EFFICIENT DEPLOYMENT OF MACHINE LEARNING MODELS ON MICROCONTROLLERS: A COMPARATIVE STUDY OF QUANTIZATION AND PRUNING STRATEGIES.

Rafael Bessa Loureiro[1], Paulo Henrique Miranda Sá[1], Fernanda Vitória Nascimento Lisboa[1], Rodrigo Matos Peixoto[1], Lian Filipe Santana Nascimento[1], Yasmin da Silva Bonfim[1], Gustavo Oliveira Ramos Cruz[1], Thauan de Oliveira Ramos[1], Carlos Henrique Racobaldo Luz Montes[1], Tiago Palma Pagano[1], Rafael Borges[2], Oberdan Rocha Pinheiro[1].
1, Departamento de Software, Centro Universitário SENAI CIMATEC, Salvador 41650010, Brasil;
2, HP Inc., Brasil RD;

Abstract: With the advancement and growth of Internet of Things tools, the necessity for more complex and intelligent systems increases, presenting many challenges due to device limitations in memory, computation, and energy consumption. The objective of this study is to do a literature review of optimization techniques with quantization and pruning in machine learning models for deployment on microcontrollers. The methodology consists of searching the literature, the tools, the models, and the techniques. Results show that pruning has better accuracy, while quantization has better inference time and size reduction, with the best results reducing 90% with 3.5x faster inference. Careful consideration of trade-offs between inference time, model size, and accuracy is crucial for deployment on edge devices.

Keywords: edge computing; microcontroller; optimization; quantization; pruning.

PROCESSO DE IMPLANTAÇÃO DE MODELOS DE APRENDIZADO DE MÁQUINA EM MICROCONTROLADOR: UM CASO DE USO PARA IDENTIFICAÇÃO DE MEDIDORES INDUSTRIAIS

Resumo: Com o crescimento da Internet das Coisas, a demanda por sistemas complexos e eficientes aumenta, embora desafios surjam devido a limitações de dispositivos em termos de memória, poder de processamento e energia. O Objetivo desse estudo é realizar uma revisão da literatura das técnicas de otimização, como quantização e pruning, para modelos de aprendizado de máquina implantados em microcontroladores. A metodologia inclui pesquisa de literatura, ferramentas, modelos e técnicas. Os resultados revelam que a poda melhora a precisão, enquanto a quantização otimiza o tempo de inferência e a redução de tamanho, com destaque para reduções de até 90% no tamanho e 3,5 vezes mais rapidez na inferência. Considerar as compensações entre tempo, tamanho e precisão é vital para a implantação em dispositivos de borda.

Palavras-chave: computação de ponta; microcontrolador; otimização; quantização; pruning.

1. INTRODUCTION

With the advancement of Internet of Things (IoT) tools, its applications have been growing and can be seen in areas such as agriculture, healthcare, vehicle traffic, and industry (4.0). With its significant adoption, surges the necessity for more complex and intelligent systems, like Deep Learning (DL) algorithms, which have been successfully adopted in a variety of applications, such as speech recognition, recommendation systems, and video classification. Nonetheless, DL is still heavily dependent on high-performance computing platforms with dedicated Graphic Processing Units (GPU) and is very expensive in terms of computation, memory, and power consumption [1].

Edge computing emerges to cover those issues, providing computing service with low communication latency by deploying the computing and storing resources as close to data sources as possible. Low energy cost, since 70% of the device power consumption is spent on communication wireless communication. Higher security by reducing data-sensitive transference. Reduced costs with local processing with edge servers and devices [1]. Some examples of DL being applied in edge devices are energy efficiency, fault detection of a system, healthcare, smart cities, anomaly detection, and Human Activity Recognition in the industry [2].

Edge computing presents many challenges due to device limits such as memory, computation, and energy consumption. Since the majority of DL algorithms are created for high-accuracy targets by utilizing high-performance Central Processing Units, GPU, or High-Performance Computing, one of the main challenges of deploying DL models on edge devices, is to reduce the model without losing its accuracy. There are methods for improving DL models, to enable them to be applied in edge devices, such as quantization [2], pruning [3], and knowledge distillation [4].

To address the issue of reducing the model’s size and improving its performance on Microcontroller Units (MCU), The objective of this study is to do a literature review of optimization techniques with quantization and pruning in machine learning (ML) models for deployment in an MCU.

To validate the pipeline, it was applied to a real industrial problem, of industrial gauges inspection, the pipeline was applied to the task of capturing images that need to be processed and locating the gauges in those images. The processing can be done by an MCU embedded in the robot. However, most classical computer vision models require memory and processing that are incompatible with the MCU. Therefore, the model needs to be reduced without significantly compromising its performance.

This paper is organized into five sections. The section 2, describes the search for the literature**,** tools, methods, and techniques used in the experiment, while 3 details the results, and finally, 4 presents the final considerations and recommendations for future research.

2. METHODOLOGY

The methodology of this study consists of four stages: searching the literature, the tools in ML in Edge Computing, the Neural Networks (NN) models, and the techniques.

2.1 Searching the Literature

The searches were conducted on IEEEXplorer, Scopus, Web of Science, and Google Scholar, limited for the period 2020 to 2023. There were made search string:

• TITLE-ABS-KEY ((("deep learning" OR "machine learning" OR "edge AI" OR "edge inference") AND ("Microcontroller" OR "Arduino") AND ("pruning" OR "quantization" OR "tuning" OR "knowledge distillation")))

The search string was limited to repositories and journals, early-access articles, and magazines published. This research returned a total of 101 results, which, after reading its abstracts and excluding duplicates, have been reduced to only 10. To select these works, the following exclusion criteria were adopted:

1. Papers more related to cloud computing or without using edge computing;

2. Papers that do not apply one or more of the ML model optimization techniques;

3. Documents that use ML models to improve only device performance;

4. Articles that do not make the model inference on the edge device.

2.2 Tools

Among the tools are libraries that provide tools for the compression and reduction of AI models listed in Table 1. Some allow the creation of models with lower computational cost, such as TensorFlow Lite (TFLite) [5] which is an adaptation of TensorFlow [6] for mobile devices allowing the creation and conversion of models to lighter versions, or Artificial Intelligence for Embedded Systems (AIfES) [7] which is a library focused on Arduino and brings the possibility of creating more compact models in MCUs, bringing adaptations of layers, activation functions, and optimizers allowing the adaptation of models for systems from 8 to 64 bits, however, there are limitations, primarily for Computer Vision (CV) applications. Furthermore, some focus on the reduction of models already made, some of them are the TensorFlow Lite itself which also incorporates compression techniques, and PyTorch [8] which is widely used for CV and DL being a library for creating and training NN also includes the tools for model reduction, these libraries allow the use of techniques such as Pruning, Quantization, and Knowledge Distillation focused on reducing the computational cost or increase the efficiency of NN.

Table 1: Table with different tools for model optimization.

Toolkit	Version	Pruning	Quantization	Knowledge Distillation	Arduino support
AIfES	2.1.1	no	yes	no	yes
TFLite	2.12.0	yes	yes	yes	yes
Pytorch	2.0.0	yes	yes	yes	yes
TinyNeural Network	0.1.0	yes	yes	yes	yes

AIfES, short for Artificial Intelligence for Embedded Systems, is a C platform framework for the creation of machine learning models on MCUs. It permits model conversions from other libraries, like Keras and TensorFlow, and supports platforms ranging from 8 to 64 bits. AIfES only supports one type of neural network, the Feed

Forward Neural Network, this framework's key feature allows models to be trained directly within the embedded system, which is made possible by prior numerical optimizations made to the available layers.

TFLite is made to run machine learning models on hardware with constrained computational capabilities. It supports techniques like quantization and pruning to decrease the size and speed up the model, and it provides a highly optimized runtime for executing models effectively on different embedded devices.

PyTorch is an open-source machine learning framework developed by Facebook. It is based on the Torch library, which is a scientific computing framework that provides a wide range of functionalities for mathematical operations, signal processing, and machine learning. This framework also has techniques that allow optimization of large ML models by means of reducing them. Those techniques include pruning and quantization, with the possibility of porting such models to embedded devices, such as Arduino.

For DNNs, TinyNeuralNetwork's [9] model compression framework is designed with the goal of porting large models to IOT and mobile devices. For simpler IOT deployment, the framework supports direct conversion between TensorFlow Lite and PyTorch, along with classic compression techniques such as pruning, and quantization.

2.3 Models

To use the toolkits described above, would need to create or use a neural network to solve our problem. In this way, the MobileNetV2 networks were used, in addition to the development of other neural networks, such as a CNN neural network and a Multilayer Perceptron (MLP) neural network. All three were used for image classification tasks.

The MobileNetV2 architecture is comparable to the original MobileNet architecture, except that it uses inverted residual blocks with bottleneck resources. Furthermore, it allows images larger than 32x32 and has a much lower parameter count than the original architecture.

For the Pytorch compatible toolkits, a CNN neural network was designed to work with grayscale image classification. It was designed to contain 2 layers of Conv2d and MaxPooling2d, with a dense output layer.

The pytorch-playground example was used for the MLP model. It is composed of three linear layers, two layers of 20% Dropout, and ReLU activation function.

2.4 Techniques

Low Magnitude Pruning in TensorFlow Lite works by zero-masking specific regions of the model. In practice, it allows the user to select the ultimate level of sparsity, which is attained during training, while the advantages of a quicker model can be utilized during inference.

TinyNeuralNetwork's One Shot Channel Pruner aims to prune CNNs in a single passing, allowing for reduced computational costs in the forms of retraining and fine-tuning.

XNNPACK aims to improve NNs performance on low-memory devices. For that matter, the library provides highly optimized neural network operators that allow for accelerating models developed in other frameworks, such as TensorFlow and

PyTorch. In the past few years, the library has received many improvements from external sources.

Random Unstructured Pruning works by selecting modules or layers from a model to apply pruning parameters. It works by removing randomly selected unpruned value units. Thus, the modified module will be returned with the binary mask applied to the parameter previously selected by the pruning method. Consequently, after pruning, the selected parameter is replaced by its pruned version, while the original (unpruned) one is stored.

Post-training Quantization aims to reduce the precision of a given model's weights and activations by truncating them from 32-bit float to the smallest possible size, such as a 4-bit integer, after the model has already learned the prediction task.

Dynamic quantization is a PyTorch post-training quantization technique that performs this task by determining the scaling factor for triggers based on the range of data observed during runtime. In addition, model arithmetic is performed using INT8-type vectorization, which allows model optimization, reducing memory usage and computational requirements. This technique is considered good to apply in production pipelines because it is relatively free of tuning parameters. Thus, it was applied in the image classification task, after training the CNN network. Subsequently, the network was evaluated again in order to compare the performance between the original and quantized networks.

As part of the linear quantization approach, each layer of the neural network will undergo conversion to 8 bits in parameters, inputs, and outputs. Attributes will be randomly initialized at module creation time, then original layers will be added with quantized layers, making inference time longer.

3. RESULTS AND DISCUSSION

3.1 Correlated Works

There are Significant efforts focused on developing and executing state-of-the-art models for specific tasks using low-power MCUs with extreme resource constraints [10-12]. For Vision-based classification and detection tasks, like face detection, and handwritten digits' classification to be able to run in MCUs, pre-processing techniques are used in the data, like quantization of the input image to 8-bit values, and training specific miniaturized networks based on state-of-the-art ones, like a Mini VGG which reduced the original architecture through a smaller input size, and the use of less: Channel dimension, multiple convolution layers, and multiple fully connected layers. It's also noted the specific reduction can vary depending on the problem's complexity, resulting in a reduction of size at the cost of robustness to more complex settings, like poor lighting [10].

For audio and image classification assignments, a methodology is proposed focused only on pruning aware training, based on MCU specifications. There is presented that compressed models are produced faster and more accurately due to better use of the MCU resource budget. This approach shows that MobileNetV2 training is faster than without the optimization technique, and compares resource usage between GPU and mobile-levels [11].

A method called adaptive scale quantization is used for quantizing tiny Convolutional Neural Network (CNN)s models in low-bit. ImageNet was used to classify objects, and an approach for balancing quantization error and information entropy to retrain the model and preserve its accuracy was used. The deployment was made in inexpensive devices, with the potential for designing small footprint CNNs accelerators with higher efficiency, enabling a wide range of new applications using Tiny Machine Learning (TinyML) [12].

Model specific pipelines are also present in the literature, for a CNN specific one, five stages are defined for training and deploying the model, focused on model size, workload, operations, and quantization awareness, mapping the desired model to the specification of a resource-constrained MCU specification. As a result, when users realize all five pipeline stages, they can fit, deploy, and execute multiple CNNs across multiple open-source MCU boards, with a reduction to a 1/10th of their original size while preserving accuracy around the first decimal point.

3.2 Pruning and Quantization

The results of pruning and quantization techniques are presented in this section, as can be seen in Tables 2 and 3.

The pruning technique reduced most of the libraries' inference times, although techniques based on wrapping layers with masks increased model size, as shown in Table 2. The calculation of model size, inference time, precision, and sparsity are comparisons between the original model and the model after the corresponding technique was applied.

The results varied greatly between models, TF Lite used the Low Magnitude pruning approach, wrapping a mask around weights and activations to interpret them as zeros and reduce their impact on calculations, however, since information is lost in the process, it caused a reduction in accuracy. Another noteworthy point is that most weights will have masks of either 0 or 1, which caused the model to almost double in size. As for XNNPACK, it was possible to greatly increase sparsity and decrease inference time at the cost of about 60% increase in model size.

Pytorch's Random Unstructured Pruning yielded a 20% increase in inference time while improving model accuracy by 0.1%, which is rather small when compared with other techniques, and a small increase of 20% in model size. It should be noted that the pruning technique performed with PyTorch results in an unstructured model, making it harder to train or optimize.

Lastly, the One Shot Channel Pruner was used with TinyNeuraNetwork to prune the model, obtaining the largest increase in inference time, 70%, with a great amount of sparsity at 55% and an increase of 90% in model size, being the second-largest increase.

Table 2: Table of Results for Pruning Approaches

Pruning Library	Model	Technique	Inference Time	Accuracy	Sparsity	Model Size
TF Lite	Mobile NetV2	Low Magnitude	8% ↓	0.3% ↓	60%	1.94x ↑

XNNPACK	Mobile NetV2	XNNPack	8% ↓	0.1% ↑	75%	1.6x ↑
PyTorch	Custom CNN	Random Unstructured	20% ↑	0.1% ↑	32%	1.1x ↑
TinyNeural Network	Mobile NetV2	OneShot Channel Pruner	70% ↑	0.1% ↓	55%	1.9x ↑

With the use of the quantization techniques, the inference variation of most models improved, except for Openvinotoolkit, which stayed the same, as shown in Table 3. Aimet’s post-training quantization reached a similar margin at 90% speed-up, a very small drop in accuracy, and a great model reduction, making it 3.5 times smaller. However, the Pytorch-playground technique had a result increase of 32% in inference time, with a small increase in accuracy, maintaining the same size.

As for TensorFlow Lite, the inference time was reduced by about 3 quarters of the original, with a huge accuracy drop of 10%. One possible cause of the large accuracy drop is that the representative dataset was unable to activate the most important activation ranges, which requires more investigation. Another important result of this framework is that it achieved the largest reduction in model size, making the quantized model 4 times smaller. Lastly, Pytorchs’s Dynamic Quantization offered inferior speed-ups with greater overall stability, making the model 65% smaller without harming accuracy.

Table 3: Table of Results for Quantization Approaches

Quantization Library	Model	Technique	Inference Time	Accuracy	Model Size
TF Lite	MobileNetV2	Post-training INT8	75% ↓	10% ↓	4x ↓
PyTorch	Custom CNN	Dynamic Quantization	14% ↓	0%	1.65x ↓
PyTorch-playground	Custom MLP	Linear Quantization	32% ↑	0.07% ↑	0x
Aimet	MobileNetV2	Post-training INT8	90% ↓	1% ↓	3.5x ↓
Openvino toolkit	MobileNetV2	Post-training INT8	0%	0%	0.53x ↓

4. CONCLUSION

This study focused on addressing on the challenges of deploying DL models on MCUs in edge computing scenarios. The proposed multi-stage pipeline, along with optimization techniques like quantization and pruning, shows potential for reduce model size and improve performance on resource-constrained devices. However, careful consideration of trade-offs between inference time, model size, and accuracy is crucial for successful deployment on edge devices. Further research and experimentation are recommended to explore more optimization possibilities and applications for the proposed approach.

Acknowledgments

This publication is the result of a project regulated by the Brazilian Information Technology Law (Law No. 8,248 of 1991 and its subsequent updates) and was developed under Agreement HP 077-2022 - HP/Cimatec - ViaLab 2023 between SENAI CIMATEC and HP Brasil Indústria e Comércio de Equipamentos Eletrônicos Ltda.

6. REFERENCES

[1] VÉSTIAS, M; et al. **Moving deep learning to the edge**. Algorithms, 13(5), 2020.

[2] GHIBELLINI, A; et al. **Intelligence at the iot edge: Activity recognition with low-power microcontrollers and convolutional neural networks**. In Proceedings - IEEE Consumer Communications and Networking Conference, 2022.

[3] MOLCHANOV, P; et.al. **Pruning convolutional neural networks for resource efficient transfer learning**. CoRR, abs/1611.06440, 2016.

[4] HINTON, G; DEAN, J. **Distilling the knowledge in a neural network**, 2015.

[5] DAVID, R; et al. **Tensorflow lite micro: Embedded machine learning for tinyml systems**. In A. Smola, A. Dimakis, and I. Stoica, editors, Proceedings of Machine Learning and Systems, volume 3, pages 800–811, 2021.

[6] ABADI, M; et al. **TensorFlow: Large-scale machine learning on heterogeneous systems**, 2015. Software available from tensorflow.org.

[7] GEMBACZKA, P; et al. **Combination of sensor-embedded and secure server-distributed artificial intelligence for healthcare applications**. Current Directions in Biomedical Engineering, 5(1):29–32, 2019.

[8] PASZKE, A; et al. **Pytorch: An imperative style, high-performance deep learning library**. In Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

[9] DING, Huanghao; PU, Jiachen; HU, Conggang. **Tinyneuralnetwork: An efficient deep learning model compression framework.** https://github.com/alibaba/TinyNeuralNetwork, 2021.

[10] COOPER, G; MANJUNATH, B.S.; ISUKAPALLI, Y. **Edge machine learning for face detection**, 2021.

[11] E Liberis and N D Lane. **Differentiable neural network pruning to enable smart applications on microcontrollers**. Proceedings of the ACM on Interactive Mobile Wearable And Ubiquitous Technologies, 2022.

[12] XU, K; et al. **Etinynet: Extremely tiny network for tinyml**, 2022.