DOI 10.5162/SMSI2023/A4.1

Boosting Energy Efficient Machine Learning in Smart Sensor Systems

Julio Wissing, Stephan Scheele
Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits IIS,
Division Smart Sensing and Electronics, Erlangen, Germany
julio.wissing@iis.fraunhofer.de, stephan.scheele@iis.fraunhofer.de

Summary:

Smart intelligent sensor systems are key for many application areas in the domain of ubiquitous computing like embedded autonomous systems, wireless sensor networks, internet of things and wearables. In this article, we provide an overview of recent approaches to improve the design, structure and deployment of machine learning methods for smart sensor systems in order to make them as lightweight and energy efficient as possible. The techniques considered cover both hardware and software perspectives.

Keywords: Smart Sensor Systems, Machine-Learning, Energy Efficiency, Hierarchical Machine-Learning, Industry 4.0

Introduction

Smart Sensor Systems combine advanced signal processing and data-fusion techniques with machine learning (ML) algorithms to ensure low-latency processing of sensor and measurement data directly at the edge device. They found numerous applications in Industry 4.0, IoT, medical diagnostics, smart cities, autonomous embedded systems etc.

In recent years, a growing interest in optimizing machine learning methods with respect to their energy efficiency can be observed as challenging and attractive research topics in both academia and industry. In particular**,** facing a recent global energy crisis, it seems more appropriate than ever to improve the energy efficiency of AI methods and their sparing use of computational resources, as used in countless smart sensing devices worldwide.

Optimization Methods

As discussed, optimizing the resource efficiency for smart sensor systems is a crucial point to ensure the viability of a TinyML approach. Key requirements of intelligent sensor systems are low-latency processing at a high data rate, high reliability, data security and a long battery lifetime[3]. During the development life cycle of such systems, there are multiple optimization points that a developer can utilize for efficient processing at the edge. In the following, we will go through the life cycle step by step to highlight possible optimization mechanisms and pitfalls to ensure the most efficient execution. In this short overview we will cover the hardware side only briefly and focus on methodical optimizations on the software side.

Pre-processing and Feature Extraction

Before any data is fed into the ML-pipeline, the sensor data needs to be prepared to be more efficiently processed. At this point, denoising and data filtering methods can be applied, so that only needed data is passed to the model. After the data preprocessing step, the remaining sensor data is fed into the feature extraction stage. Here, the dimensionality is reduced to contain only relevant information. This has the benefit of quicker training and the possibility to use more efficient models in the next stage. However, creating a viable set of features can be hard to achieve. Usually, some sort of expert knowledge is needed to identify patterns, possible transformations or meaningful statistical moments. Additionally, some features might not be as influential as others and may therefore be left out. To overcome this issue, feature pruning or search can be included in the process. This is achieved by either tracing back through the machine learning model to find the most relevant inputs [4] or by applying search algorithms like random search, genetic algorithms or reinforcement learning. Additionally, a standard feature extraction might be unnecessary in the case of NNs, which inherently identifies meaningful input data in the

training process but come with the downside of increased computational complexity.

Machine Learning Model Optimization

After feature extraction, a machine learning algorithm is applied to transform the given inputs into the desired output format. We divide possible optimizations into two groups: Ante and post-hoc optimization. The first one can be applied before the actual training and comes down to a meta-approach of choosing the most efficient classifier. In a greedy fashion, the most efficient model wrt. energy usage and precision can be selected. It must be noted that for suitable models not only different kinds of neural networks (NNs) should be considered. Even though a neural architecture search is an applicable approach to find efficient models, classical methods like decision trees might be a suitable option with a good working feature extraction.

Post-hoc optimization can be applied after training. This step is very dependent on the chosen ML-Method but can broadly be described in either reducing the bit width of calculations or leaving out unnecessary parts of the model. The first one is referred to as quantization [2], which brings down a model from floating point to integer 8-bit precision or lower. The second approach falls into the category of pruning, which might be pruning branches in a decision tree, omitting classifiers in an ensemble, or leaving out neurons in a NN [2].

Hierarchical Machine Learning

In some applications it might be possible to divide a classification into smaller sub-tasks. In a divide-and-conquer fashion, instead of using one ML-model to solve the task, smaller and more efficient models are used. Consider an industry 4.0 scenario with a smart accelerometer to find bearing damages in a machine. A standard approach would be to classify every time frame of sensor data with a big NN or a support-vector-machine. In the hierarchical case, we can implement an event-wake-up-trigger and split the task into three stages: Anomaly detection, fault localization and severity classification. Because of the smaller sub-problems, the first stage can be solved with a linear classifier, the second one with a random forest and the third one with a selection of tree-based classifiers [1].

Hardware selection

From the hardware perspective, the platform can highly impact the resulting efficiency of a system and must be selected alongside with the software. The algorithms should be tested for the lowest precision possible and for the need of floating point calculations. Most optimally, all calculations can be done in 16 or 8 bit and a floating point unit can be omitted. Additionally, further hardware modules are helpful to accelerate the execution of certain algorithms. Usually, the more specialized an acceleration unit is, the more efficient the execution becomes if the algorithm can utilize it. A DSP-unit is one of the broader kind of accelerators, which can optimize various calculations like vector/matrix operations, Fourier-transformations, or statistics. Furthermore, typical SIMD-Accelerators like vector or neural processing units can accelerate both NNs and matrix-operations. NNs can also utilize extremely specialized hardware like analog Neuromorphic circuits, which can cut down latency and energy usage drastically [5]. A final consideration when picking hardware platform(s) is software support. A lot of silicon manufacturers and IP-providers are delivering software-librarys alongside their chips that help to utilize their hardware more efficiently. Typical examples for that are ARM-CMSIS or STM32-Cube AI.

Conclusion and Outlook

We presented an overview of methods to improve the efficiency of machine learning algorithms for smart sensor systems. Depending on the application or ML-model, these can be applied before, during, or after training an ML model. Existing results show that a good trade-off between model efficiency and accuracy can be achieved, significantly increasing efficiency (e.g. reduction of 47% energy consumption [1]) without sacrificing the ML model accuracy and fidelity.

References

[1] J. Wissing, S. Scheele, A. Mohammed, D. Kolossa, und U. Schmid, HiMLEdge – Energy-Aware Optimization for Hierarchical Machine Learning, *Advanced Research in Technologies, Information, Innovation and Sustainability*, 15-29 (2022); doi: 10.1007/978-3-031-20316-9_2

[2] T. Liang, J. Glossner, L. Wang, S. Shi, X. Zhang, Pruning and quantization for deep neural network acceleration: A survey, *Neurocomputing* 461, 270-403 (2021) ; doi:10.1016/j.neucom.2021.07.045.

[3] S. Li, L. Xu, S. Zhao, 5G Internet of Things: A survey, *Journal of Industrial Information Integration 10*, 1-9 (2018); doi:10.1016/j.jii.2018.01.005.

[4] S. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K. Müller, W. Samek, Pruning by explaining: A novel criterion for deep neural network pruning, Pattern Recognition 115 (2021), doi:10.1016/j.patcog.2021.107899.

[5] R. Müller et al., Hardware/Software Co-Design of an Automatically Generated Analog NN, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, 385–400 (2022); doi: 10.1007/978-3-031-04580-6_26