chips

MDPI

Article

A Quantitative Review of Automated Neural Search and On-Device Learning for Tiny Devices

Danilo Pietro Pau *, Prem Kumar Ambrose and Fabrizio Maria Aymone

System Research and Applications, STMicroelectronics, 20864 Agrate Brianza, Italy
* Correspondence: danilo.pau@st.com

Abstract: This paper presents a state-of-the-art review of different approaches for Neural Architecture Search targeting resource-constrained devices such as microcontrollers, as well as the implementations of on-device learning techniques for them. Approaches such as MCUNet have been able to drive the design of tiny neural architectures with low memory and computational requirements which can be deployed effectively on microcontrollers. Regarding on-device learning, there are various solutions that have addressed concept drift and have coped with the accuracy drop in real-time data depending on the task targeted, and these rely on a variety of learning methods. For computer vision, MCUNetV3 uses backpropagation and represents a state-of-the-art solution. The Restricted Coulomb Energy Neural Network is a promising method for learning with an extremely low memory footprint and computational complexity, which should be considered for future investigations.

Keywords: tiny devices; resource constraints; tiny machine learning; micro controllers; neural architecture search; hyper parameter optimizations; on device learning

1. Introduction

Tiny Machine Learning (TinyML) [1] is an artificial intelligence (AI) field which focuses on the technologies and applications for extremely low-complex devices. Since 2019, the TinyML Foundation has created a vibrant community focused on the development and knowledge sharing of algorithms, tools, hardware, software, and applications of tiny machine learning for the analytics of data, aimed at being deployed on low-power (<1 mW) resource-constrained devices such as micro controllers. In parallel, given the rising interest in the field, the MLCommons Tiny working group developed open benchmarks [2] focused on micro-controllers and tiny neural processing units. In such a rapidly evolving context, the aim of this paper is to review the different approaches and technologies introduced within the TinyML community and other research groups which help the deployment of ML on low-profile edge devices. The limitations, constraints, and the challenges of these approaches are discussed in detail, and the quantitative results are reported when available from the corresponding papers. In Section 2, Automated Machine Learning and Neural Architecture Search (NAS) are discussed to provide an overview of such technologies which aim to help the productivity of the TinyML community. In Section 3, the available state-of-the-art papers discuss how NAS helps in deriving architectures within technological constraints. It also lists the approaches and the different hardware constraints imposed on the search of such machine learning architectures. Section 4 discusses the on-device learning of tiny models with respect to the real-time data and how it helps in preserving memory in tiny devices. Based on the learning method used, different solutions for on-device learning are also presented. The latter part of the section discusses the Restricted Coulomb Energy Neural Network (RCE-NN) and how it could be helpful in some future works to achieve an extremely low memory footprint and less computational power with online learning capabilities. The initial results using this approach are also presented in this paper.

Citation: Pau, D.P.; Ambrose, P.K.; Aymone, F.M. A Quantitative Review of Automated Neural Search and On-Device Learning for Tiny Devices. *Chips* **2023**, *2*, 130–141. https://doi.org/10.3390/chips2020008

Academic Editor: Paolo Crippa

Received: 9 February 2023
Revised: 21 March 2023
Accepted: 23 April 2023
Published: 9 May 2023

2. Automated Machine Learning

Automated Machine Learning (AutoML) [3] tools automatically design a machine learning algorithm and simultaneously set its hyperparameters to optimize its empirical performances on a given dataset that shapes an application problem.

AutoML sets a field which helps the ML and embedded C developer experts to achieve higher design productivity. These people do not have years of pre-existing knowledge and experience of applying ML to their problems; therefore, there is an increasing need to make the process easier and more productive.

AutoML focuses on the automation of several problems associated with the extraction, transformation, and loading of data and the training and deployment of the models which need to be deployed on resource-unconstrained (non-tiny) processors. Optimization of the hyperparameters (HPO) is one of the major focuses of AutoML.

When dealing with tiny devices, resource constraints are the major challenges to address. In most known cases, the resource constraints are not factored in by AutoML tools in their inner operations.

Neural Architecture Search

The associated tools require an extensive level of expertise in this area and strong insights to find a more efficient hyperparameter choice and architecture. This demands the training, validation, and testing of any feasible and deployable architecture over the span of a few days to weeks. It might be difficult for end users and developers with little to no background in neural networks and machine learning to comprehend and create such an effective design. The Neural Architecture Search techniques are used to support efforts in such circumstances. When an ML algorithm is an Artificial Neural Network (ANN), AutoML specializes in Neural Architecture Search (NAS) [4]. It is therefore a focused subset of AutoML to ANN. NAS aims to find the best architecture with a better performance for a neural network. It takes the task carried out by human experts hand-crafting an ANN (topology and associated hyper parameters) and automates this task to find out more complex architectures, and it performs even better than manually shaped ones. It comprises a set of tools and methods which explore a large hyper dimensional search space to train, evaluate, and test using several optimization strategies, and it selects the resulting ANN which performs accurately for the given target by maximizing an objective function. Although NAS seems to be a relatively different field, the underlying problem is similar to that of the hyperparameter optimization. Designing an optimal, accurate, and lightweight ANN to fit the target devices with limited resources is a challenge which is addressed by the research community. Many neural network architectures that were hand-crafted do not take into consideration the hardware constraints that are faced by embedded and tiny edge devices. These expertly designed architectures are usually meant for the deployment of powerful GPUs, which have a lot of memory and computational assets available in their chip and associated system embodiments. NAS can be made to design an accurate network automatically by optimizing over a large search space of the given requirements. NAS can optimize several metrics during the design process, such as memory requirements, FLOPs, MACCs, latency, inference per second, etc. Starting with these considerations, this paper reviews the state-of-the-art approaches and optimization methods for designing neural architectures for tiny devices such as MCU which helps in deploying complex ANN into off-the-shelf MCUs. Then, this paper reviews the different approaches addressed by the research community to enable and improve on-device learning on the MCUs.

3. NAS Approaches for Tiny Devices

These solutions are mainly focused on memory-constrained embedded devices. The space for the architecture search is optimized to find tiny ANNs.

Hard Constrained diffeRentiable Neural Architecture Search (HardCoReNAS) [5], which targets resource-constrained NAS, works by formulating the exact requirements of resources with a scalable search. This approach generates architectures that strictly adhere

to the tight resource limitations without the need for any tuning. A continuous probability distribution is induced over the search space of the proposed HardCoReNAS approach, and this makes the search space continuous. This helps to create a sample sub-network using the Gumbel SoftMax Trick [6]. The search space is divided into Micro and Macro spaces, where the Micro space is used to control the internal structures of each building block of the network. These blocks are the elastic versions of the MBInvRes block [7]. In contrast, the Macro search space is used to control how these blocks relate to each other and how these blocks are interconnected. This solution mainly focuses on the latency and is performed on the ImageNet dataset with a top-1 accuracy of 75.7%, 77.3% and 77.9% under latencies of 27 ms, 32 ms and 41 ms, respectively, using an Nvidia P100 GPU with 400 + GPU hours of search.

The structural wired Neural Architecture Search for the internet of things (MSNet) [8] is a graph-based NAS that overcomes the difficulties with graph-based approaches discovering high-quality models because of the search flexibility, accuracy density, and node dependency restrictions. In contrast to other NAS works that focus on identifying the best neural architecture, MSNAS looks for a variety of neural architectures that can meet various resource budgets. This work focuses on reducing the size of the model to as low as 200 KiB of peak memory usage and 42 M MACCs (multiply and accumulate operations) on a Visual Wake Words (VWW) dataset with an accuracy of 93.5%, which is 3.4% higher than MobileNetV2, and with 250 KiB peak memory usage for ImageNet-1000 with an accuracy of 59.1%, which is 0.9% higher than MobileNetV2. In addition, the structure level pruning method is used to explore a compact architecture with a higher pruning level to lower the MACCs, and the latency decreases with the increased level of pruning. By exploiting the structure level pruning, MSNet was able to generate new ANN for new tasks without further training, and on the MNIST task, it performed with an accuracy of 99.24%.

Co-Design NAS [9] is a framework which enables the joint exploration of the space of neural architectures, hardware implementation, and quantization. It is a combination of pure software NAS and hardware-aware NAS which is a joint exploration of neural architecture and hardware spaces. Exploring both the spaces and finding a Pareto frontier between hardware efficiency and accuracy is the proposal. The search process is computationally heavy considering the joint exploration on CIFAR10 with up to 300,000 LUTs. From the experiments, the two best architectures were found by the NAS in 1000 episodes. The strides from these two sampled designs were removed, which produced another two designs. A quantization search of these four designs produced results with accuracies of 86.16%, 86.26%, 87.34% and 88.53%. This method is more flexible and robust compared to a traditional design using fixed architecture.

The authors of [9] presented a lightweight design with an accuracy of 82.98% and 1293 images/second throughput, where even the conventional approaches are unable to produce a valid solution.

E-DNAS [10] is a differentiable architecture search method for a designed lightweight ANN. This method finds networks with low latency and a more accurate ANN which can be deployed on memory-constrained devices. The three main ideas behind this approach are a depth-aware convolution to compute high-resolution feature maps, a parallel architecture search pipeline on the feature maps, and to learn the optimal size and parameters of the convolution kernels. This optimization process is driven by a multi-objective differentiable loss function of accuracy and latency. Lastly, to increase the architecture search speed, a novel block is used which connects the learned meta kernels during training. The results were achieved with an ImageNet top-1 accuracy of 76.9% with 5.9 M parameters and a latency of 38 ms on ARM Cortex-A15-based hardware TDA2 which was running at 1.5 GHz, a dual-core DSP C66x processor capable of running deep learning inference, and an embedded vision engine subsystem (EVE). Unfortunately, this method has not been tested on MCU-level memory constraints.

Approaches for Microcontrollers

SVM-CBO$_{RF}$ [11] proposes a framework, based on Bayesian Optimization (BO), to optimize the hyperparameters of a Convolutional Neural Network (CNN) by dealing with black box deployable constraints (e.g., memory occupation) extracted from the STM32Cube.AI tool; in short, this tool automatically profiles a pre-trained ANN and reports several complexity metrics. It is composed of two different phases. In the first phase, a non-linear SVM classifier is used to approximate the feasible region of the search space associated with the hyperparameter values most likely to lead to ANN models deployable on MCU. In the second phase, BO is focused on the estimated feasible region with the aim of optimizing the loss function. Moreover, a probabilistic regression model, specifically a Random Forest, is used to approximate the objective function by using the Lower Confidence Bound. The experiments were performed on an STM32L4 series MCU and an ARM Cortex M4 MCU running at 80 MHz, embodying1MByte ROM and 128 KiByte RAM. An STM32Cube.AI tool was used to output the optimized ANSI C code which can be compiled and run on MCU for any ANN topology. The C code can be deployed on STM32 series MCU devices. The results of the human activity recognition task show that the optimal CNN models identified by SVM-CBO$_{RF}$ performed with an accuracy of 92.93 +/− 0.55%, which is 0.86% higher than the baseline model, by using peak RAM of 23.48 KiByte.

MCUNet [12] Tiny Deep Learning on IoT Devices is a framework of system-algorithm co-design that jointly optimizes the neural architecture with TinyNAS and the scheduling of the inference with TinyEngine in the same loop. TinyEngine offloads redundant operations from the runtime to compile time and only generates the code that will be executed by the TinyNAS which helps in reducing the memory requirements of the inference and allows more memory for the model size. TinyNAS takes advantage of the reduction in memory by TinyEngine and finds a high-accuracy model compared to existing frameworks. It is a combination of one-shot NAS giving a super network with all subnetworks. This optimized space of subnetworks undergoes an evolution search to find the best architecture. MCUNet achieved a record large-scale ImageNet accuracy of 70.7% on off-the-shelf STM32H743 MCU with a peak SRAM usage of ~490 KiB and Flash usage of 1.9 MB. Other experiments include VWW and object detection on different series of STM32 MCUs (F412, F746, F765, H743). MCUNet produced solutions which accelerated the inference of VWW applications by 2.4–3.4 times. Page No. 8 Figure 9 of [12] plots the Accuracy vs. Latency/SRAM memory trade-off on VWW and Speech Commands datasets. Large datasets such as ImageNet can be used by this method. This supports different constraints such as RAM, Flash, and inference time.

µNAS [13] is the combination of a highly granular search space which takes into consideration almost every aspect of a network such as a layer's kernel size, stride, channels, pooling size, fully connected layer's output dimension, connection between each layer, etc. Next, the accurate resource use of computation is considered where the peak memory usage, model size, and latency are the different aspects driving the loss function. Aging evolution and BO were the two search algorithms compared. The former tends to perform better than the latter. In addition to that model, compressing and pruning were performed to reduce the memory requirements of the MCU. The experimental results showed an accuracy of 77.49% on CIFAR-10 with a model size of 685 KiB, RAM usage of 909 Bytes, and 41.2 K MACs. This performance is better compared to the previous solutions in a highly constrained device such as an MCU. It is still time-consuming to search for architectures. Improvements can be made to reduce the search time by employing weight sharing to reduce the cost of training each candidate network and by not using the same search space throughout the entire search process by using a parameterized space granularity which can vary throughout the search space.

When it comes to ANNs whose computation graph contains branches, such as ResNet or Inception, there are different possible orders of evaluation of the layers. Starting from the observation that certain orders of evaluation require more peak memory usage with respect to others, ref. [14] proposes a method that finds the memory-optimal execution

schedule for an arbitrary network. This method was able to save up to 50 KiB of peak memory usage for SwiftNet Cell and 186 KiB reduction in MobileNet v1 with an increased execution time and decreased power consumption.

In a CNN architecture, the memory bottleneck is due to an imbalanced activation memory distribution across the layers. The activation maps of the first layers, in fact, have an order of magnitude larger memory than the later layers, thus representing the peak memory usage of the network. This leaves room for improvement, prompting researchers to find solutions to the problem. MCUNetV2 [15] proposes patch-by-patch inference scheduling for the first memory-intensive stages of the CNN. The technique consists of operating on one small spatial region of the feature map at a time, reducing the peak memory usage of existing networks by 4-8x. However, it also leads to overlapping patches and computation overhead, which is alleviated by redistributing the size of the receptive field to later stages of the network. With the joint design of network architecture and inference scheduling, MCUNetV2 achieves a 71.8% ImageNet record accuracy on MCU and >90% accuracy on the visual wake words dataset under only 32 KiB SRAM. It also unblocks object detection for tiny devices, achieving 16.9% higher mAP on Pascal VOC with respect to state-of-the-art results. A naive solution to the high peak memory usage of the initial layers of a CNN would be to aggressively down sample images via pooling or strided convolution, leading to deteriorated accuracy. RNNPool [16] introduces a novel pooling operator based on a Recurrent Neural Network (RNN), capable of aggregating features over large patches of an image and rapidly down sampling activation maps. It effectively reduces computational complexity and peak memory usage while retaining accuracy. RNNPool was applied on the S3FD architecture to construct a face detection method that achieves state-of-the-art MAP under 256 KiB. Table 1 shows the overview of all the above-mentioned approaches for both MCUs and other edge devices and their performances.

Table 1. Overview of different resource-constrained NAS approaches and their performances.

Solution	Approach	Target Device	Constraint	Dataset	Best Acc.%	GPU Hours	Tested Application
HardCoRe-NAS [5]	Differentiable search space + one shot	Edge-GPU/CPU	Latency	ImageNet	77.9	400	Image classification
MSNet [8]	Evolution Search	100–320 K SRAM, (256 KiB-1 MB) Flash	Peak memory usage	CIFAR-10	89.09	8	Image classification, VWW
Co-Design NAS [9]	RL (architecture and quantization space)	(0.5–3.5 MB) Flash AIoT/Mobile embedded	Peak memory usage and throughput	CIFAR-10	88.53	NA	Image classification
E-DNAS [10]	Gradient	SoC (ARM cortex-A15)	Low Latency, Memory	ImageNet	76.9	70	Image classification
SVM-CBO$_{RF}$ [11]	Bayesian Optimization	MCU	Memory	User Identification from Walking Activity	92.93	200	Human activity recognition
MCUNet [12]	One shot + Evolution Search	MCU	Memory (model size + peak memory usage), Latency	ImageNet	70.7	300	Image classification, object detection, VWW
uNAS [13]	Aging Evolution	MCU	Model size, RAM usage	MNIST	99.19	30	Image classification
MCUNetV2 [15]	One shot + Evolution Search + Patch-based inference to reduce memory	MCU	Same as MCUNet	ImageNet Visual Wake Words Pascal VOC	71.8 >90 64.6–68.3 mAP	NA	Image classification, object detection, VWW

Neuro-symbolic artificial intelligence is a novel field of AI which combines the integrity of symbolic rule-based techniques with the robustness of machine learning models. They have demonstrated that they can outperform SOTA learning models in domains such as image and video reasoning and can obtain high accuracy with significantly less

training data [17]. TinyNS [18] is the first platform-aware neurosymbolic architecture search framework for the joint optimization of symbolic and neural operators. It uses a fast, gradient-free, black-box BO over discontinuous, conditional, numeric, and categorical search spaces to find the best combination of symbolic codes and neural networks within the MCU's resources. It has been demonstrated that TinyNS outperforms purely neural or purely symbolic approaches, while ensuring execution on constrained hardware.

4. On-Device Learning

This is an increasingly important topic for the evolution of the TinyML community since it tries to match the capability to learn with the available MCU constrained memory and computational power. In 2022, an ad hoc working group was created by the TinyML Community with the goal of making edge devices "smarter" and more efficient by observing changes in the data collected and self-adjusting/reconfiguring the device's operating model. Optionally, the "knowledge" gained by the device is shared with other deployed devices.

Indeed, the environment in which the inference ANN model is deployed is assumed to be constantly changing. Since the models were trained with past knowledge, the processing of the real and time data causes an accuracy drop by the ML inferences. This phenomenon is known as concept drift. Therefore, there is a need for constant updates of the inference model by learning from current data. On-device learning (ODL) helps shift the process from offline ML model training to automatically update it using real-time data. By ingesting new samples at run-time, the device simultaneously learns and deploys the model with constant adaptation. The accuracy needs to be kept at higher levels, and the learning process shall also reduce its memory footprint to fit MCU hardware assets. Learning on the MCU helps since it allows for their deployment scale, offers better personalization, increases privacy, and also enables federated learning (FL) [19]. In the current scenario of Industry 4.0, ref. [20] has introduced intelligent Cyber-Physical Systems (CPSs), which can predict faults with autonomous behavior and self-adaptation using ODL. This method helps to increase energy efficiency, reduce the bandwidth, and achieve device autonomy. When it comes to training a model on-device, the main hindrance with respect to sole inference resides in backpropagation (the most widely used algorithm to train ANNs) and the amount of memory it needs to run the learning task.

It is computationally expensive and memory intensive, as it performs an additional backward pass to backpropagate the gradient and stores the activations of all layers to update the weights. Starting from this premise, in the next section, this paper reviews and discusses, following the historical and consequential order of the publications, the different approaches for on-device learning in the MCUs.

4.1. Extreme Learning Machines

ELM [21], extreme learning machines, are feedforward neural networks for classification, regression, etc., where the parameters of hidden nodes need not be tuned. These hidden nodes can be randomly assigned and never updated or can be inherited from their ancestors without being changed. In most cases, the output weights of hidden nodes are usually learned in a single step, which essentially amounts to learning a linear model. ELM can learn thousands of times faster than networks trained using backpropagation and can drastically reduce peak memory usage and computations by not performing the backward pass.

The authors of [22] proposed an online sequential ELM (OS-ELM) which can adjust the parameters over new samples sequentially. This avoids the retraining over old samples. FOS-ELM [23] proposes a forgetting mechanism in addition to the OS-ELM which improves the performance on sequential data with time validity. FOS-ELM was tested with data of CHINA TEX. MCH. The average RMSE on testing data for a 4 sec period was 2.2335×10^{-3} which was less compared to 4.0175×10^{-3} for OS-ELM. The authors of [24] proposed a modified ELM with local connections (ELM-LC) which is designed for sparsifying the input-

hidden weights. The input and hidden nodes are made into groups, and a group of input nodes is connected to only one group of hidden node groups, and hidden-output weights are calculated by a least square learning method. This performs better than a traditional ELM. These methods are suitable candidates for the implementation of ELM in MCUs. A UCI dataset was tested with this method and achieved a mean test error of 4.04×10^{-2} compared to a traditional ELM with 7.86×10^{-2} and a reduction in the number of input-hidden weights from 3922 to 434 by ELM-LC.

4.2. Reservoir Computing

Reservoir computing (RC) extends the advantages of ELM to RNNs by replacing the hidden nodes with an untrained recurrent non-linear block, referred to as a reservoir. Extensive research has been conducted in the application of RC to ODL on MCUs.

The authors of [25] proposed a novel solution for online learning and real-time anomaly detection of pathological conditions using a low-power MCU from ECG signals. The proposed system, shown in Figure 1, is based on RC followed by Principal Component Analysis (PCA) and One-Class Support Vector Machine (OC-SVM). This eliminates the need for storing ECG signals for longer periods of time and avoids the time-consuming off-line search of anomalies. Echo State Networks (ESNs) are a well-studied reservoir computing paradigm, where the reservoir consists of a large, sparsely and randomly connected, non-linear layer. The authors of [26] proposed a Block-based Binary Shallow Echo State Network (BBS-ESN) which is a deeply quantized anomaly detector of oil leaks that happen in the wind turbines with fixed and minimal computational complexity. This network can be deployed on an off-the-shelf MCU, and the power consumption is greatly reduced. This is achieved via the binarization of images and one bit quantization of the network's weights and activations. The authors of [27] proposed a novel Field-Oriented Control algorithm by means of extreme learning to modify the behavior and performance of electromechanical systems which are highly nonlinear at high speed. The proposed Semi Binary Deep Echo State Network (SB-DESN) achieves good control accuracy. Moreover, it proposes a novel complexity optimization which reduces the memory footprint, such that it can be deployed even on MCUs. Interesting results were obtained on STM32H7 with an inference time of 20 µs. The authors of [28] proposed a highly accurate, less complex online learning anomaly detection Deep Echo State Network for water distribution systems which adapts to the time varying data distribution and can be deployed on an MCU. In this approach, online learning can be carried out in two different ways: single iteration and batch decomposition to save memory.

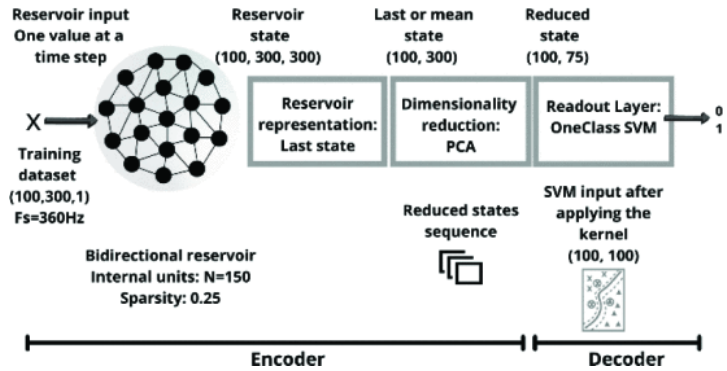

Figure 1. Architecture overview of ECG learning on anomalies [25].

4.3. Other Backpropagation-Free Solutions

ODL solutions that avoid using backpropagation, while still not using ELMs or RC, exist. TinyML with online learning on MCUs (TinyOL) [29] is implemented in C++ and can be attached to an arbitrary existing network as an additional layer in MCUs. The last additional layer learns from the new data and updates its weights. As the network learns

incrementally, there is no need to store the historical data for training, thus reducing the memory requirements. New classes can be added by the last layer and can be trained upon user request. This method is like transfer learning in which fine tuning happens in the last few layers. Only one data pair of the real-time stream is stored in the memory, thus reducing the memory footprint. It is flexible enough to modify the layer structure on the fly. It has been proven to be robust against concept drift. Yet, this approach is limited as the models are trained offline and do not have support for training with 8-bit MCUs. The experimental results on fine tuning a network as well as on classification performed better with the stream of data on a device with less than 256 KiB SRAM. The other solution, Tiny Machine Learning for Concept Drift (TML-CD) [30], is mainly focused on overcoming the accuracy drop due to concept drift even under high memory constraints. This approach is composed of a feature extractor, Dimensionality Reduction operator, k-Nearest Neighbor (kNN) classifier, and an adaptation module. The adaptation module adapts the training dataset for a kNN classifier with the new data. The adaptation mechanism is carried out in three different ways: active, passive, and hybrid, among which the latter has been proved to perform better than the other two. The hybrid adaptation is a combination of both passive and active methods. It continuously adapts over time, and it discards obsolete knowledge when a change is detected in the data due to concept drift and also sets a cap on the memory footprint. This approach has been tested with MCUs with RAM as low as 96 KiB to 512 KiB and the memory footprint is kept almost constant. It performs a faster recovery when a change is detected. The experimental results showed that the hybrid approach outperforms all the other adaptation mechanisms and recovers faster when there is concept drift. This approach can be further improved by implying learning mechanisms for the feature extractor block and by exploring sparse and quantized solutions for the TinyML algorithms.

4.4. Backpropagation-Based Solutions

Previous solutions restrict training to only the last layer of the network, limiting the capacity of the model. Complex tasks, such as Computer Vision, require a stronger adaptability of the model to the changing data. Hence, training parameters relative to earlier layers in NNs, to substantially modify the model, becomes a necessity. To do so, backpropagation is required, and as a result, research interest has focused on reducing complexity and memory requirements of such algorithms to fit MCU requirements. Tiny-Transfer-Learning (TinyTL) [31] proposes freezing weights and learning the bias modules, which eliminates the need for storing the intermediate activations. It claims to have reduced the memory with little loss in accuracy compared to fine tuning the full network. Compared to fine tuning only the last layer, TinyTL performs with better accuracy with little memory overhead. The memory can be further saved by feature extraction adaptation without losing accuracy. MCUNetV3 [32] introduced the novel techniques of Quantization-Aware Scaling (QAS) and sparse layer/tensor update to cope with the challenges posed by ODL. Quantized neural networks are characterized by low bit-precision and a lack of normalization, which leads to an unstable gradient update. QAS effectively enables fully quantized training by stabilizing the gradient via a scaling factor. Full backpropagation is not possible within the limited hardware resources of MCUs. As a solution, the update of the less important parameters is skipped to reduce the memory and computation cost, i.e., sparse layer/tensor update. The sparse update scheme is determined by a contribution analysis, which maximizes, given the memory constraint, the improvements in accuracy given by updating certain parameters. Such innovations were then implemented by a specifically designed training system named "Tiny Training Engine", which prunes the backward computation graph to support sparse updates and performs operator reordering and in-place updates, thus reducing the memory footprint. MCUNetV3 is the first solution to enable the tiny on-device training of CNNs under 256 KiB SRAM and 1 MB Flash, while matching the accuracy on visual wake words. Table 2 shows a quantitative summary of

all the above-mentioned approaches, memory requirements, applications tested, target devices, and their performances.

Table 2. Overview of different ODL approaches and their performances.

Solutions	Approach	Memory Requirement	Tested Applications	Target Device	Performance
ECG learning on anomalies [25]	Reservoir computing + principal component analysis + one-class support vector machine	60 KiB RAM	ECG anomaly detection	ARM Cortex M4 and M7	Memory reduction and 95.4% accuracy on MIT-BIH
BBS-ESN [26]	Binarization + Echo state network	2176 KiB RAM	Oil leaks in wind turbines	ARM Cortex-M7	FPR = 4.1 and FNR = 14.6 in worst noise conditions
SB-DESN [27]	Field-oriented control + Extreme Learning Machine + Reservoir computing	<192 KiB RAM	Motor control system	ARM Cortex M4 and M7	Reduction of 370% in memory and 200% complexity, good accuracy in motor control
DeepESN [28]	Reservoir computing + Leaky Integrator Echo State Network	80 to 366 KiB RAM	Water distribution systems anomaly detection	ARM Cortex M7	70% memory usage reduction with good accuracy
TinyOL [29]	Additional Modifiable layer at the end of the Network	256 KiB SRAM	Fine Tuning Network and Anomaly Classification	ARM Cortex M4	Competitive performance with the model learned offline
TML-CD [30]	Hybrid Approach: Active + Passive update of KNN training set for the classifier	96 KiB to 1024 KiB SRAM	Speech command identification and image classification	ARM Cortex M4 and M7	Better accuracy and reduced memory footprint
TinyTL [31]	Freezing the weights + lite residual bias module to refine the feature extractor	37 MB	Image classification	Edge GPUs	81.4% CIFAR 100 accuracy
MCUNetV3 [32]	Quantization-Aware Scaling + Sparse layer\tensor update + Operator reordering	<256 KiB SRAM	Image classification	ARM Cortex M7	81.9% VWW accuracy

4.5. Continual Learning

An important aspect of ODL is Continual Learning (CL), the process of adaptation of the model to new data from the surrounding environment without forgetting the original knowledge, i.e., catastrophic forgetting. One approach to ensure CL has been to preserve previous samples for replaying. When it comes to the tight memory restrictions imposed by tiny hardware, however, the memory overhead for the storage of previous samples becomes unbearable. To address this problem, ref. [33] proposed the technique of Latent Replays, where intermediate feature maps of a few old data samples, instead of large data inputs, are stored for later replay, reducing up to 48× memory usage. Consequently, only layers after the layer corresponding to the feature map stored are updated. The authors of [34] developed a HW/SW platform for CL leveraging the quantization of frozen intermediate activations for Latent Replays. Combining the HW optimization, the system can achieve continual learning using less than 64 MB of memory.

4.6. Restricted Coulomb Energy Neural Network

The Restricted Coulomb Energy Neural Network (RCE-NN) is a hyper spherical classifier [35,36]. It is like a nearest-neighbor classifier, but the hyper spherical classifiers store the examples represented as points in a metric space such as Euclidean space or hamming. These metrics are a measure of the distance between an unknown input pattern of a known category or class. Every point in this space will have a finite radius associated with them. This radius defines the region of influence of that point in the space. The hypersphere's interior is the decision region associated with the center point's category.

This region of influence makes the hyper spherical classifier more conservative in terms of storage compared to the nearest-neighbor classifier. The RCE-NN, shown in Figure 2, has three layers: input, hidden, and output. For each feature, a node is assigned in the input layer, totaling the feature vector dimension. Every feature of an input vector is fully interconnected to the hidden layer. The output layer consists of as many units as the classes obtained from the network. The output of a hidden unit is projected to only one output unit. The network assigns an input pattern to a category if the output cell of that category is activated in response to the input. The network is unambiguous only if one output unit is active when an input is given, otherwise the network is ambiguous. When an input unit falls under the region of influence of a hidden unit, then the input fires the output unit to which the hidden unit is forwarded. The overlap of regions of different categories may happen. The network needs to be trained and adjusted to avoid these overlaps. The RCE-NN has two phases: learning and classification. Feature space partitioning is performed in the learning phase while updating the weights between the input and hidden layer. The threshold values of the radius of the regions for the hidden units are also adjusted in the learning phase.

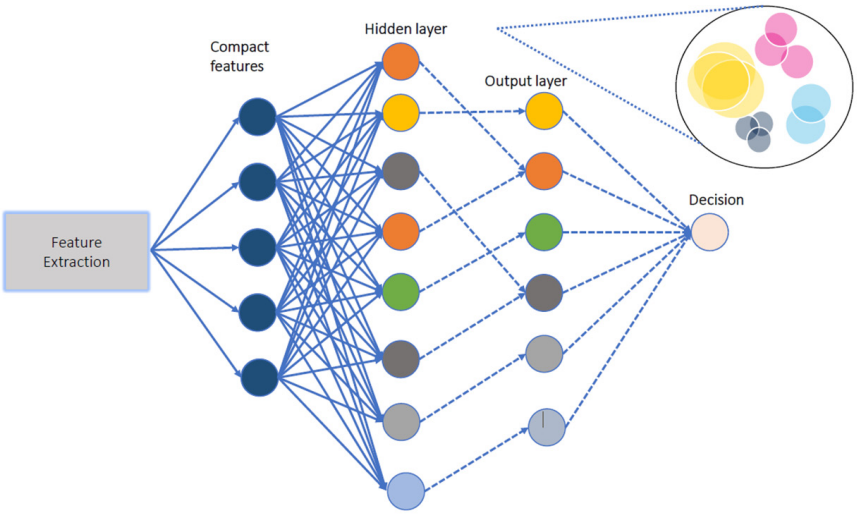

Figure 2. Restricted Coulomb Energy Neural Network with feature extractor at its input.

In the classification phase, the decisions are made for each input based on the regions that the input falls in and the feature space created by RCE-NN. Multiple class affiliations can happen in some regions. While training the network, the first step is to create hidden units for every category of the output units, and the next is to adjust the radius, which is the region of influence of those hidden units which overlaps with the other categories. Initially, the network starts with no hidden units and the data is fed into the network. One by one, the network creates a hidden unit for every input which activates an output unit. This hidden unit is created with the input vector values, and a default radius is set for the newly created hidden unit. When an input pattern activates more than one output unit, then there is an overlap which is detected by the network, and it reduces the radius of the activated hidden units associated with the output other than the ground truth. By iterating through the given number of inputs, the network learns and tries to become more accurate. RCE-NN classifies a region as an ambiguous region if there are no outputs or outputs from multiple classes. Some major advantages of RCE-NN are the conservative storage, no local minima problem, and the learning is faster.

There are very few works related to this topic; ref. [37] proposes a scheme based on RCE-NN which can be used to classify human skin color. The RCE-NN is slightly altered with an additional iteration strategy which improves the performance, and a reduction in the number of repetitive calculations was also proposed. This can help reduce the computational cost of the network. Compared with a hierarchical prototype learning

(HPL) [38] framework, it has been proved to perform better with less training time than the traditional RCE and HPL.

5. Conclusions

This paper discussed several approaches regarding (1) Neural Architecture Search for severely resource-constrained devices and with the aim to automate the design and training process of accurate ML architectures considering the technological constrains imposed by the tiny devices since the very beginning; and (2) on-device learning to cope with the concept drift and the accuracy drop in edge devices. It further discussed how these approaches can be improved in the future to achieve better performances and robust solutions. Both of these research fields are very important for TinyML, a worldwide community focused on the machine learning ecosystem for mW (and below) power consumption envelope devices and concerned with the deployment of machine learning on edge tiny devices.

Author Contributions: Conceptualization, D.P.P., P.K.A. and F.M.A.; methodology, D.P.P., P.K.A. and F.M.A.; investigation, D.P.P., P.K.A. and F.M.A.; resources, D.P.P., P.K.A. and F.M.A.; writing—original draft preparation, writing—review and editing, D.P.P., P.K.A. and F.M.A.; supervision, D.P.P.; project administration, D.P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All cited papers are published.

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

References

1. Ray, P.P. A review on tinyml: State-of-the-art and prospects. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 1595–1623. [CrossRef]
2. Banbury, C.; Reddi, V.J.; Torelli, P.; Holleman, J.; Jeffries, N.; Kiraly, C.; Montino, P.; Kanter, D.; Ahmed, S.; Pau, D.; et al. MLCommons tiny benchmark. In Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, Virtual, 28 November–9 December 2021.
3. Shawi, R.E.; Maher, M.; Sakr, S. Automated Machine Learning: State-of-The-Art and Open Challenges. *arXiv* **2019**, arXiv:1906.02287.
4. Wistuba, M.; Rawat, A.; Pedapati, T. A Survey on Neural Architecture Search. *arXiv* **2019**, arXiv:1905.01392.
5. Nayman, N.; Aflalo, Y.; Noy, A.; Zelnik-Manor, L. HardCoRe-NAS: Hard Constrained diffeRentiable Neural Architecture Search. In Proceedings of the International Conference on Machine Learning ICML, Virtual, 18–24 July 2021.
6. Jang, E.; Gu, S.; Poole, B. Categorical reparameterization with Gumbel-Softmax. In Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, 24–26 April 2017; pp. 1–12.
7. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
8. Cheng, H.-P.; Zhang, T.; Yang, Y.; Yan, F.; Teague, H.; Chen, Y.; Li, H.H. MSNet: Structural Wired Neural Architecture Search for Internet of Things. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Seoul, Republic of Korea, 27–28 October 2019; pp. 2033–2036.
9. Lu, Q.; Jiang, W.; Xu, X.; Shi, Y.; Hu, J. On Neural Architecture Search for Resource-Constrained Hardware Platforms. *arXiv* **2019**, arXiv:1911.00105.
10. López, J.G.; Agudo, A.; Moreno-Noguer, F. E-DNAS: Differentiable Neural Architecture Search for Embedded Systems. In Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 10–15 January 2021; pp. 4704–4711.
11. Perego, R.; Candelieri, A.; Archetti, F.; Pau, D. Tuning Deep Neural Network's Hyperparameters Constrained to Deployability on Tiny Systems. In Proceedings of the Artificial Neural Networks and Machine Learning–ICANN 2020: 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, 15–18 September 2020.
12. Lin, J.; Chen, W.-M.; Lin, Y.; Cohn, J.; Gan, C.; Han, S. MCUNet: Tiny Deep Learning on IoT Devices. In Proceedings of the Advances in Neural Information Processing Systems NeurIPS 2020, Vancouver, BC, Canada, 6–12 December 2020.
13. Liberis, E.; Dudziak, L.; Lane, N.D. μNAS: Constrained Neural Architecture Search for Microcontrollers. In Proceedings of the 1st Workshop on Machine Learning and Systems, Online, UK, 26 April 2021.

14. Liberis, E.; Lane, N.D. Neural networks on microcontrollers: Saving memory at inference via operator reordering. *arXiv* **2019**, arXiv:1910.05110.
15. Lin, J.; Chen, W.-M.; Cai, H.; Gan, C.; Han, S. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), Virtual, 6–14 December 2021.
16. Saha; Kusupati, A.; Simhadri, H.V.; Varma, M.; Jain, P. Rnnpool: Efficient non-linear pooling for ram constrained inference. In *Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H., Eds.; Curran Associates, Inc.: New York, NY, USA, 2020; Volume 33, pp. 20473–20484.
17. Susskind, Z.; Arden, B.; John, L.K.; Stockton, P.; John, E.B. Neuro-symbolic ai: An emerging class of ai workloads and their characterization. *arXiv* **2021**, arXiv:2109.06133.
18. Saha, S.S.; Sandha, S.S.; Aggarwal, M.; Wang, B.; Han, L.; de Gortari Briseno, J.; Srivastava, M. Tiny-NS: Platform-Aware Neurosymbolic AutoML for TinyML. *ACM Trans. Embed. Comput. Syst.* 2022; *submitted*.
19. Mahlool, D.H.; Abed, M.H. A Comprehensive Survey on Federated Learning: Concept and Applications. *arXiv* **2022**, arXiv:2201.09384.
20. Cogliati, D.; Falchetto, M.; Pau, D.; Roveri, M.; Viscardi, G. Intelligent Cyber-Physical Systems for Industry 4.0. In Proceedings of the 2018 First International Conference on Artificial Intelligence for Industries (AI4I), Laguna Hills, CA, USA, 26–28 September 2018; pp. 19–22.
21. Wang, J.; Lu, S.; Wang, S.-H.; Zhang, Y.-D. A review on extreme learning machine. *Multimed. Tools Appl.* **2021**, *81*, 41611–41660. [CrossRef]
22. Huang, G.B.; Liang, N.; Rong, H.J.; Saratchandran, P.; Sundararajan, N. On-Line Sequential Extreme Learning Machine. In Proceedings of the IASTED International Conference on Computational Intelligence, Calgary, AB, Canada, 4–6 July 2005; pp. 123–130.
23. Zhao, J.; Wang, Z.; Park, D.S. Online sequential extreme learning machine with forgetting mechanism. *Neurocomputing* **2012**, *87*, 79–89. [CrossRef]
24. Li, F.; Yang, J.; Yao, M.; Yang, S.; Wu, W. Extreme Learning Machine with Local Connections. *Neurocomputing* **2019**, *368*, 146–152. [CrossRef]
25. Abdennadher, N.; Pau, D.; Bruna, A. Fixed complexity tiny reservoir heterogeneous network for on-line ECG learning of anomalies. In Proceedings of the 2021 IEEE 10th Global Conference on Consumer Electronics (GCCE), Kyoto, Japan, 12–15 October 2021; pp. 233–237.
26. Cardoni, M.; Pau, D.P.; Falaschetti, L.; Turchetti, C.; Lattuada, M. Online Learning of Oil Leak Anomalies in Wind Turbines with Block-Based Binary Reservoir. *Electronics* **2021**, *10*, 2836. [CrossRef]
27. Federici, N.; Pau, D.; Adami, N.; Benini, S. Tiny Reservoir Computing for Extreme Learning of Motor Control. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8.
28. Pau, D.; Khiari, A.; Denaro, D. Online learning on tiny micro-controllers for anomaly detection in water distribution systems. In Proceedings of the 2021 IEEE 11th International Conference on Consumer Electronics (ICCE-Berlin), Berlin, Germany, 15–18 November 2021.
29. Ren, H.; Anicic, D.; Runkler, T.A. TinyOL: TinyML with Online-Learning on Microcontrollers. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8.
30. Disabato, S.; Roveri, M. Tiny Machine Learning for Concept Drift. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, 1–12. [CrossRef] [PubMed]
31. Cai, H.; Gan, C.; Zhu, L.; Han, S. TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning. *arXiv* **2020**, arXiv:2007.11622.
32. Lin, J.; Zhu, L.; Chen, W.-M.; Wang, W.-C.; Gan, C.; Han, S. On-device training under 256kb memory. *arXiv* **2022**, arXiv:2206.15472.
33. Pellegrini, L.; Graffieti, G.; Lomonaco, V.; Maltoni, D. Latent replay for real-time continual learning. *arXiv* **2019**, arXiv:1912.01100.
34. Ravaglia, L.; Rusci, M.; Nadalini, D.; Capotondi, A.; Conti, F.; Benini, L. A TinyML platform for on-device continual learning with quantized latent replays. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2021**, *11*, 789–802. [CrossRef]
35. Reilly, D.L.; Cooper, L.N.; Elbaum, C. A Neural Model for Category Learning. *Biol. Cybern.* **1982**, *45*, 35–41. [CrossRef] [PubMed]
36. Reilly, D.L.; Cooper, L.N. An Overview of Neural Networks: Early Models to Real World Systems. In *An Introduction to Neural and Electronic Networks*; Zornetzer, S.F., Davis, J.L., Lau, C., Eds.; Academic Press: San Diego, CA, USA, 1990.
37. Sui, C.; Kwok, N.M.; Ren, T. A Restricted Coulomb Energy (RCE) Neural Network System for Hand Image Segmentation. In Proceedings of the Canadian Conference on Computer and Robot Vision, St. Johns, NL, Canada, 25–27 May 2011; pp. 270–277.
38. Dong, G.; Xie, M. Color clustering and learning for image segmentation based on neural networks. *IEEE Trans. Neural Netw.* **2005**, *16*, 925–936. [CrossRef] [PubMed]