Neural Architecture Search for Automated Machine Learning Deployment on Extreme Edge Devices

Philipp van Kempen, Technische Universität München, Munich, Germany

***Abstract*—Machine Learning and Artificial Intelligence are omnipresent topics nowadays. Especially since small personal devices, such as smartphones, are capable of running this computation intensive tasks on their own, they can improve our lives in many different ways, e.g. computer vision or speech detection. Deploying neural network models on tiny and therefore performance-, memory- and power-constrained devices, such as microcontrollers, is currently an important research topic. For mobile and other embedded target devices Neural Architecture Search (NAS) is a new method to find appropriate network models without requiring manual error-prone adjustments by engineers, which allows to automate the workflow from the definition of a real work problem to a deployable solution [1]. While there are a lot of comprehensive surveys on NAS methods in general [1], [2], none of them considers the application of those ideas on edge devices like microcontrollers (MCUs). This work provides an overview of the recent developments and compares different approaches, which appeared over the last few years, against one another and with well-known reference applications intended for the usage on mobile phones. In the end some opportunities for future work on this topic are discussed.**

I. Introduction

The training of large machine learning models is typically done on specific high-performance computers, but may still require a lot of time. This makes is hard to tune hyper-parameters or the model architecture in an intensive way. Therefore a way to reduce the design time human interaction and know-how is needed. This is not the case for smaller models targeting embedded systems as their architecture is often much simpler resulting in training times of just a few minutes for mid-size applications. Finding the best suitable model architecture to run on edge devices, which are known to have several resource constraints like computational power and memory size with typical values in the range of several KB to very few MB, is currently a highly relevant topic being explored by many researchers worldwide. After a brief introduction of some core concepts of Machine Learning like NAS, AutoML and available Frameworks, a survey on papers on the aforementioned subject follows in section II. The benchmark results for these approaches are compared as far as possible in section III before the core results and ideas for further work on the topics are summarized in section IV.

A. Neural Architecture Search and AutoML

Short training times enable the application of optimization techniques to either improve the accuracy or the size of the resulting models in a meaningful way. Especially Neural Architecture Search methods benefit of such targets as they allow many design iterations and therefore more promising results in shorter time. AutoML workflows, which combine NAS with further automated process steps as shown in figure 1, are already being used in research groups and will become more relevant for the future AI industry [3]. While Data Preparation and Feature Engineering are prerequisite steps to apply NAS, automated machine learning may make full use of potential of Neural Architecture Search during the generation and estimation of models. Search space exploration and optimization methods like heuristics or evolutionary algorithms may be used to gain architectures fulfilling the given constraints. There are many methods which can be used for model compression in the final estimation step, where weight-sharing is one of the most relevant ones. In the method of weight-sharing a larger super-network is typically trained to provide a higher number of possible sub-networks for evaluation. This reduces the search cost as the parameters of overlapping sub-networks are only computed once [4]. Other approaches to reduce the model complexity are the quantization of floating point model state values and parameters to smaller fixed-point data types or pruning which removes a certain amount of parameters determined by a given pruning method [5].

B. Frameworks

To design machine learning models and deploy them on a target device some framework or library is typically used. As later shown in this paper, this choice can have a large impact on the performance and size of the resulting design. Especially the “engine” used for running the inference on a microcontroller device may be a bottleneck or provide some room for improvements. While Tensorflow Lite (TFLite) mostly targets mobile devices, there is a relatively novel but popular library called Tensorflow Lite Micro (TFLM) with kernels, e.g. methods to execute operations of the model graph, specialized to resource constrained architectures [6]. The term TinyML is often used in the context of TFLM but may also refer to machine learning models running on edge devices in general. Custom kernels often referred as CMSIS-NN exist for some targets with hardware accelerators for typical Digital Signal Processing (DSP) instructions using neural network calculations like Multiply and Accumulate [7]. The data in table I also presents the relevance of both frameworks compared to less widely used MicroTVM and Pytorch, which are not going to be discussed here. Each completely custom framework proposed in a paper has to be compared to the aforementioned state of the art approaches.

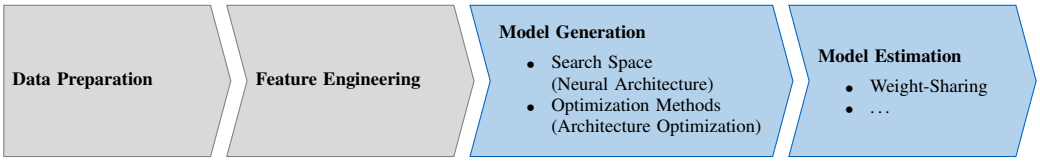

Fig. 1. Typical AutoML Flow [3] (Highlighted Steps are relevant for NAS)

TABLE I
FRAMEWORKS USED IN THE DISCUSSED PAPERS (○: COMPARISONS ONLY)

Frameworks	TFLite (Micro)	CMSIS-NN	MicroTVM	Pytorch	Custom
MCUNet[8]	○	○	○		●
µNAS[9]	●	●			
SpArSe[10]	N/A	N/A	N/A	N/A	N/A
MicroNets[11]	●	●			
Once-for-All[12]				●	

II. State of the Art

Although the following brief overview on research papers focuses on the proposed NAS methods instead of the inference engine or model types, the framework used for the design and deployment sometimes also need consideration due to specific model-optimizations, which may only have certain effects when applied together with the suitable counterparts.

Beside frameworks there exist many research results on NAS performance for popular reference models like ImageNet [13] and CIFAR-10 [14]. Unfortunately they are initially targeted to devices offering more computing power. Due to their large size or operation count (computational demand) not all of them are suitable for comparing NAS results on microcontrollers. To demonstrate the possibilities of machine learning on edge devices some popular application spaces like Image Recognition (ImageNet), Visual Wake Words and Speech Commands (Keyword Spotting, KWS), can be used. There also exist many datasets for the classification of handwritten characters or digits (MNIST). Applications like Anomaly Detection or DSP Classification have higher relevance for the real world but often lack of representative datasets, hence why they are only used in some of the research papers.

A. MCUNet

The MCUNet system-model co-design framework consists of two main components. The first one is a Neural Architecture Search algorithm suitable for MCUs and a lightweight inference engine based on code generation. The latter is called TinyEngine and profits of memory scheduling, which takes the overall network topology into account instead of only considering individual layers for optimization. This leads to smaller models in terms of memory requirements and operation count and therefore to an increase of the size of the NAS search space. The two-stage algorithm of the so called TinyNAS component optimizes the aforementioned search space according to given resource constraints before specializing its properties to handle various constraints at once [8]. Inference results in comparison with the state of the art Tensorflow Lite Micro library and CMSIS-NN [7] are presented in section III.

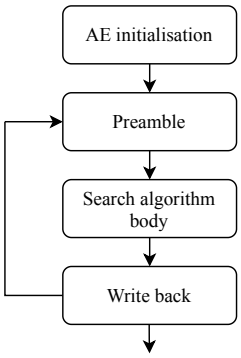

Fig. 2. General search procedure for μNAS [9]

TinyNAS at first searches for the best search space configuration $S*$ in the set of possible configuration S which is a non-trivial task. By using the cummulative-distribution-function (CDF) of FLOPS instead of the CDF of model accuracy the high cost of training each network can be avoided. One-shot architecture search making use of weight sharing in combination with evolutionary search determines the resulting model suitable for the given set of constraints.

By performing in-place depth-wise convolutions using a single temporal buffer, TinyEngine can reduce the peak memory for N channels from $2N$ to $N+1$. Additionally methods such as adaptive scheduling or loop unrolling are applied on parts of the model architecture during code generation to reduce the number of required operations for running a neural network model. [8]

B. µNAS

Similar to TinyNAS, μNAS generates mid-tier MCU-level networks with sizes from 0.5 KB to 64 KB, which are primarily intended for image classification tasks. Multi-objective optimization takes RAM-size, persistent storage and processor speed into account [9]. The experiment results in the recently published paper already featuring comparisons with for example the MCUNet framework.

Different to traditional approaches for GPU- or mobile-level NAS design, μNAS was designed following two special design requirements: A highly granular search space and the accurate computation of used resources. The resulting four-step search procedure is based on the concepts of aging evolution and pruning and is briefly shown in figure 2.

C. SpArSe

To demonstrate the possibility of designing Convolutional Neural Networks (CNNs) which generalize well via AutoML, while still being small enough to fit onto memory-constrained MCUs, [10] proposes an architecture search method combining neural architecture search with pruning. The resulting applications for the IoT field definitely display a success of the aforementioned challenge.

Using a balanced multi-objective optimization problem operating on the problem space $\Omega = \{\alpha, \theta, \omega, \Theta\}$ composed of the network connectivity graph α, network weights ω, operations θ and training hyper-parameters Θ small but performant CNNs can be generated. The three objective functions

- $f_1(\Omega) = 1 - \text{VALIDATIONACCURACY}(\Omega)$
- $f_2(\Omega) = \text{MODELSIZE}(\Omega)$
- $f_3(\Omega) = \max_{l \in 1,\dots,L} \text{WORKINGMEMORY}_l(\Omega)$

are used to design the SpArSe architectures with L being the number of network layers. After a model size compression via pruning the best suitable sub-spaces are determined using Bayesian optimization. Instead of weight-sharing, Network morphism is applied in combination with random scalarizations [10] to evaluate each configuration Ω^n at low computational costs.

D. MicroNets

To employ a new differentiable NAS (DNAS) realization for edge devices with tight memory and performance constraints in [11] the properties of NAS search spaces for MCU model design were studied first. The most important result of this step is the correlation between the model latency and the model operation count under a uniform prior over models in the search space. Therefore the op count is used as viable proxy to latency when searching for the optimal network architecture. Designed models can target three different size-classes (S/M/L). Another aspect of NAS is the underlying latency/energy model which is also mentioned in [11]. TinyML performance for industry standard benchmarks is shortly mentioned in III.

After deriving the correlation between the layer operation count and the layer latency and the energy model the paper proposes MircoNet Models following a DNAS approach. Instead of using model constraints to restrict the feasible space, regularization terms are incorporated in the DNAS objective function. Compression techniques such as 'Sub-Byte' quantization follow. [11]

E. Once-for-All (OFA)

While the generation and inference of models especially designed for microcontrollers was the main objective in the previously mentioned works, the Once-for-All approach [12] is built to design optimal networks suitable for a wide range of devices. Usually this is a costly goal as training one model for each architectural settings is required. By decoupling the training and search, it is possible to get a sub network which can be then optimized in multiple dimensions like depth, width and kernel size. As a novel progressive shrinking algorithm, a generalized pruning algorithm is proposed and state-of-the-art performance especially for the mobile setting presented.

The following special CNN techniques are applied in the OFA search space generation process to create a single large Once-for-All network.

- **Elastic depth:** arbitrary numbers of layers per unit
- **Elastic width:** arbitrary numbers of channels per layer
- **Elastic kernel size:** arbitrary kernel sizes
- **Elastic resolution:** arbitrary input image sizes

Smaller sub-networks can make use of this trained OFA network by exploiting weight sharing and progressive shrinking. In the deployment stage no additional expensive training time is required due to the decoupling. [12]

III. COMPARISON

There are a lot of metrics to evaluate how the proposed NAS methods improve or degrade the performance of the resulting models. Often the model accuracy after deployment and the size of the generated program are primarily considered because of the given tight memory constrains in terms of program memory and SRAM usage on edge devices. If inference latency or frequency is relevant, the execution time, count of instructions or number of Multiply or Accumulate operations will be taken into account. This is a more accurate approach than counting Floating Point Operations (FLOPS) on embedded systems. It may also be helpful to evaluate metrics like training time, search cost or even the resulting energy consumption as done in paper [12].

ProxylessNAS is proposed in [15] and mainly focuses on mobile target devices. It is not included in the previous section but shortly explained here as three of the MCU-class NAS approaches mention it as a reference design. In [16] a survey on deep learning techniques, which are critical for edge intelligence implementation, statements on the performance of MCUNet, Sparse and OFA can be found. Unfortunately the table is missing important information such as the search cost.

By using the MCUNet framework instead of Tensorflow Lite Micro and CMSIS-NN the required amount of MCU-memory drops by the large factor of 3.4 while the inference time is 1.7- to 3.3-times shorter [8]. This achievement is likely due to the interpreter based inference engine used in the TFLite Micro Library which introduces a large memory overhead compared to TinyEngine's code-generation approach. In addition the layer-level optimization strategies which are used in the MicroTVM framework as well are sub-optimal, which lead to further advantages for MCUNet. [8].

Model accuracy was first evaluated by comparing top1 ImageNet results which achieved >70% on common microcontrollers. Meanwhile the required SRAM memory and flash compared to quantized MobileNetV2 and ResNet-18 is 3.5 or 5.7 times smaller and especially for visual and audio wake words, which are popular reference implementations, the MCUNet framework is faster and also smaller than MobileNetV2 and ProxylessNAS-based solutions [8].

Experiments with μNAS have been conducted on image classification data sets. Either the top-1 classification accuracy can be increased by up to 4.8%, the memory footprint can be lowered by 4–13 times or alternatively a reduction of the number of multiply-accumulate operations by $\approx 900\times$

is possible according to [9]. Further interesting numbers can be found in a table of pareto-optimal architectures as it can be seen that μNAS outperforms SpArSe in multiple character classification

SpArSe achieves to provide a more accurate and up to $4.35\times$ smaller models compared to previous approaches on IoT data sets. Unfortunately no comparisons with either of the other papers are available, hence why performance on standard datasets and the Bonsai-Net paper [17] are the only references in this case. Character classification with SpArSe turned out to to have similar higher accuracy or drastically lower memory requirements and network sizes.

MicroNet models have been deployed on MCUs to compete with Tensorflow Lite Micro in experiments. Visual wake words, audio keyword spotting and anomaly detection have been compared with MobileNetV2, ProxylessNAS, Tensorflow Lite Micro and MSNET [11].
The paper also states the pareto optimality of MicroNet models against MCUNet for KWS tasks.

Using the Once-for-All tools over 1000 sub-networks suitable to run on several constrained target hardware platforms could be obtained. Regardless of the actual latency constrains it was possible to maintain the level of accuracy after the initial training. Actual numbers are provided for ImageNet top-1 accuracy under the mobile setting with sub-600 million Multiply-Accumulate operations for running the network while achieving an accuracy of 80% [12].

Even on edge devices the results are promising as either an increase of ImageNet top1 accuracy by 4.0% or a $1.5 - 2.6$ times lower latency compared with MobileNetV3 and EfficientNet at constant accuracy level is possible. It is emphasized that the amount of GPU hours for training also drops by many orders of magnitude resulting in less CO2 emission [12].

IV. Conclusion

To summarize the results of the recent work on Neural Architecture Search for edge devices, it is easy to see that it is an highly relevant research topic which offers a lot of innovation for the future AI industry.

The biggest improvements compared to reference models can been acquired if the NAS algorithm is co-designed with the corresponding inference engine. The reason for this is that a larger search space provides more opportunities to find optimal designs, due to faster model invocation.

All approaches are published under an open-source license which should allow other researchers to built up on these work to obtain even better results. For example Tensorflow Lite Micro, which is the most relevant microcontrollers inference framework, could profit a lot if NAS would be integrated in the design workflow. Beside the reusability improvement this could also make it easier to gather data for comparisons with other approaches.

There are a lot of other relevant research results on lightweight machine learning unrelated to NAS which use optimization techniques such as mixed low bitwidth quantization [18]. Implementing CNNs on MCUs with standard convolutions instead of depth-wise convolutions by trading accuracy with computational cost also offers appealing results as proposed in [19]. Another model compression method is proposed in [20] and uses a reordering of the model moderation to reduce its memory footprint.

References

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv*, vol. 20, pp. 1–21, 2018.

[2] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, Z. Chen, L. Wang, A. Xiao, J. Chang, X. Zhang, and Q. Tian, "Weight-sharing neural architecture search: A battle to shrink the optimization gap," *arXiv*, pp. 1–24, 2020.

[3] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-Art," *arXiv*, no. Dl, 2019.

[4] X. Chu, B. Zhang, R. Xu, and J. Li, "FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search," 2019.

[5] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019.

[6] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," 2020. [Online]. Available: http://arxiv.org/abs/2010.08678

[7] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm cortex-M CPUs," *arXiv*, pp. 1–10, 2018.

[8] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny Deep Learning on IoT Devices," no. NeurIPS, pp. 1–12, 2020. [Online]. Available: http://arxiv.org/abs/2007.10319

[9] E. Liberis, Ł. Dudziak, and N. D. Lane, "μNAS: Constrained Neural Architecture Search for Microcontrollers," 2020. [Online]. Available: http://arxiv.org/abs/2010.14246

[10] I. Fedorov, R. P. Adams, M. Mattina, and P. N. Whatmough, "SpArSe: Sparse architecture search for CNNs on resource-constrained microcontrollers," *arXiv*, pp. 1–26, 2019.

[11] C. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, "MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers," 2020. [Online]. Available: http://arxiv.org/abs/2010.11267

[12] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," *arXiv*, pp. 1–15, 2019.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[14] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html

[15] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv*, pp. 1–13, 2018.

[16] D. Liu, H. Kong, X. Luo, W. Liu, and R. Subramaniam, "Bringing AI To Edge: From Deep Learning's Perspective," pp. 1–23, 2020. [Online]. Available: http://arxiv.org/abs/2011.14808

[17] R. Geada, D. Prangle, and A. S. McGough, "Bonsai-Net: One-shot Neural Architecture Search via differentiable pruners," 2020.

[18] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," *arXiv*, 2019.

[19] M. Rusci, M. Fariselli, A. Capotondi, and L. Benini, "Leveraging automated mixed-low-precision quantization for tiny edge microcontrollers," *arXiv*, pp. 1–12, 2020.

[20] E. Liberis and N. D. Lane, "Neural networks on microcontrollers: saving memory at inference via operator reordering," *arXiv*, pp. 1–8, 2019.