Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-4 (2022) 272–279

# A Practical View on Training Neural Networks in the Edge

**Marcus Rüb** * **Prof. Dr. Axel Sikora** **

* *Hahn-Schickard, Villingen-Schwenningen 78048 Germany, (e-mail: Marcus.rueb@hahn-schickard.de).*

** *Hochschule Offenburg, Offenburg, 77652 Germany, (e-mail: Axel.Sikora@hs-offenburg.de)*

**Abstract:** In recent years, the topic of embedded machine learning has become very popular in AI research. With the help of various compression techniques such as pruning, quantization and others compression techniques, it became possible to run neural networks on embedded devices. These techniques have opened up a whole new application area for machine learning. They range from smart products such as voice assistants to smart sensors that are needed in robotics. Despite the achievements in embedded machine learning, efficient algorithms for training neural networks in constrained domains are still lacking. Training on embedded devices will open up further fields of applications. Efficient training algorithms would enable federated learning on embedded devices, in which the data remains where it was collected, or retraining of neural networks in different domains. In this paper, we summarize techniques that make training on embedded devices possible. We first describe the need and requirements for such algorithms. Then we examine existing techniques that address training in resource-constrained environments as well as techniques that are also suitable for training on embedded devices, such as incremental learning. At the end, we also discuss which problems and open questions still need to be solved in these areas.



*Keywords:* Embedded Systems, Embedded AI, Edge AI, Neural networks, efficient training.

## 1. INTRODUCTION

We are surrounded by billions of edge devices with a broad variety of sensors that collect and process data, increasingly with the help of artificial intelligence (AI). Our cars are full of sensors, our smart-home devices have various sensors and just about every electrical system include some data processing.

Typical devices used for edge devices are microcontrollers (MCUs), field programmable gate arrays (FPGAs), single-board computers such as Raspberry Pis. Also smartphones can be part of the edge in a broader sense. Production and supply of MCUs shows how big the edge sector is. It is estimated that worldwide shipment of MCUs amounts to 30 billion per year until 2023 Statista (30.07.2021) and demand for MCUs is expected to grow steadily.

Often, AI is based on machine learning (ML) which based on data from the application. Currently, this data is collected and sent to a server where the ML model is trained. Unfortunately, the devices that collect the data are located in sensitive places as far as privacy is concerned - just think of the voice assistant in the living room.

Edge devices are resource-constrained systems that often run on batteries. They are designed to last for a long time with sometimes less than 0.1 W power consumption and limited resources, e.g. 64 MHz CPU frequency and 256 KB RAM. Recently, remarkable results on MCUs have been achieved, e.g. in speech and face recognition Sanchez-Iborra and Skarmeta (2020)Moons et al. (16.04.2018).

The main advantages of Edge AI relate to:

- Data security: Since no information needs to be transmitted to external environments, data security is better ensured.
- Latency: Data transmission takes time and is often associated with a delay. When this process is eliminated, the result is available immediately.
- Energy saving and cost: The transmission of information requires a server infrastructure. If there is no data transmission, energy and resources and thus costs are saved.
- No connection dependency: If the device relies on the internet to function and the connection to the internet breaks down, the data cannot be sent to the server. This for example happens, if you try to use a voice assistant and it does not respond, because it is not connected to the internet.

Since most microcontrollers do not have an operating system, several bare-metal inference frameworks have been developed to support the execution of neural networks (NN) on MCUs, while keeping computational and memory requirements low. Among them are the NN-SDK Arm Ltd. (06.06.2021) by Arm and TensorFlow Lite for Microcontrollers David et al. (17.10.2020) by Google. However, these libraries assume that the model is trained on pow-


10.1016/j.ifacol.2022.06.045

erful machines or in the cloud and then deployed to the edge device.

The MCU then only has to perform model inference. In this strategy, the model is treated as a static object Arm Ltd. (06.06.2021). In order to learn from new data, the model has to be trained in the cloud again and uploaded to the MCU again, making the use of Edge AI in an industrial environment a difficult task:

- The embedded devices operate in a distributed environment. Some devices are installed in remote and difficult-to-access locations. Considering the enormous amount of edge devices, it might be costly to update the model on each of them.
- Each system behaves differently. Component tolerances, different enviroments, different ways of operating the system, make the same ML model perform differently.
- Embedded devices have limited capabilities to store field data. Transferring field data back to the data centre is expensive and subject to delays.
- The environment is constantly changing, the performance of the ML model decrease, when the distribution of input data changes, which is called concept drift. The flexibility of the model is limited to MCUs, as the neural networks are mostly compiled graphs that are difficult to change in operation.

To overcome these challenges, algorithms need to be developed that enable to learn neural networks on the edge. This paper gives an overview of the existing algorithms and techniques that can be used for learning on the edge. A distinction is made between techniques that can be used directly on the edge and algorithms that can be used on the edge in a broader sense.

## 2. RELATED WORK

There are review papers that describe some of the topics that are only superficially considered in this paper in much more detail. In the area of incremental learning, for example Delange et al. (2021) summarizes existing techniques and compares the results. Surveys dealing with the compression of neural networks either focus on quantization and pruning Liang et al. (24.01.2021) or on knowledge distillation Gou et al. (2021). The topic of sparse-learning methods for deep neural networks is treated by Ma et al. Ma and Niu (2018).

Existing work does focus on specific techniques and their derivatives. However, there is a lack of a global view of the algorithms that apply to the edge and not, as is usually the case, to computers.

Edge or fog devices have a wide range of processing power and memory space. They range from ultra-low-power controllers like ASICs or MCUs to mobile phones or small computers like the Nvidia Jetson or a Raspberry Pi. In general, the paper will discuss learning algorithms for MCUs, FPGAs, single-board computers and smartphones. We deliberately omit ASICs and other specialized hardware, because these devices are heterogeneous and are usually only dedicated to inference of trained neural networks. Of course, some of the techniques presented here can also be applied to ASICs, but we limit ourselves to the target devices mentioned above.

## 3. REASONS FOR A SEPARATE CONSIDERATION OF TRAINING NEURAL NETWORKS ON EDGE DEVICES

Many of the techniques presented here are excellent for AI applications in the Edge. However, the methods often lack the consideration for resource-limited systems. We would like to briefly explain the reasons why a special approach is necessary and why the techniques can only be applied in a practicable way.

Training or re-training neural networks on the edge has many challenges. Most edge devices have the following limitations:

- Limited device memory, because to train a network we need to save more than just the weights and biases, e.g. activations, gradients etc.
- Limited energy resources, because complex calculations require more energy.
- Limited computing power, so that the device cannot calculate the result of the network in real time.
- Limited RAM, so that the network is too large to be loaded on the device at all.

The challenges that arise from these restrictions are:

- To store all the training data on the device.
- To train the network in a practicable time.
- To train the network with few mathematical operations.
- Not having enough RAM to load the whole model to train in on device.

These challenges must always be considered as a whole and no attempt should be made to solve them individually.

## 4. METHODS TO TRAIN NEURAL NETWORKS ON THE EDGE

In the following chapter we would like to go into the methods that are suitable for being executed in the Edge. Thereby, we go into techniques which:

- reduce the size of the neural networks itself.
- accelerate the training process of the neural networks.
- reduce the memory consumption in neural networks training.
- not need all data at the same time to train neural networks.

For this purpose, the most important publications in the field are briefly mentioned. After the short overview, we will go into one of the techniques in more detail.

### *4.1 Reduction of inference time and compression of neural networks*

Improving the energy efficient inference of deep neural networks on resource-constrained edge devices has recently attracted much attention. Starting from Han et al. (08.06.2015, 01.10.2015); Gong et al. (18.12.2014); Denton et al. (03.04.2014), one research direction focuses on the compression of pre-trained neural networks, including:

- network pruning, which removes less important neurons Han et al. (08.06.2015); Frankle and Carbin (2018) or filter channels of convolutional neural networks Liu et al. (22.08.2017);
- network quantization, which reduces the bit width of weights Han et al. (01.10.2015); Courbariaux et al. (02.11.2015) or activations Jacob et al. (16.12.2017); Wang et al. (21.11.2018);
- knowledge distillation, which uses large, well-trained networks to train smaller network architectures.

However, these techniques are limited in their application because they rely on a well-trained model for the target task as a starting point. Another line of research focuses on lightweight neural architectures that are either manually designed Sandler et al. (2019); Howard et al. (17.04.2017); Zhang et al. (2018b); Iandola et al. (24.02.2016); Huang et al. (25.11.2017) or searched for neural architectures search Tan and Le V (2020); Wu et al. (09.12.2018); Cai et al. (02.12.2018, 16.07.2017). These lightweight neural networks provide very competitive accuracies Cai et al. (26.08.2019); Tan et al. (2019), while significantly improving inference energy and memory efficiency. This also allows the network to be trained more quickly, because

(1) the forward pass is carried out faster and
(2) fewer weights and biases have to be trained

However, in terms of training memory efficiency, major bottlenecks have not been solved: the training memory is dominated by activations rather than parameters (Figure 3).

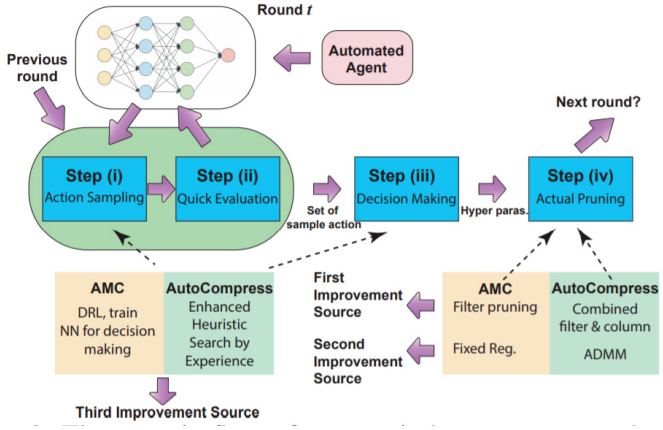


Fig. 1. The AutoCompress framework divides pruning into four steps: 1. action sampling, 2. fast action evaluation, 3. decision-making, 4. actual pruning and result generation. ( Liu et al. (06.07.2019)).

*AutoCompress* Pruning has the advantage of being fairly easy to apply to any network architecture and any target platform can benefit from it using intelligent execution algorithms. The work of Liu et al. Liu et al. (06.07.2019) presents a framework for automatic pruning of neural networks. In a pre-trained deep neural network (DNN), the automatic hyperparameter determination process determines the pruning rate of the weights per layer and the type of structured pruning scheme per layer. The goal is to maximize the reduction in the number of weights or floating-point operations per second (FLOPs), with minimal loss of accuracy. The optimum would be to accelerate the network without an accuracy loss. Since this is usually not possible, a trade off must be handelt.

The framework divides pruning into four steps (Fig. 1):

(1) action sampling,
(2) fast action evaluation,
(3) decision-making,
(4) actual pruning and result generation.

Step 1 calculates several pruning candidates which are evaluated in step 2. Because both steps must be performed as quickly as possible to keep the framework practicable, only simple pruning techniques such as magnitude pruning can be used. In step 3, the pruning hyperparameters are determined. The determination of the hyperparameters is based on the information collected in step 2 where the action samples are evaluated. Step 4 generates the pruning result, using the Alternating Direction Methods of Multipliers (ADMM) based structured weight pruning algorithm Zhang et al. (2018a). Here, the pruning algorithm can be more complicated and powerful as it is performed only once in each round. The whole automatic process is iterative, and steps 1 to 4 above reflect only one round. The reason for this is that it is difficult to look for high pruning rates in a single round and the whole pruning process of the weights is iterative. The number of rounds is given by the authors between 4 and 8 rounds to allow a fair comparison to other pruning frameworks and techniques. One advantage of AutoCompress is that it supports a flexible number of iterative rounds to achieve a trade off between accuracy and FLOP reduction. The authors report that they are at the SOTA level of pruning with this technology and can perform compression rates of up to 33 times.

*Further improvements* The techniques presented are already very well researched and widespread for use in the Edge. However, it should be mentioned that the success of pruning in terms of speedup and memory size always depends on several points. Often, unstructured pruning is used as a pruning technique, which only ”deletes” individual connections instead of entire neurons. In order to get an advantage with this technique, an intelligent memory handling and calculation algorithm is needed to avoid the 0 multiplication.

### *4.2 Efficient training algorithms*

Training neural networks tends to be time-consuming Jean et al. (05.12.2014), especially for architectures with a large number of trainable model parameters. An important reason why neural network training is typically slow, is that backpropagation requires the computation of full gradients and updates all parameters in each learning step Sun et al. (20.06.2017). As deep neural networks with a lot of parameters become more prevalent, more efforts are being made to speed up the process of backpropagation. An emerging research direction to accelerate backpropagation is sparse backpropagation Sun et al. (20.06.2017); Wei et al. (18.09.2017); Zhu et al. (01.06.2018), which aims to sparsify the full gradient vector to achieve significant computational cost savings. An effective solution to sparse backpropagation is top-k sparsity, which keeps only k

elements with the largest absolute values in the gradient vector and backpropagates them across different layers. For example, me-Prop Sun et al. (20.06.2017) uses top-k sparseness to compute only a very small but critical part of the gradient information and updates the corresponding model parameters. Going one step further, Wei et al. (18.09.2017) implements top-k sparseness for backpropagation of convolutional neural networks. Experimental results show that these methods can significantly speed-up the backpropagation process. However, despite the success in saving computational cost, top-k sparsity for backpropagation still suffers from some hard-to-fix drawbacks, which are explained in more detail below. On the theoretical side, the theoretical properties of sparse backpropagation, especially for top-k sparsity Sun et al. (20.06.2017, 2020); Wei et al. (18.09.2017), have not been fully explored yet.

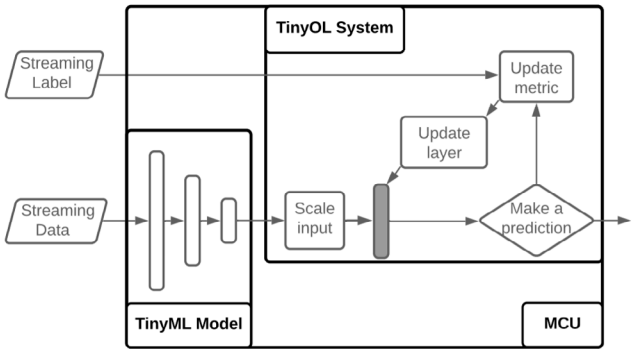


Fig. 2. The building blocks of TinyOL on MCUs ( Ren et al. (15.03.2021))

*TinyOL* An extreme variant of sparse backpropagation is the TinyOL technique Ren et al. (15.03.2021). Here, only the very last layer of a neural network is retrained. In most frameworks NNs are uploaded as a C array into the flash of the edge device, they are treated as a frozen graph and cannot be changed afterwards. Nevertheless, in order to re-train the network, Haoyu Ren et al. Ren et al. (15.03.2021) have come up with TinyOL. The core component of TinyOL is the additional layer marked in black in Fig. 2. This additional layer consists of multiple neurons that can be adapted, initialized and updated on the fly. This concept is similar to transfer learning, where part of a pre-trained model is fixed and fine-tuning takes place in the final layers. With the online learning architecture, the additional layer can learn from streaming data.

At each time, new sample data first flows through the existing NN and is then fed into TinyOL. Depending on the task, the accumulated mean and variance are updated, and the input can be standardized. The system then performs model inference. If an appropriate label is available, the evaluation metrics and the weights in the additional layer will be adapted using online gradient descent algorithms, e.g. stochastic gradient descent (SGD). In this way, the steps of training and prediction are interleaved, first the label is generated by the prediction and then the model is re-trained with the data point. Once the neurons are updated, the sample pairs can be discarded. In other words, at any one time, there is only one data pair of the data stream in memory, and there is no need to store the historical data. Compared to the batch/offline training setting, TinyOL can be trained with minimal resource consumption, making in-device training on massive streaming data possible. With this design, models can be fitted to a specific field, as the layer in TinyOL can be trained in real time as the streaming field data arrives. The consequence is that the model is robust to concept drift, which means that the statistical properties of the field data may change over time. Without retraining, the model's performance drops significantly because the model cannot anticipate these changes during the training phase.

*Further improvements* The presented sparse backpropagation algorithms have proven that the computation time can be shortened by omitting computational operations. However, there are no considerations for memory minimization or whether training up to the first layer is necessary for training or retraining. Many techniques also lack consideration of how sparsity affects the execution speed in parallel or serial computing units. Furthermore, the combination of these techniques with, for example, incremental learning is a promising idea that (1) shortens training time and (2) can significantly reduce memory.

### *4.3 Reduced memory footprint during training*

Researchers have been looking for ways to reduce the training memory footprint. A typical approach is to recalculate discarded activations during backward computation Gruslys et al. (10.06.2016); Chen et al. (21.04.2016). This approach reduces the memory footprint at the cost of a large computational overhead. Therefore, it is not preferred for edge devices. Layer-wise training Greff et al. (22.12.2016) can also reduce memory requirements compared to end-to-end training. However, it cannot achieve the same level of accuracy as end-to-end training, because not all parameters are updated at the same time and thereby the dependencies of the weights are lost. Another representative approach is activation pruning Liu et al. (01.10.2018), which builds a dynamic sparse computation graph to prune activations during training. Similarly, Wang et al. (19.12.2018) proposes to reduce the bit-width of training activations by introducing new floating-point formats with reduced precision. In addition to reducing training memory costs, there are some techniques that focus on reducing peak inference memory costs, such as RNNPool Saha et al. (27.02.2020) and MemNet Liu et al. (22.07.2019).

An interesting method of training a network using low-bit operations was presented by De Sa et al. de Sa et al. (09.03.2018). They show that it is still possible to obtain accurate solutions from low accuracy training as long as the problem is sufficiently well conditioned. They do this with an algorithm called HALP, which overcomes the accuracy limitations of ordinary low accuracy SGD. Gradient variance noise is handled with a well-known technique called SVRG (stochastic variance-reduced gradient) Johnson and Zhang (2013). To address the noise from quantization, they introduce a new technique called bit centering. The intuition behind bit centering is that, as

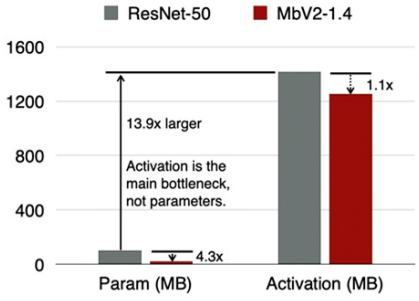


Fig. 3. Memory cost comparison between ResNet-50 and MobileNetV2-1.4 under batch size 16. Recent advances in efficient model design only reduce the size of parameters, but the activation size, which is the main bottleneck for training, does not improve much. (Cai et al. (22.07.2020))

the gradient approaches the optimum, it becomes smaller and in a sense carries less information, so we should be able to compress it. By dynamically recentering and re-scaling our low-precision numbers, we can asymptotically reduce the quantization noise as the algorithm converges. They prove that for strongly convex problems, HALP is able to produce arbitrarily accurate solutions with the same linear asymptotic convergence rate as SVRG, while using low-precision iterates with a fixed number of bits.

*TinyTL* Cai et al. Cai et al. (22.07.2020) present an interesting way of retraining networks in the field. Their method of making only the bias trainable greatly reduces memory requirements during training and thereby accelerate the training extremely. By analyzing the memory requirements during backpropagation, Cai et al. Cai et al. (22.07.2020) found that the intermediate activations (the biggest bottleneck) are only needed when updating the weights, but not when updating the biases. Inspired by this finding, they proposed to freeze the weights of the pre-trained network and to only update the biases to reduce the memory requirement (Figure 4b). To compensate for the loss of capacity, they introduce a memory-efficient bias module, called the lite-residual module, which improves model capacity by refining the intermediate feature maps of the feature extractor (Figure 4c). At the same time, the resolution and width of the lite-residual module are greatly reduced to have a low memory overhead of 3.8%. Extensive experiments on nine image classification datasets with the same pre-trained model (ProxylessNAS-Mobile Cai et al. (02.12.2018)) demonstrate the effectiveness of TinyTL compared to previous transfer learning methods. Moreover, in combination with a pre-trained once-for-all network Cai et al. (26.08.2019), TinyTL can select a specialized sub-network as a feature extractor for each transfer dataset (i.e. adaptation of the feature extractor): a larger sub-network is selected for a more difficult dataset and vice versa. TinyTL achieves the same (or even higher) accuracy compared to fine-tuning the full Inception-V3, while reducing training memory requirements by up to 12.9×.

*Further improvements* Tinytl is a technique that delivers impressive results for memory footprint reduction. Unfortunately, this technique is also very limited when it comes to adapting to completely new tasks or to highly dynamic environments. For example, a combination with other techniques could remedy this negative point.

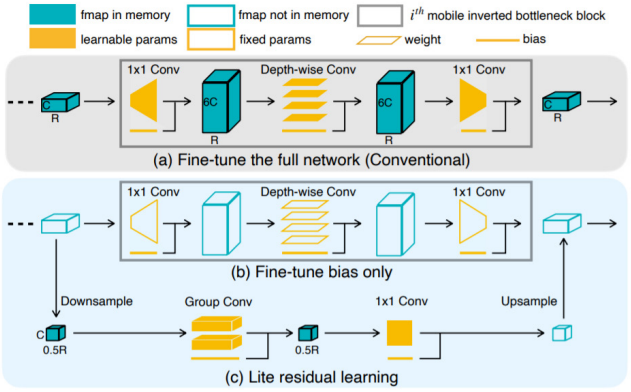


Fig. 4. TinyTL overview ("C" denotes the width and "R" denote the resolution). Conventional transfer learning relies on fine-tuning the weights to adapt the model (Fig.a), which requires a large amount of activation memory (in blue) for back-propagation. TinyTL reduces the memory usage by fixing the weights (Fig.b) while only fine-tuning the bias. (Fig.c) exploit lite residual learning to compensate for the capacity loss, using group convolution and avoiding inverted bottleneck to achieve high arithmetic intensity and small memory footprint. The original skip connection remains unchanged (omitted for simplicity). ( Cai et al. (22.07.2020))

A list of other ideas that could significantly improve existing methods:

- Develop efficient algorithms to enable full-size training on the device. The shown algorithms usually train only parts of the neural network, TinyOl only one layer, tinytl only the biases. However, algorithms are still needed that can train the entire network quickly and with low memory consumption.
- Develop efficient algorithms for further optimizers and neural network operations, e.g. recurrent neural networks, or spiking neural networks. Most of the work has focused on dense and convolutional networks, but mobile algorithms are also needed for transformer or recurrent networks, for example.
- Develop efficient algorithms for online training. It would be conceivable, for example, to combine efficient learning algorithms with incremental learning techniques.
- Explore training with reduced numerical accuracy, e.g. on 8-bit MCUs.
- Develop efficient algorithms for federated learning for distributed IoT devices.

### *4.4 Incremental learning*

Continual learning explores the problem of learning from an infinite stream of data with the goal of incrementally expanding acquired knowledge and using it for future learning Chen and Liu (2016). This approach is relevant for learning on embedded devices, because the training data does not have to be stored completely in the limited mem-

ory. Instead, the data is used as it occurs. The data may come from changing input domains (e.g. varying imaging conditions) or be associated with different tasks (e.g. fine-grained classification problems). Continuous learning is also referred to as lifelong learning Chen and Liu (2016); Parisi et al. (21.02.2018), sequential learning Aljundi et al. (14.06.2018); Shin et al. (24.05.2017) or incremental learning Ganea et al. (11.05.2021); Zhang et al. (07.04.2021); Liu et al. (10.10.2020). The main criterion is the sequential nature of the learning process, with only a small portion of the input data from one or a few tasks available at a time. The main challenge is learning without catastrophic forgetting: Performance on a previously learned task or domain should not significantly deteriorate over time as new tasks or domains are added. This is a direct consequence of a more general problem in neural networks, namely the stability-plasticity dilemma Grossberg (1982), where plasticity refers to the ability to integrate new knowledge and stability retains previous knowledge during encoding. Although it is a challenging problem, advances in continuous learning have led to the emergence of the first real-world applications Lange et al. (2020). To get a deeper insight into incremental learning, several surveys have been written about the results of state-of-the-art techniques. Masana et al. (29.10.2020); Delange et al. (2021); Parisi et al. (21.02.2018); Belouadah et al. (03.11.2020)

*incremental learning* A method that addresses incremental learning directly on an embedded device is presented by Abdul Qader et al. AbdulQader et al. (25.03.2021). Their method uses an evolutionary strategy (ES) based technique to perform incremental training without the need to use backpropagation or gradient calculation. Due to the omission of the backpropagation algorithm, the same forward pass algorithm can be used, which in the case of an FPGA can be executed significantly faster with lower hardware consumption. In their algorithm, a subset of the model weights or the weights of a particular layer are selected for training. At each iteration, the loss function for each generated population is evaluated over the entire training data, which is saved on the embedded device. The loss function used in their method is the negative of the mean absolute error (MAE). This was chosen because it is computationally less expensive than other loss functions such as mean squared error (MSE) or root mean squared error (RMSE). No multiplication or square root is required, making MAE easier to implement on a device with minimal resources. This technique is an interesting approach to enable training on FPGAs without increasing the hardware complexity. While the backpropagation algorithm is more accurate, it provides a solution to train a network when backpropagation is not possible.

*Further improvements* Incremental learning is very suitable for use in the edge and will realistically have to be used in practice. Nevertheless, there are open questions around the topic of incremental learning:

- How can incremental learning be combined with online learning methods to further reduce data storage?
- The retention of previously learned knowledge must be made more stable.
- To what extent is incremental learning also practicable for other types of networks and problems apart from classification problems?
- When does it make sense to forget previously learned knowledge? And how is this to be handled?

## 5. SUMMARY

In this paper, different techniques to train neural networks in resource limited environments were reviewed. Starting with how to decrease the inference time of neural networks, which is also useful in the forward pass of their training. After that we show techniques how to speed up the backpropagation algorithm by e.g. sparse backpropagation. Besides accelerating neural networks, their memory footprint is also a point of attack, which was solved by tinyTL. Finally, we look at incremental learning, as this technique can reduce the memory needed to store training data.

## REFERENCES


AbdulQader, D., Krishnan, S., and Coelho, JR, C.N. (25.03.2021). Enabling incremental training with forward pass for edge devices. URL https://arxiv.org/pdf/2103.14007.

Aljundi, R., Rohrbach, M., and Tuytelaars, T. (14.06.2018). Selfless sequential learning. URL https://arxiv.org/pdf/1806.05421.

Arm Ltd. (06.06.2021). Arm nn sdk. URL https://www.arm.com/ products/ silicon-ip-cpu/ ethos/arm-nn.

Belouadah, E., Popescu, A., and Kanellos, I. (03.11.2020). A comprehensive study of class incremental learning algorithms for visual tasks. URL https://arxiv.org/pdf/2011.01844.

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. (16.07.2017). Efficient architecture search by network transformation. URL https://arxiv.org/pdf/1707.04873.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. (26.08.2019). Once-for-all: Train one network and specialize it for efficient deployment. URL https://arxiv.org/pdf/1908.09791.

Cai, H., Gan, C., Zhu, L., and Han, S. (22.07.2020). Tinytl: Reduce memory, not parameters for efficient on-device learning. URL https://arxiv.org/pdf/2007.11622.

Cai, H., Zhu, L., and Han, S. (02.12.2018). Proxylessnas: Direct neural architecture search on target task and hardware. URL https://arxiv.org/pdf/1812.00332.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. (21.04.2016). Training deep nets with sublinear memory cost. URL https://arxiv.org/pdf/1604.06174.

Chen, Z. and Liu, B. (2016). Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(3), 1–145. doi: 10.2200/S00737ED1V01Y201610AIM033.

Courbariaux, M., Bengio, Y., and David, J.P. (02.11.2015). Binaryconnect: Training deep neural networks with binary weights during propagations. URL https://arxiv.org/pdf/1511.00363.

David, R., Duke, J., Jain, A., Reddi, V.J., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T., and Warden, P. (17.10.2020). Tensorflow lite micro: Em-

bedded machine learning on tinyml systems. URL https://arxiv.org/pdf/2010.08678.

de Sa, C., Leszczynski, M., Zhang, J., Marzoev, A., Aberger, C.R., Olukotun, K., and Ré, C. (09.03.2018). High-accuracy low-precision training. URL https://arxiv.org/pdf/1803.03383.

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP, 1. doi: 10.1109/TPAMI.2021.3057446.

Denton, E., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. (03.04.2014). Exploiting linear structure within convolutional networks for efficient evaluation. URL https://arxiv.org/pdf/1404.0736.

Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. URL http://arxiv.org/pdf/1803.03635v5.

Ganea, D.A., Boom, B., and Poppe, R. (11.05.2021). Incremental few-shot instance segmentation. URL https://arxiv.org/pdf/2105.05312.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. (18.12.2014). Compressing deep convolutional networks using vector quantization. URL https://arxiv.org/pdf/1412.6115.

Gou, J., Yu, B., Maybank, S.J., and Tao, D. (2021). Knowledge distillation: A survey. doi:10.1007/s11263-021-01453-z. URL https://arxiv.org/pdf/2006.05525.

Greff, K., Srivastava, R.K., and Schmidhuber, J. (22.12.2016). Highway and residual networks learn unrolled iterative estimation. URL https://arxiv.org/pdf/1612.07771.

Grossberg, S. (1982). *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition, and Motor Control*, volume 70 of *Boston Studies in the Philosophy of Science*. Springer Netherlands, Dordrecht. doi:10.1007/978-94-009-7758-7.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., and Graves, A. (10.06.2016). Memory-efficient backpropagation through time. URL https://arxiv.org/pdf/1606.03401.

Han, S., Mao, H., and Dally, W.J. (01.10.2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. URL https://arxiv.org/pdf/1510.00149.

Han, S., Pool, J., Tran, J., and Dally, W.J. (08.06.2015). Learning both weights and connections for efficient neural networks. URL https://arxiv.org/pdf/1506.02626.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (17.04.2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. URL https://arxiv.org/pdf/1704.04861.

Huang, G., Liu, S., van der Maaten, L., and Weinberger, K.Q. (25.11.2017). Condensenet: An efficient densenet using learned group convolutions. URL https://arxiv.org/pdf/1711.09224.

Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., and Keutzer, K. (24.02.2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. URL https://arxiv.org/pdf/1602.07360.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (16.12.2017). Quantization and training of neural networks for efficient integer-arithmetic-only inference. URL https://arxiv.org/pdf/1712.05877.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (05.12.2014). On using very large target vocabulary for neural machine translation. URL https://arxiv.org/pdf/1412.2007.

Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, 315–323. Curran Associates Inc, Red Hook, NY, USA.

Lange, M.D., Jia, X., Parisot, S., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2020). Unsupervised model personalization while preserving privacy and scalability: An open problem. URL http://arxiv.org/pdf/2003.13296v1.

Liang, T., Glossner, J., Wang, L., and Shi, S. (24.01.2021). Pruning and quantization for deep neural network acceleration: A survey. URL https://arxiv.org/pdf/2101.09671.

Liu, L., Deng, L., Hu, X., Zhu, M., Li, G., Ding, Y., and Xie, Y. (01.10.2018). Dynamic sparse graph for efficient deep learning. URL https://arxiv.org/pdf/1810.00859.

Liu, N., Ma, X., Xu, Z., Wang, Y., Tang, J., and Ye, J. (06.07.2019). Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates. URL https://arxiv.org/pdf/1907.03141.

Liu, P., Wu, B., Ma, H., and Seok, M. (22.07.2019). Memnet: Memory-efficiency guided neural architecture search with augment-trim learning. URL https://arxiv.org/pdf/1907.09569.

Liu, Y., Schiele, B., and Sun, Q. (10.10.2020). Adaptive aggregation networks for class-incremental learning. URL https://arxiv.org/pdf/2010.05063.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (22.08.2017). Learning efficient convolutional networks through network slimming. URL https://arxiv.org/pdf/1708.06519.

Ma, R. and Niu, L. (2018). A survey of sparse-learning methods for deep neural networks. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence*, 647–650. IEEE, Piscataway, NJ. doi: 10.1109/WI.2018.00-20.

Masana, M., Liu, X., Twardowski, B., Menta, M., Bagdanov, A.D., and van de Weijer, J. (29.10.2020). Class-incremental learning: survey and performance evaluation on image classification. URL https://arxiv.org/pdf/2010.15277.

Moons, B., Bankman, D., Yang, L., Murmann, B., and Verhelst, M. (16.04.2018). Binareye: An always-on energy-accuracy-scalable binary cnn processor with all memory on chip in 28nm cmos. URL https://arxiv.org/pdf/1804.05554.

Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., and Wermter, S. (21.02.2018). Continual lifelong learning with neural networks: A

review. doi:10.1016/j.neunet.2019.01.012. URL https://arxiv.org/pdf/1802.07569.

Ren, H., Anicic, D., and Runkler, T. (15.03.2021). Tinyol: Tinyml with online-learning on microcontrollers. URL https://arxiv.org/pdf/2103.08295.

Saha, O., Kusupati, A., Simhadri, H.V., Varma, M., and Jain, P. (27.02.2020). Rnnpool: Efficient non-linear pooling for ram constrained inference. URL https://arxiv.org/pdf/2002.11921.

Sanchez-Iborra, R. and Skarmeta, A.F. (2020). Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3), 4–18. doi:10.1109/MCAS.2020.3005467.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.C. (2019). Mobilenetv2: Inverted residuals and linear bottlenecks. URL http://arxiv.org/pdf/1801.04381v4.

Shin, H., Lee, J.K., Kim, J., and Kim, J. (24.05.2017). Continual learning with deep generative replay. URL https://arxiv.org/pdf/1705.08690.

Statista (30.07.2021). Global microcontroller unit shipments 2015-2023 | statista. URL https://www.statista.com/statistics /935382/worldwide-microcontroller-unit-shipments/.

Sun, X., Ren, X., Ma, S., and Wang, H. (20.06.2017). meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. URL https://arxiv.org/pdf/1706.06197.

Sun, X., Ren, X., Ma, S., Wei, B., Li, W., Xu, J., Wang, H., and Zhang, Y. (2020). Training simplification and model simplification for deep learning: A minimal effort back propagation method. doi:10.1109/TKDE.2018.2883613. URL https://arxiv.org/pdf/1711.06528.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le V, Q. (2019). Mnasnet: Platform-aware neural architecture search for mobile. URL http://arxiv.org/pdf/1807.11626v3.

Tan, M. and Le V, Q. (2020). Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning*. URL http://arxiv.org/pdf/1905.11946v5.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. (21.11.2018). Haq: Hardware-aware automated quantization with mixed precision. URL https://arxiv.org/pdf/1811.08886.

Wang, N., Choi, J., Brand, D., Chen, C.Y., and Gopalakrishnan, K. (19.12.2018). Training deep neural networks with 8-bit floating point numbers. URL https://arxiv.org/pdf/1812.08011.

Wei, B., Sun, X., Ren, X., and Xu, J. (18.09.2017). Minimal effort back propagation for convolutional neural networks. URL https://arxiv.org/pdf/1709.05804.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. (09.12.2018). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. URL https://arxiv.org/pdf/1812.03443.

Zhang, C., Song, N., Lin, G., Zheng, Y., Pan, P., and Xu, Y. (07.04.2021). Few-shot incremental learning with continually evolved classifiers. URL https://arxiv.org/pdf/2104.03047.

Zhang, T., Ye, S., Zhang, K., Tang, J., Wen, W., Fardad, M., and Wang, Y. (2018a). A systematic dnn weight pruning framework using alternating direction method of multipliers. *ECCV*, 11212(1), 191–207. doi:10.1007/978-3-030-01237-3_12. URL https://arxiv.org/pdf/1804.03294.

Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018b). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2018)*, 6848–6856. IEEE, Piscataway, NJ. doi:10.1109/CVPR.2018.00716.

Zhu, M., Clemons, J., Pool, J., Rhu, M., Keckler, S.W., and Xie, Y. (01.06.2018). Structurally sparsified backward propagation for faster long short-term memory training. URL https://arxiv.org/pdf/1806.00512.