has been growing interest in pruning as it compresses ML models to execute on resource-limited settings. Guo et al. [65] introduced the concept of dynamic network surgery by combining pruning and splicing. The splicing mechanism enables the recovery of pruned connections at any point in time once they are found useful. The authors conducted the test with LeNet-5 and AlexNet to validate the model and obtained a compression factor of 108, which is far superior to the typical pruning approach. Frankle et al. [59] speculate that a randomly initialized dense network consists of a sub-network that can be trained in isolation initialized with the parameters of the dense network to match the accuracy. However, randomly initializing the sub-network results in a high loss of accuracy. In contrast, Liu et al. [66] demonstrated that random initialization of weights could deliver the same level of accuracy by fine-tuning the learning rate. Furthermore, there are several extensions of pruning proposed in the literature, which, can be found in [59]. To determine the appropriate set of parameters and pruning rules is still an active research pursuit. In [58], a detailed analysis of the quantization process was reported. The illustrated results indicate that accuracy loss is due to weight quantization, whereas quantizing activations to 8-bits bears no loss. It was also demonstrated that by incorporating quantization-aware training, the model accuracy improves to a great extent. In quantization-aware training the weights and activations are quantized during the training process.

Pruning, KD, quantization, and encoding are state-of-the-art methods for implementing ML models in resource-limited settings. However, the developer has to deal with the cumbersome task to iteratively train and test the network individually in each step to reach an acceptable model and therefore slows down the model development process. The majority of these challenges are due to the lack of benchmark to deal with large volumes of data paired with difficulty in choosing an appropriate framework when it comes to compressing and incorporating ML in low-end devices.

1.2.5.2. Datasets and Results

As aforementioned in earlier days fitting neural in resource-constrained settings was limited to the shallow neural network. In doing so, a shallow NN is trained to extract the mathematical model, which is typically programmed in a microcontroller to perform prediction/classification. However, in advanced classification tasks like context recognition, audio wake words, keyword detection, visual wake words, object detection, etc., the shallow NN performs very poorly as it fails to capture good feature representation. The rapid advancement in ML algorithms and the arrival of DNN provided much more efficient results compared with shallow NN, due to which they are gaining popularity. But due to the inherent large size of DNNs, it requires enormous memory capacity for computation and storage, thus cannot be fit in devices with limited memory settings. Therefore, the realization of advanced ML algorithms, NNs, and DNNs was limited to high-end processors like CPUs, GPUs, data-centres, and supercomputers. However, the advent of TinyML made it possible to compress complex DNN models taking advantage of the deep compression techniques. It efficiently incorporates DNN models in devices with a limited memory footprint.

With the rise of TinyML, deep compression, in which the goal is to develop a miniature model of ML algorithm, is attracting attention from researchers. It is pertinent to mention that the objective of deep compression is not to enhance the accuracy but to maintain the same level of accuracy as the original model. Deep compression and its constituents have proven to be beneficial for a wide variety of data sets and use cases. Some of the significant works are discussed subsequently in detail.

In [53], a preliminary investigation was conducted on the MINST dataset applying KD on an ensemble of DNNs. The MNIST is basically a large database consisting of 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. The results obtained demonstrate the effectiveness of KD to compress the model size. Experiments were further conducted to support the findings on automatic speech recognition and Google's image dataset, JFT. Inspired by the study, Polino et al. [56] proposed model compression using both distillation and quantization on image datasets CIFAR10. Much earlier, McDonnell et al. in [67] used shallow CNN in the CIFAR10 dataset and obtained an accuracy of 75.86%. In [56], the teacher model was built using CNN architecture that consists of 5.3M parameters, having a size of 21MB, and has an accuracy of 89.1%. They developed three quantized students models (SMs): SM1, SM2 and SM3. SM1 achieved accuracy in the range 84.5%-88.8%, consists of 1M parameters and size of 4MB. Similarly, SM2 has a size of 1.27MB and yields an accuracy of 80.3%-84.3% with 0.3M parameters. SM3 has accuracy in the range 71.6%-78.2%, with 0.1M parameters, and requires 0.45MB storage capacity. Further, they corroborate and extend the study on the CIFAR100 dataset, OpenNMT integration dataset for German-English translation using LSTM, and ImageNet dataset using ResNet architecture. In another study [57], a three-stage deep compression pipeline was proposed: pruning, trained quantization, and Huffman coding. The proposed method was tested on AlexNet and VGG-16 architectures using the ImageNet dataset and obtained compression at each pipelining stage. In the first stage, pruning reduced the model size by $9\times$ to $13\times$. The reduced model is then processed using trained quantization which further downsized the model $27\times$ to $31\times$, and at the last stage, encoding squeezed the model in the range $35\times$ to $49\times$.

In contrast with the conventional ML algorithms that require off-chip DRAM for model storage, deep compression allows fitting the model even in resource-constrained devices with small size on-chip SRAM. Further, KD, pruning, and quantization allow layer-wise speed up of the model due to reduced structure, less number of connections, and reduced bit size. For instance, the deep compression strategy applied on AlexNet and VGG-16 architectures in [57] considering the ImageNet dataset obtained $35\times$ and $49\times$ less memory space. The downsizing resulted in a reduced memory footprint requirement from 240MB to 6.49MB in AlexNet and 552MB to 11.3MB in VGG-16. This technique not only facilitates the implementation of ML on resource-constrained devices with lower bandwidth but also decreases the inference time as well as energy consumption. Therefore, TinyML aims to exploit the concept of deep compression to integrate ML in low-end devices by making the ML itself tiny in size, require less bandwidth, layer-wise computation speed up and energy-efficient. However, the absence of benchmark and incomparable experimental settings makes it difficult to confidently compare the model compression techniques proposed over the years.

TinyML tries to replicate the full potential of DNN in resource-constrained devices like microcontrollers. However, the resource-constrained devices operate at low speed, have limited memory capacity, and can deal with limited data size, and therefore cannot be used to perform training and requires the help of other high-end devices. Further, data from new features/examples can be easily introduced in the DNN model in the high-end devices as they can adapt through retraining. In contrast, retraining the TinyML is complicated basically due to two main reasons. First, the low-memory device cannot perform training of its own, and second, the model trimmed for TinyML passes through various stages of fine-tuning and compression, consequently, retraining in TinyML is similar to building a new TinyML model.

1.2.5.3. TinyML Tools

Among recent advances in TinyML software tools, several frameworks have shown great promise for the advancement of ML research in MCUs. One of the key characteristics that attract TinyML research is that most of the tools and frameworks are open-source and not proprietary. Some such tools are summarized in Table II [45]. The comparison among the available ML architectures in

TABLE II
Popular TinyML Software Frameworks.

Framework	Developer	Specialization
TensorFlow Lite Micro (TFLM)	Google	i Open source. ii Flexible interpreter based solution. iii Developer can modify the library code. iV Designed for wide range of MCUs.
Embedded Learning Lbrary (ELL)	Microsoft	i Open source library. ii Cross-compiler tool chain similar to TFLM.
Graph Lowering (GLOW)	Research Group	i Open source compiler. ii Uses ahead-of-time compilation for both floating point and quantized arithmetic.
STM32 X-CUBE.AI	STMicroelectronics	i Optimized code for STM-32 series MCUs. ii Supports FP32 and quantized models. iii Built-in optimization.
TensorFlow-Native	Google	i Compiles TensorFlow graphs into C++ code. ii Quantization and optimization unavailable.
TinyEngine	Research Group	i Inference engine for MCUs. ii Compile based method. iii Removes memory overhead.
TVM	Research Group	i Open source DL compiler. ii Offers optimized models for any hardware.
uTensor	uTensor	i Specifically designed for ARM. ii Converts TF model into C++.

different frameworks is not done as these are ongoing works and each model has the potential to support different ML paradigms. If appropriate modifications in hardware and software continue to thrive, TinyML could play an increasingly important role in the IoT ecosystem.

Supplementing innovations in hardware by layering on task-specific, inference engines by virtue of novel architecture design [68] provides an opportunity for improved TinyML solutions in resource-limited settings by decreasing the power requirement and renovating inference on next-generation MCUs for rapid industrialization [69]. In this regard, to accelerate ML inference processors like Ethos-U55 and ECM 3532 has been launched recently by ARM [70] and Eta Computer [71] respectively. Additionally, hardware companies have released their own backend NN support libraries like CMSIS-NN, PULP, X-CUBE-AI, and PULP-NN, etc. [72], to support resource-limited settings of MCUs. Moreover, the integration of these libraries with optimization frameworks to fully exploit the hardware architecture of MCUs could provide a more robust hardware-software solution. However, the development of optimized hardware to meet the requirements of TinyML still remains a technical challenge.

1.3. Advancement in TinyML Research

As research and technology evolve, TinyML is finding its place as an adjunct tool in a wide spectrum of IoT applications. This section provides an overview of newly developed ML algorithms for TinyML, it also presents the varied alteration that is carried out in ML paradigms for integration in MCUs along with some of the well established TinyML use cases. However, the current landscape of TinyML lacks maturity due to the unavailability of a unified benchmarking scheme. Although, it impedes the systematic comparison and evaluation across different TinyML frameworks, the rich pool of methodologies proposed by various researchers has the ability to breed a solid technical basis on which a TinyML benchmarking strategy can stand. In this context, Banbury et al. [30] presented the challenges and direction regarding benchmarking TinyML. They pointed out that low power, limited memory capacity, hardware heterogeneity, and software heterogeneity are the key issues that are needed to be addressed in order to develop a suitable benchmark. In [17], detailed analysis of TinyML in the context of frugal smart objects is presented. In pursuance of accessing the efficacy of TinyML based algorithms for classification task on a synthetic dataset of 10,000 training samples using smart multi-radio access network, the performance of popular ML algorithms, namely, SVM, MLP, decision trees, and RF, in conjunction with Arduino Uno board were appraised for the same task. The outcome of the analysis asserts that the best performance was attained by decision trees and RF classifiers.

Prado et al. [73] deployed tiny-convolution neural networks (CNNs) (LeNet 5 CNN architecture) on three different ultra low power MCUs, viz., GAP8, STM 32 L476, and NXP K64f for autonomous navigation of mini vehicles. The STM platform integrated with CMISIS-NN was taken as a reference for comparison with GAP8 and NXP K64f integrated with PULP-NN. They observed that GAP8 reduces its latency by 20 $\times$ compared to STM 32 L476 or obtains 21.4% higher accuracy than NXP K64f when tested under similar conditions. Besides, for a single inference, the MCUs consumed barely 3.9 μJ in real-time. Similarly, in [74], YOLO Nano, a highly compact CNN paradigm for the task of object detection was proposed. The compressed network was exploited on Jetson AGX Xavier embedded module as a means to obtain good performance at a low cost. Experimental results showed that the computation cost of YOLO Nano was far below the popular frameworks (like, Tiny YOLOv2 and Tiny YOLOv3) used for compressing ML models for embedded object detection. In the recent past, algorithmic advancement has enabled competitive classification accuracy in CNN even when the weights during training was limited to binary (+1/-1) format. This made it possible for new optimization opportunities by negating the multiplication operations. Andri et al. in [75], used an accelerator optimized for binary weight CNNs known as YodaNN and achieved an overall energy and area efficiency of 61 Top/s/w@0.6V and 1.1 Top/s/MGE@1.2 V respectively compared to other recent accelerators. Moreover, Liberis et al. [76] designed a tool to reorder the operators in the NN model created using the TFL framework, which translates into reducing the memory footprint for building a compressed CNN model. The modified algorithm was capable of running on a MCU with only 512 KB of RAM on a NUCLEO-F767Z1 prototyping board.

To realize real-time DNN execution, Niu et al. [77] proposed an end-to-end framework, viz., PatDNN. The proposed framework outperforms the current state-of-the-art end-to-end DNN execution frameworks in terms of speed up to 44.5 $\times$ without affecting the accuracy. In [78], architecture for aggressive compression of MobileNets was proposed to replace the traditional multiplication of weight matrix in DNN with activations as a 2-layer sum-product network. Moreover, the design of the architecture permits the construction of a model with 46.4% and 51.07% reduction in multiplication and model size, while increasing addition to 48%. This resulted in a 28% saving in energy requirement per inference while ensuring no degradation in the DNN hardware accelerator. Wong et al. [79] introduced the notion of AttendNets, which is a low precision and highly compact DNN tailored for on-device image recognition. A comparative study on the ImageNet50-benchmark dataset was performed with state-of-the-art methodologies and it was observed that AttendNets obtained higher accuracy (~7.2%), requires low power (~4.17 $\times$), and very low memory (16.7 $\times$) of 728 KB.

In another work [80], the idea of attention condensers was introduced to build deep speech recognition NNs (TinySpeech) on the edge devices. TinySpeech was tested on the Google Speech Commands benchmark dataset and compared to previous works. The analysis showed that the model complexity, storage requirement, and computational cost were significantly reduced by 507 $\times$ fewer parameters, 2028 $\times$ lower weight memory and 48 $\times$ fewer multiply-add operations. In [81], a case study of four NN models LSTM, GRU, CNN-LSTM, and CNN-GRU was presented to determine the best model for environment prediction. To demonstrate the potential of the best model (LSTM), it was deployed on STM 32 MCU through model compression with the help of the X-CUBE-AI framework. The optimized model consumed 45.5 KB of ROM and 480 Bytes of ROM; it was also used to record the environment prediction for a span of 30 days which resulted in an RMSE of 2.10 hPa. In [82], Legendre Memory Unit (LMU) based RNN algorithm [83] was used to design an efficient hardware aware training (HAT) network for keyword spotting. They evaluated the network in ARM M4F and

observed an almost 14 × reduction in power usage than current state-of-the-art technologies.

In [73], an open source flexible CNN library, namely, CMix-NN has been introduced, for resource-constraint edge devices, which aims to offer an optimized backend to deploy mixed low-precision deep NNs on ARM CortexM processors. The proposed framework has been compared with other state-of-the-art 8-bit and FP32 models, considering ImageNet problem fitting 2 MB memory, and an increased accuracy of 8% and 23% was obtained. Further, Vuppalapati et al. [47] designed a cow necklace (using tiny IoT edge-PCB board) to demonstrate the possibility of small farmers in rural areas to join the data revolution and highlighted the importance on democratization of AI. In [84], a wearable sensor was designed and developed using nRF52480 MCU to detect precise concentration of alcohol. They used Google Colab to build the TinyML model with the help of TFL library.

Further, many successful applications of TinyML include integrating KB-sized ML models to tiny IoT devices [85], eating detection [86], parameter estimation of Li-Ion batteries [87], and medical face mask detection [88], etc. Besides, TinyML has also been extensively investigated in many areas, such as audio analysis (audio wake words, context recognition, control words, and keyword detection), image recognition (visual wake words, object detection, gesture recognition, object counting, and text recognition), psychological/behavioral metrics (segmentation, forecasting, and activity detection), and industry telemetry (sensing, anomaly detection, motor control, and predictive maintenance) which are briefly pointed out in [30].

Computer vision, sound recognition, and design of low power accurate ML models are the most sought after research area among TinyML community (see [89] and the works presented therein). This is probably due to the fact that preliminary research in these areas can present us with the usability and practicability aspects of TinyML. While the use of ML is currently limited in low resource settings, aforementioned works of ML integration into MCUs and development of efficient quantized models offer a pathway for increasing its use in running analytics close to the device. Indeed, applying TinyML in more complex ML applications, in order to understand their performance over complex terrains would be extremely beneficial. Apart from these technical breakthroughs, a number of community, industries, and websites have come forward to provide free resources for research practitioners to get a broad perceptive of TinyML. Some of such resources are detailed in Table III.

1.4. Role of Industries in TinyML Research

Evolution of IoT from computation in cloud to near the proximity of the sensor have shown many challenges, both in hardware and software design, and their fields of application. Some of the leading software companies, hardware companies, and service providers are engaged in TinyML research and development to make TinyML reach every IoT device. Table IV depicts the vision, challenges, and services along with present research activities by research laboratories around the world.

1.5. Role of 5G in TinyML and IOT

Here the authors would like to discuss another technological breakthrough that would change the communication among the various devices, 5G. A special role in the context of handling large data with low latency can be covered by 5G communication [110]. 5G will create a new hyper-connected world where everybody and everything will be connected. The union of ML, 5G, and TinyML is depicted in Fig. 6.

Accordingly, with the advent of 5G, new directions and challenges of TinyML will come into being. Here the authors would like to argue that although TinyML is designed to solve ML problems within the edge, however, the prediction from metadata or collective decision from thousands of devices would still be done in the cloud. Therefore, it would be beneficial to draw the line between device solvable problems and cloud solvable. Moreover, combining the advantage of both the technologies while avoiding the limitations of each will provide a more robust architecture. Scholars believe that 5G will play a crucial role in connecting these two frameworks more

TABLE III
TinyML Free Resources.

Resource Provider	Nature	Content	ML Model	MCU/Development Board	Software Used for Deployment in MCU
O'Reilly [90]	Book	Introduction and basic use cases	NNs	Arduino Nano 33BLE Sense, Sparkfun Edge, STM 32F746G Discovery kit	TensorFlow Light (TFL)
Digikey [91]	Tutorial	Implementing TinyML in Arduino	3 layer fully connected NN model	Nano 33BLE Sense	TFL
Hackster [92]	Tutorial	Easiest way to deploy TFL model	3 layer fully connected NN model	ESP 32	TFL
Edge Impulse/ARM Community [93]	Tutorial	Recognize sounds using only 23 KB RAM	Customized CNN structure	STM IoT NODE Discovery Kit Board	Edge impulse software tool
Arduino [94]	Tutorial	Cough detection using 20 KB RAM	CNN	Nano 33BLE Sense or Cortex-M4+ board	Edge impulse software tool
Github [95]	Tutorial	Gesture recognition	CNN	Nano 33BLE Sense	TFL
TinyML Community [96]	Tutorial	Mikro-Kernel based hardware accelerator	-	-	-
YouTube [97]	Tutorial	AutoML forting ML with once-for-all-network	Once-for-all-model	-	AutoML
CodeLabs [98]	Tutorial	AI based speech recognition	-	Spark fun Edge Board	TFL

TABLE IV
Vision of TinyML: Industrial Perspective.

Research Industries	Vision and Mission
ARM [99]	Arm has recently launched a new ML processor, Ethos-U55 for area constrained and IOT devices. ARM is actively carrying out research to: i Design MCU architecture central to IOT. ii Optimize tooling for CortexM hardware, Keil MDK, and its own IoT operating system, Mbed OS.
Google [100]	Google has come up with various frameworks (like TFL) to bridge the gap in implementing ML algorithms in embedded systems.
ARM and Qualcomm [99]	The prime focus of their partnership is to help developers with resources to build and deploy faster ML models in a more efficient way so that it can fit in any environment.
ARM and Google [99]	Arm and Google has been the global pioneering in development of TinyML solutions. They are working on efficient combination of ARM CMSIS-NN libraries with Google's TFL micro framework to take the full advantage of ARMs hardware optimization. Some of the key goals of their coalition in context of TinyML are: i ARM- Build hardware, tooling, and, design of compilers and libraries ii Google- Training and optimization of models to deliver: Hardware capable of learning.Compact ML models.Software stack.
Edge Impulse [101]	It offers full TinyML pipeline that covers the entire data analytics i.e. data collection, model training, and model optimization.
ARM and Cartesiam.ai [102]	Cartesiam.ai provides Nano_Edge AI, a tool that creates software models on the endpoint based on sensor behavior observed in real-time conditions.
Microsoft [103, 104]	Microsoft has recently introduced their TinyML framework, Embedded learning library (ELL). Moreover, Microsoft has also: i Released Bonsai an optimized Tree based algorithm. ii Released ProtoNN an optimized KNN based algorithm.
Ericsson [105]	Introduced the concept of TinyML-as-a-service and are focused to provide wide variety of market solutions.
Texas Instruments (TI) [106]	Recently, TI has come up with Sitara line of processors to enable vision-based deep learning interface at the edge in factory automation.
Eta Computer and Edge Impulse [107]	Aims to combine the strength of Eta Computer's neural sensor processor, the world's lowest power neural sensor processor, ECM 3532 with TinyML platform of Edge Impulse. Their objective is to: i Accelerate time-to-market of ML in billions of low power IoT products. ii Design extremely low battery consuming IoT devices.
Qeexo [108]	The AutoML platform offered by Qeexo supports the smallest ARM cortex processors. Qeexo is the first vendor to automate ML on the ARM processor.
Renesas [109]	Renesas have optimized the libraries generated from Qeexo AutoML platform for running TinyML on Renesas Synergy S5D9 and Renesas RA6M3 processors. Further, the platform of Renesas have: i Wide ranges of ML models like: GBM, XGBoost, Random Forest, Logistic Regression, CNN, RNN, ANN, Local Outlier Factor, and Isolation Forest. ii No coding requirement. iii TinyML solutions targeted for wearable, mobile, IoT, automotive, smart home and appliances.

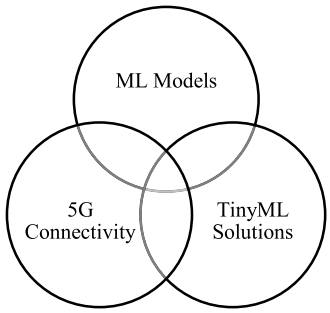


Fig. 6. Union of ML, 5G, and TinyML.

efficiently due to its following vision [111].

- Data rate in real-time is aimed at = 1∼10 Gbps [112], with latency<10 ms [113].
- It promises to offer enormous bandwidth, low cost, and longer battery life [114-116].
- More number of connected devices [116].
- Energy consumption is aimed to be reduced by almost 90% [110].

1.6. Challenges and Opportunities

Although a good amount of applications have shown the potential of TinyML, more research is required to understand the true benefits and shortcomings of these areas under discussion [33]. In our viewpoint among many others, here are some key issues that require further research in this area.

1.6.1. Benchmarking

Benchmarking has been extensively used earlier as a comprehensive assessment methodology to compare and validate the computational efficiency and readiness level of different technologies. The lack of a recognized benchmark due to the challenges posed by low power, limited memory, hardware heterogeneity, and software heterogeneity is an important impediment that may hamper TinyML services [30]. In this context, the IoT community has shown an increasing interest in benchmarking as a way to scientifically compare the performance of various TinyML solutions. So far, no standard assessment model has been formally available therefore, a unified and broadly applicable benchmark methodology for TinyML is eagerly anticipated.

1.6.2. Inconsistent Power Usage

Inconsistent power drainage is a serious concern in TinyML, firstly, due to the power requirement for different MCUs are dissimilar and, secondly, analytics like data processing and ML services requires more power compared with simple data gathering task. This also acts as a barrier in benchmarking TinyML systems. Albeit, rapid progress in TinyML power management is critical to achieving the reliability goals of always-on battery-powered applications. Therefore, there is a major headroom for research in this direction. Furthermore, as the need for computation in MCUs intensified the processor performance has increased exponentially, even outpacing Moore's law [117]. However, battery capacity and technology are improving very slowly (5-8% per year) almost at a linear pace [118]. Thus, to provide data-driven features within the same thermal limits and battery life is a challenging task.

1.6.3. Memory Constraints

Insatiable demand for computation and high accuracy has continued to push the innovations in ML algorithms. However, integrating novel ML models in MCUs presents two key challenges: 1) the size complexity of the model would not allow for direct implementation in MCU, and 2) there will be a requirement for innovative optimization techniques for each algorithm. As the consequence of aggressive compression as the ML model shrinks, it is important that the performance parameters be matched to the original model. Further, a balance has to be drawn among reduced memory footprint and system accuracy. In the present scenario, IoT devices largely depend upon the cloud to perform complex analysis. However, researchers are desperately working towards a new technological breakthrough that would effortlessly facilitate the large models in tiny MCUs. This breakthrough in our opinion would accelerate TinyML use cases and acceptance.

1.6.4. Processor Power

Although MCUs like ARM Cortex M series are equipped with high-end processors, compared to cloud-based systems their speed is still very low. Therefore it may hamper the quality of service while shifting analytics from cloud to device. Concurrently, an exponentially growing number of software frameworks are addressing the issue of compressing the ML algorithms, resulting in rich and highly heterogeneous frameworks. However, a standardized framework has not yet been available.

1.6.5. TinyML Cannot Solve Everything

Shifting ML in IoT devices does not necessarily negate the use of cloud and edge services. Services like cloud and edge are required or not are application specific.

1.6.6. Web and Embedded are Different

The web and embedded world are totally different technologies. The web contains powerful CPU architectures with huge memory, processing power, and storage. Web services encompass the comfort and flexibility to choose among sophisticated operating systems and complex hardware. MCUs on the other hand, do not have any operating system and perform only the particular task specified by the programmer. The size of the program or memory requirement which does not concern the web is crucial in the case of MCUs. A broad comparison of embedded and web technology is detailed in Table V [18].

TABLE V
Resource Comparison of Embedded and Web Technologies.

Parameters	**CPU**	
	Embedded	**Web**
CPU	MCU	x86/ARM64
Memory	500 KB (flash)	In GBs
Storage	2 MB	In TBs
Operating System	Real time, Embedded	Multi-user, Multi-tasking, Distributed
Virtualization	No	Yes

1.6.7. The ML is big and resource-demanding

There is a lack of ML-dedicated hardware that undermines the homogenous and seamless ML cloud to embedded deployment. Further, the use of sophisticated high-level language to design a ML algorithm and software runtime makes software portability or compression very difficult. Therefore it is irrefutable that cumbersome ML tasks are still to be performed in the cloud. So, ML in embedded would replace web services completely is unrealistic.

1.6.8. Cost

It is well known that as the research in TinyML expands, alongside it an increasing requirement in performance and reliability of MCUs is desired. Indeed, maintaining the cost-reduction pace can be difficult while achieving the performance and reliability goals of MCUs [117]. Moreover, in large-scale IoT applications hundreds and thousands of sensors are to be employed, so cost-reduction is one of the major concerns in TinyML.

The issues presented here offer an important step forward in the future research direction. Since considerable challenges still exist, it is therefore imperative that we understand the complexities that must be addressed to unfold the route to new technologies that would deliver optimum TinyML solutions.

1.7. Conclusion and Discussion

TinyML however is a method that lends itself to adaptation for resource-limited settings by offering portability, intelligence, and simple protocol that complement pre-existing IoT devices. While IoT devices in the cloud-based frameworks continue to expand in the IoT ecosystem, TinyML technology to perform data analytics at the sensor node is critical to improving outcomes. Innovative approaches to optimization of ML algorithms along with innovation in hardware for implementation in resource-limited settings, however, have the potential to redefine the IoT architecture. On the other hand, caution must be taken while optimizing the ML algorithm as it can deteriorate the performance of the classifier.

In particular, TinyML gave rise to a new class of machine learning framework targeted for battery-powered and resource-constrained extreme edge devices of the IoT ecosystem. Though researchers have paid more attention to the development of software frameworks that aims to produce a compressed version of the ML model to incorporate in MCUs, optimizing the ML model is still a challenging issue in the TinyML community. TinyML use cases are widely gaining popularity in a broad range of applications. Accordingly, the effectiveness of the framework must be guaranteed to encourage innovation and further investment in this space. Therefore, an effective benchmarking method for TinyML is necessary. Based on the research works so far, we envisage that the development of a benchmarked TinyML solution, which would be accurate, simple, fast, and reliable enough to be applied in varied IoT environments, is in the offing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors

References

[1] Processor IP for the Wildest Range of Devices, Nov. 7, 2020. Accessed on[Online] Available, https://www.arm.com/products/silicon-ip-cpu.
[2] M.A. Mazidi, J.G. Mazidi, R.D. McKinlay, The 8051 microcontroller and embedded systems using assembly and C, Pearson Education, 2005, 2006.
[3] J. Portilla, G. Mujica, J.S. Lee, T. Riesgo, The extreme edge at the bottom of the Internet of Things: A review, IEEE Sensors J 19 (9) (2019) 3179–3190.
[4] W. Shi, J. Cao, Q. Zhang, Y. Li, L Xu, Edge computing: Vision and challenges, IEEE Internet Things J 3 (5) (2016) 637–646.
[5] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: A survey, IEEE Internet Things J 5 (1) (2017) 450–465.
[6] A.E.-H.G. El-Barbary, L.A.A. El-Sayed, H.H. Aly, M.N. El-Derini, A cloudlet architecture using mobile devices, in: Proc. IEEE/ACS 12th Int. Conf. Comput. Syst. Appl. (AICCSA), Nov. 2015, pp. 1–8.
[7] Y. Liu, M.J. Lee, Y. Zheng, Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system, IEEE Trans. Mobile Comput. 15 (10) (2016) 2398–2410.
[8] A. Taherkordi, F. Eliassen, Poster abstract: Data-centric IoT services provisioning in fog-cloud computing systems, in: *Proc. IEEE/ACM 2nd Int. Conf. Internet Things Design Implement. (IoTDI)*, Pittsburgh, PA, USA, 2017, pp. 317–318.
[9] M.R. Alam, M.B.I. Reaz, M.A.M. Ali, A review of smart homes—Past, present, and future, IEEE Trans. Syst., Man, and Cybern. C, Appl. Rev. 42 (6) (2012) 1190–1203.
[10] M. Roopaei, P. Rad, K.K.R. Choo, Cloud of things in smart agriculture: Intelligent irrigation monitoring by thermal imaging, IEEE Trans. Cloud Comput. 4 (1) (2017) 10–15.
[11] K. Zhang, J. Ni, K. Yang, X. Liang, J. Ren, X.S. Shen, Security and privacy in smart city applications: Challenges and solutions, IEEE Communications Magazine 55 (1) (2017) 122–129.
[12] S. Pinto, J. Cabral, T. Gomes, We-care: An IoT-based health care system for elderly people, in: Proc. IEEE Int. Conf. on Indus. Tech. (ICIT), Mar. 2017, pp. 1378–1383.
[13] Y.U. Devi, M.S.S. Rukmini, IoT in connected vehicles: Challenges and issues—A review, in: IEEE Int. Conf. on Sig. Proces., Commun., Power and Embedded System (SCOPES), Oct. 2016, pp. 1864–1867.
[14] L. Deng, D. Yu, Deep learning: methods and applications, Foundations and trends in signal processing 7 (3–4) (2014) 197–387.
[15] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang, S. Wanga, Image and video compression with neural networks: A review, IEEE Trans. Circuits Syst. Video Technol. 30 (6) (2020).

[16] Y. Jaafra, J.L. Laurent, A. Deruyver, M.S. Naceur, Reinforcement learning for neural architecture search: A review, Image and Vision Computing 89 (2019) 57–66.
[17] R. Sanchez-Iborra, A.F. Skarmeta, TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities, IEEE Circuits and Systems Magazine 20 (3) (2020) 4–18.
[18] H. Doyu, R. Morabito, J. Höller, Bringing Machine Learning to the Deepest IoT Edge with TinyML as-a-Service, IEEE IoT Newsletter 11 (Mar.2020).
[19] C. MacGillivray, M. Torchia, Internet of Things: Spending trends and outlook, Tech. Rep. (2019) [Online]Available, https://www.idc.com/getdoc.jsp?containerId=US45161419.
[20] D. Huang, Mobile cloud computing, IEEE COMSOC Multimedia Commun. Tech. Committee E-Lett. 6 (10) (Oct. 2011) 27–31.
[21] P.M. Mell, T. Grance, The NIST definition of cloud computing, Nat. Inst. Stand. Technol. (2011) 800–1145. Gaithersburg, MD, USA, Tech. Rep.
[22] T. Brummett, P. Sheinidashtegol, D. Sarkar, M. Galloway, Performance metrics of local cloud computing architectures, in: *Proc. IEEE 2nd Int. Conf. Cyber Security Cloud Comput. (CSCloud)*, New York, NY, USA, Nov. 2015, pp. 25–30.
[23] T. Zhao, S. Zhou, X. Guo, Y. Zhao, Z. Niu, A cooperative scheduling scheme of local cloud and Internet cloud for delay-aware mobile cloud computing, in: *Proc. IEEE Globecom Workshops (GC Wkshps)*, San Diego, CA, USA, Dec. 2015, pp. 1–6.
[24] Z. Pang, L. Sun, Z. Wang, E. Tian, S. Yang, A survey of cloudlet based mobile computing, in: *Proc. Int. Conf. Cloud Comput. Big Data (CCBD)*, Shanghai, China, Nov. 2015, pp. 268–275.
[25] T.H. Luan, L. Gao, Z. Li, Y. Xiang, L. Sun, Fog computing: Focusing on mobile users at the edge, CoRR abs/1502.01815 (2015).
[26] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Communications Surveys & Tutorials 19 (4) (2017) 2322–2358.
[27] Y.C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computing—A key technology towards 5G, ETSI white paper 11 (11) (2015) 1–16.
[28] M.T. Beck, M. Werner, S. Feld, S. Schimper, Mobile edge computing: A taxonomy, in: Proc. 6th Int. Conf. Adv. Future Internet, 2014, pp. 48–54.
[29] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, H. Kim, Characterizing the deployment of deep neural networks on commercial edge devices, in: Proc. IEEE International Symposium on Workload Characterization (IISWC), Nov. 2019, pp. 35–48.
[30] C.R. Banbury, et al., Benchmarking TinyML systems: Challenges and direction, in: Proc. 1st Int. Workshop Benchmarking Machine Learning Workloads Emerging Hardware - 3rd Conf. Machine Learning and Systems (MLSys), 2020 [Online]Available, https://arxiv.org/pdf/2003.04821.pdf.
[31] Tiny Ml. 2020. Accessed on Nov. 7, 2020. [Online]. Available: https://www.tinyml.org/home/.
[32] ARM-NN. 2020. Accessed on Nov. 7, 2020. [Online]. Available: https://github.com/ARM-software/armnn.
[33] *TinyML as a Service and the challenges of machine learning at the edge*. Accessed on Nov. 7, 2020. [Online]. Available: https://www.ericsson.com/en/blog/2019/12/tinyml-as-a-service.
[34] *What is the Cloud? How Does it Fit into the Internet of Things (IoT)?*Accessed on Nov. 7, 2020. [Online]. Available: https://www.iotforall.com/what-is-the-cloud#:~:text=The%20cloud%20is%20a%20huge,for%20businesses%20and%20for%20people.
[35] *Boeing 787s to Create Half a Terabyte of Data Per Flight, Says Virgin Atlantic*. Accessed on Nov. 7, 2020. [Online]. Available: https://datafloq.com/read/self-driving-carscreate-2-petabytes-data-annually/172.
[36] *Self-Driving Cars Will Create 2 Petabytes of Data, What are the Big Data Opportunities for the Car Industry?*Accessed on Nov. 7, 2020. [Online]. Available: http://www.computerworlduk.com/news/data/boeing-787screate-half-terabyte-of-data-per-flight-says-virgin-atlantic-3433595/.
[37] M. Wollschlaeger, T. Sauter, J. Jasperneite, The future of industrial communication: Automation networks in the era of the Internet of Things and industry 4.0, IEEE Ind. Electron. Mag. 11 (1) (Mar. 2017) 17–27.
[38] H. Faiedh, Z. Gafsi, K. Torki, K. Besbes, Digital hardware implementation of a neural network used for classification, in: Proc. IEEE Int. Conf. on Microelectronics (ICM), Dec. 2004, pp. 551–554.
[39] Y.F. Lure, Y.S.P. Chiou, H.Y.M. Yeh, N.C. Grody, Hardware based neural network data fusion for classification of Earth surface conditions, in: Proc. IEEE Signals, Systems and Computers (SC), ., Pacific Grove, CA, Jan. 1992, pp. 761–765.
[40] U. Farooq, M. Amar, K.M. Hasan, M.K. Akhtar, M.U. Asad, A. Iqbal, A low cost microcontroller implementation of neural network based hurdle avoidance controller for a car-like robot, in: *Proc. IEEE 2nd Int. Conf. on Computer and Automation Engineering (ICCAE)*, Feb. 1, 2010, pp. 592–597.
[41] L. Dutta, C. Talukdar, A. Hazarika, M. Bhuyan, A novel low cost hand-held tea flavor estimation system, IEEE Trans. Ind. Electron. 65 (6) (2017) 4983–4990.
[42] L. Zhang, Z.F. Wang, Design of embedded control system based on arm9 microcontroller, in: Proc. IEEE Int. Conf. on Electrical and Control Engineering, Jun. 2010, pp. 3579–3582.
[43] Guimarães, C.J. and Fernandes, M.A., "Real-time Neural Networks Implementation Proposal for Microcontrollers," Jun. 2020. [Online]. Available: https://arxiv.org/pdf/2006.05344.pdf.
[44] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, M .S. Lew, Deep learning for visual understanding: A review, Neurocomputing 187 (2016) 27–48.
[45] R. David, et al., "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," Oct. 2020. [Online]. Available: https://arxiv.org/pdf/2010.08678.pdf.
[46] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," Nov. 2017. [Online]. Available: https://arxiv.org/pdf/1711.07128.pdf%EF%BC%89%E5%B7%B2%E6%B7%BB%E5%8A%A0%E8%BF%9B%E6%9D%A5.
[47] C. Vuppalapati, et al., Democratization of AI, Albeit Constrained IoT Devices & Tiny ML, for Creating a Sustainable Food Future, in: Proc. IEEE 3rd Int. Conf. on Information and Computer Technologies (ICICT), Mar. 2020, pp. 525–530.
[48] K. Wang, Z. Liu, Y. Lin, J. Lin, S. Han, Haq: Hardware-aware automated quantization with mixed precision, in: Proc. IEEE conf. on computer vision and pattern recognition, 2019, pp. 8612–8620.
[49] I. Fedorov, R.P. Adams, M. Mattina, P. Whatmough, SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers, Advances in Neural Information Processing Systems 32 (2019) 4977–4989.
[50] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," Jan. 2018. [Online]. Available: https://arxiv.org/pdf/1801.06601.pdf.
[51] A. Garofalo, M. Rusci, F. Conti, D. Rossi, L. Benini, PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors, Philosophical Transactions of the Royal Society A 378 (2164) (2019) 2020, 0155.
[52] A. Hazarika, P. Barman, C. Talukdar, L. Dutta, A. Subasi, M. Bhuyan, Real-time implementation of a multidomain feature fusion model using inherently available large sensor data, IEEE Trans. Ind. Informat. 15 (12) (2019) 6231–6239.
[53] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," Mar. 2015. [Online]. Available: https://arxiv.org/pdf/1503.02531.pdf.
[54] N. Romano, and R. Schucker, "Distilling Knowledge to Specialist Networks for Clustered Classification," 2015. [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/120_Report.pdf.
[55] S. Han, J. Pool, J. Tran, and W.J. Dally, "Learning both weights and connections for efficient neural networks," Jun. 2015. [Online]. Available:https://arxiv.org/pdf/1506.02626.pdf%20http://arxiv.org/abs/1506.02626.pdf.
[56] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," Aug. 2016. [Online]. Available: https://arxiv.org/pdf/1802.05668.pdf.
[57] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," Oct. 2015. [Online]. Available: https://arxiv.org/pdf/1510.00149).
[58] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," Jun. 2018. [Online]. Available: https://arxiv.org/pdf/1806.08342.pdf.
[59] J. Frankle, and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," Mar. 2018. [Online]. Available: https://arxiv.org/pdf/1803.03635.pdf.
[60] G. Urban, et al., "Do deep convolutional nets really need to be deep and convolutional?" Mar. 2016. [Online]. Available: https://arxiv.org/pdf/1603.05691.pdf.

[61] W.M. Czarnecki, S. Osindero, M. Jaderberg, G. Świrszcz, and R. Pascanu, "Sobolev training for neural networks," Jun. 2017. [Online]. Available: https://arxiv.org/pdf/1706.04859.pdf.
[62] T. Furlanello, Z. Lipton, M. Tschannen, L. Itti, A. Anandkumar, Born again neural networks, in: Proc. PMLR 35th Int. Conf. on Machine Learning, Jul. 2018, pp. 1607–1616.
[63] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage. Advances in neural information processing systems, 1990, pp. 598–605.
[64] B. Hassibi, D.G. Stork, Second order derivatives for network pruning: Optimal brain surgeon. Advances in neural information processing systems, 1993, pp. 164–171.
[65] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," Aug. 2016. [Online]. Available: https://arxiv.org/abs/1608.04493.
[66] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," Oct. 2018. [Online]. Available: https://arxiv.org/pdf/1810.05270.pdf.
[67] M.D. McDonnell, T. Vladusich, Enhanced image classification with a fast-learning shallow convolutional neural network, in: *Proc. IEEE Int. Joint Conf. on Neural Networks (IJCNN)*, Killarney, Ireland, 12 July 2015, pp. 1–7.
[68] B. Moons, D. Bankman, L. Yang, B. Murmann, M. Verhelst, BinarEye: An always-on energy-accuracy-scalable binary CNN processor with all memory on chip in 28nm CMOS, in: *Proc. IEEE Custom Integrated Circuits Conference (CICC)*,, Apr. 2018, pp. 1–4.
[69] E. Flamand, F.Conti D.Rossi, I. Loi, A. Pullini, F. Rotenberg, L. Benini, GAP-8: A RISC-V SoC for AI at the Edge of the IoT, in: Proc. IEEE 29th Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP), Jul. 2018, pp. 1–4.
[70] ETHOS-U55. Accessed on Nov. 7, 2020. [Online]. Available: https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55.
[71] ECM3532 - Eta Compute Accessed on Nov. 7, 2020. [Online]. Available: https://en.wikichip.org/wiki/eta_compute/ecm353x/ecm3532.
[72] A. Capotondi, M. Rusci, M. Fariselli, L. Benini, CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices, IEEE Trans. Circuits Syst., II, Exp. Briefs 67 (5) (2020) 871–875.
[73] M. de Prado, R. Donze, A. Capotondi, M. Rusci, S. Monnerat, L. Benini, and N. Pazos, "Robust navigation with tinyML for autonomous mini-vehicles," Jul. 2020. [Online]. Available: https://arxiv.org/pdf/2007.00302.pdf.
[74] A. Wong, M. Famuori, M. J. Shafiee, F. Li, B. Chwyl, and J. Chung, "YOLO nano: A highly compact you only look once convolutional neural network for object detection," Oct. 2019. [Online]. Available: https://arxiv.org/pdf/1910.01271.pdf.
[75] R. Andri, L. Cavigelli, D. Rossi, L. Benini, YodaNN: An Architecture for Ultralow Power Binary-Weight CNN Acceleration, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 37 (1) (Jan. 2018) 48–60.
[76] E. Liberis and N. D. Lane, "Neural networks on microcontrollers: saving memory at inference via operator reordering," Oct. 2019. [Online]. Available: https://arxiv.org/pdf/1910.05110.pdf.
[77] W. Niu, et al., Patdnn: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning, in: Proc. Twenty-Fifth Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Mar. 2020, pp. 907–922.
[78] D. Gope, J. G. Beu, U. Thakker, and M. Mattina, "Aggressive compression of mobilenets using hybrid ternary layers," 2020. [Online]. Available: https://www.tinyml.org/summit/abstracts/Gope_Dibakar_poster_abstract.pdf.
[79] A. Wong, M. Famouri, and M. J. Shafiee, "AttendNets: Tiny Deep Image Recognition Neural Networks for the Edge via Visual Attention Condensers," Sep. 2020. [Online]. Available: https://arxiv.org/pdf/2009.14385.pdf.
[80] A. Wong, M. Famouri, M. Pavlova, and S. Surana, "Tinyspeech: Attention condensers for deep speech recognition neural networks on edge devices. Aug. 2020. [Online]. Available: https://arxiv.org/pdf/2008.04245.pdf%20.
[81] F. Alongi, N. Ghielmetti, D. Pau, F. Terraneo, W. Fornaciari, Tiny Neural Networks for Environmental Predictions: an integrated approach with Miosix, in: Proc. IEEE Int. Conf. on Smart Computing (SMARTCOMP), Sep. 2020, pp. 350–355.
[82] P. Blouw, G. Malik, B. Morcos, A. R. Voelker, and C. Eliasmith, "Hardware Aware Training for Efficient Keyword Spotting on General Purpose and Specialized Hardware," Sep. 2020. [Online]. Available: https://arxiv.org/pdf/2009.04465.pdf.
[83] A. Voelker, I. Kajić, C. Eliasmith, Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks, in: Proc. 33rd Conference on Neural Information Processing Systems (NeurIPS), 2019, pp. 15570–15579.
[84] S.V. Lahade, S. Namuduri, H. Upadhyay, S. Bhansali, Alcohol Sensor Calibration on the Edge Using Tiny Machine Learning (Tiny-ML) Hardware, in: 237th ECS Meeting with the 18th International Meeting on Chemical Sensors (IMCS 2020), May 2020, p. 1848. ECS Meeting Abstracts MA2020-01 1848.
[85] S. Gopinath, N. Ghanathe, V. Seshadri, R. Sharma, Compiling KB-sized machine learning models to tiny IoT devices, in: Proc. 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, Jun. 2019, pp. 79–95.
[86] M.T. Nyamukuru, K.M. Odame, Tiny Eats: Eating Detection on a Microcontroller, in: Proc. IEEE Second Workshop on Machine Learning on Edge in Sensor Systems (SenSys-ML), Apr. 2020, pp. 19–23.
[87] G. Crocioni, D. Pau, J.M. Delorme, G. Gruosso, Li-Ion Batteries Parameter Estimation With Tiny Neural Networks Embedded on Intelligent IoT Microcontrollers, IEEE Access 8 (2020) 122135–122146.
[88] P. Mohan, A. J. Paul, and A. Chirania, "A Tiny CNN Architecture for Medical Face Mask Detection for Resource-Constrained Endpoints," [Online]. Available: https://arxiv.org/pdf/2011.14858.pdf.
[89] *Tiny ML Summit*. Accessed on Nov. 7, 2020. [Online]. Available: https://www.tinyml.org/summit.
[90] P. Warden, D. Situnayake, Tinyml: Machine Learning with tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers, O'Reilly, UK, 2019.
[91] *Intro to Tiny ML*. Accessed on Nov. 7, 2020. [Online]. Available: https://www.digikey.in/en/maker/projects/intro-to-tinyml-part-1-training-a-model-for-arduino-in-tensorflow/8f1fc8c0b83d417ab521c48864d2a8ec.
[92] Easy TinyML on ESP32 and Arduino, Nov. 7, 2020. Accessed on[Online]. Available, https://www.hackster.io/news/easy-tinyml-on-esp32-and-arduino-a9dbc509f26c.
[93] Train a TinyML model that can recognize sounds using only 23 kB of RAM, Nov. 7, 2020. Accessed on[Online]. Available, https://www.edgeimpulse.com/blog/train-a-tiny-ml-model.
[94] *Cough Detection with TinyML on Arduino.*Accessed on Nov. 7, 2020. [Online]. Available: https://create.arduino.cc/projecthub/edge-impulse/cough-detection-with-tinyml-on-arduino-417f37.
[95] *Tiny ML on Arduino*, Gesture recognition tutorial results in a blank model.h, Nov. 7, 2020. Accessed on[Online]. Available, https://github.com/arduino/ArduinoTensorFlowLiteTutorials/issues/11.
[96] Tutorial on micro-kernel based hardware acceleration, Nov. 7, 2020. Accessed on[Online]. Available, https://forums.tinyml.org/t/tinyml-talks-on-august-13-2020-tutorial-on-micro-kernel-based-hardware-acceleration-by-manu-rastogi/327.
[97] AutoML for TinyML with Once-for-All Network, Nov. 7, 2020. Accessed on[Online]. Available, https://www.tinyml.org/wp-content/uploads/2020/04/tinyML_Talks_Song_Han_200428.pdf.
[98] AI Speech Recognition with TensorFlow Lite for Microcontrollers and SparkFun Edge, Nov. 7, 2020. Accessed on[Online]. Available, https://codelabs.developers.google.com/codelabs/sparkfun-tensorflow.
[99] TinyML Enables Smallest Endpoint AI Devices, Nov. 7, 2020. Accessed on[Online]. Available, https://www.arm.com/blogs/blueprint/tinyml.
[100] Deploy machine learning models on mobile and IoT devices, Nov. 7, 2020. Accessed on[Online]. Available, https://www.tensorflow.org/lite.
[101] Embedded TinyML for beginner and advanced developers, Nov. 7, 2020. Accessed on[Online]. Available, https://www.edgeimpulse.com/.
[102] Discover NanoEdge AI studio the market leader for EDGE, Dec. 4, 2020. Accessed on[Online]. Available, https://cartesiam.ai/product/.
[103] Embedded Learning Library (ELL), Nov. 7, 2020. Accessed on[Online]. Available, https://microsoft.github.io/ELL/.
[104] Resource-efficient ML for Edge and Endpoint IoT Devices, Nov. 7, 2020. Accessed on[Online]. Available, https://www.microsoft.com/en-us/research/project/resource-efficient-ml-for-the-edge-and-endpoint-iot-devices/.
[105] *Tiny ML as-a-Service*, What is it and what does it mean for the IoT Edge?, Nov. 7, 2020. Accessed on[Online]. Available, https://www.ericsson.com/en/blog/2019/12/tinyml-as-a-service-iot-edge.

[106] Sitara™ processors, Nov. 7, 2020. Accessed on[Online]. Available, https://www.ti.com/processors/sitara-arm/overview.html.
[107] Eta Compute Partners with Edge Impulse to Accelerate the Development and Deployment of Machine Learning at the Edge, Nov. 7, 2020. Accessed on[Online]. Available https://etacompute.com/news/press-releases/eta-compute-partners-with-edge-impulse-to-accelerate-the-development-and-deployment-of-machine-learning-at-the-edge/.
[108] *AutoML.*Accessed on Nov. 7, 2020. [Online]. Available: https://automl.qeexo.com/project/create.
[109] *Qeexo AutoML for Embedded Devices.*Accessed on Nov. 7, 2020. [Online]. Available: https://www.renesas.com/us/en/products/microcontrollers-microprocessors/ra-cortex-m-mcus/ra-partners/qeexo-automl.
[110] L. Chettri, R. Bera, A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems, IEEE Internet Things J 7 (1) (2019) 16–32.
[111] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A Survey on Internet of Things: Architecture, Enabling Technologies, Security, privacy, and Applications, IEEE Internet Things J 4 (5) (Oct. 2017).
[112] M. Agiwal, A. Roy, N. Saxena, Next Generation 5G Wireless Networks: A Comprehensive Survey, IEEE Commun. Surveys Tuts. 18 (3) (2016) third quarter.
[113] C. Bockelmann, et al., Massive Machine-Type Communications in 5G: Physical and MAC-Layer Solutions", IEEE communications magazine (Sep. 2017).
[114] E. Borgia, The Internet of Things vision: Key features, applications and open issues, Comput. Commun 54 (12) (2014) 1–31.
[115] H. Shariatmadari, et al., Machine-type communications: current status and future perspectives toward 5G systems",, IEEE communications magazine-communication standard supplement (Sep. 2015).
[116] M.R. Palattella, et al., Internet of Things in the 5G era: Enablers, architecture, and business models, IEEE J. Sel. Areas Commun. 34 (3) (Mar. 2016) 510–527.
[117] M. Helm, et al., A 128Gb MLC NAND-Flash device using 16nm planar cell, in: Proc. IEEE Int. Solid-State Circuits Conference Digest of Technical Papers (ISSCC), Feb. 2014, pp. 326–327.
[118] M.K. Tsai, Cloud 2.0 clients and connectivity—Technology and challenges, in: Proc. IEEE Int. Solid-State Circuits Conference Digest of Technical Papers (ISSCC), Feb. 2014, pp. 15–19.