Optimized TinyML Implementation for Resource-Constrained Microcontrollers

Om lbaneen Audi[1*], Sara Awada[1*], Mohamad Yaacoub[2] and Ali Ibrahim[1]

[1] Computer and Communication Engineering Department, Lebanese International University (LIU), Beirut 1105, Lebanon
[2] Department of Naval, Electrical, Electronics, and Telecommunications Engineering, University of Genoa, Genoa, Italy
***These authors have equally contributed**
ali.ibrahim@liu.edu.lb

Abstract. TinyML has emerged as a transformative technology enabling the development of tiny yet powerful models for on edge real-time sensory processing. This study investigates the optimization of TinyML models for the deployment on resource-constrained devices like the Arduino Nano 33 BLE Sense. Optimization techniques such as pruning followed by a sparsity removal approach and quantization were applied using the TensorFlow framework. The results obtained demonstrate a considerable reduction in processing time of around 98%, RAM use of 65% on the Arduino Nano 33 BLE Sense with a slight loss in accuracy by 4 %.

Keywords: Tiny machine learning, Resource-constrained devices, Pruning, Quantization, Sparsity removal.

1 Introduction

The growth of the Internet of Things (IoT) has resulted in an explosion of data-generating devices, ranging from wearables to smart household appliances, creating a huge opportunity for machine learning (ML) to extract insights and enable intelligent decision-making [1] [2]. However, conventional ML models are often too large and computationally intensive for resource-constrained edge devices [3] [4]. These devices often have limited memory and processing capacity, but TinyML enables them to analyze data and make intelligent decisions directly on the device [5] [6]. This technology leverages the potential to replace cloud computation serving systems, enabling the design of sophisticated, always-on, battery-powered devices avoiding the need to transfer data to the cloud for processing continually.
In this paper, the aim is to design a robust TinyML model for classification by utilizing optimization techniques such as low-magnitude pruning followed by removing sparsity and full-integer post-quantization. The model is optimized to reduce latency, memory use, and power consumption, allowing it to be deployed in resource-constrained contexts while maintaining high accuracy.

The rest of the paper is organized as follows: Section 2 provides an overview of optimization methods used in the literature. Section 3 describes the experimental setup and methodology adopted in this work. Section 4 presents the results of the methodology compared with a reference model. Conclusions are drawn in Section 5.

2 Related Works

Several state-of-the-art solutions have proposed optimization techniques to reduce the complexity of large models to meet the limitations of tiny devices. Edge platforms require compact deep neural network (DNN) models due to limited memory and power constraints. Effective model compression strategies, such as structured and unstructured pruning, as well as quantization, have significantly reduced the memory footprint and computational demands.

To give an example, structured pruning achieves around four times memory compression [7], whereas class-blind unstructured pruning, as demonstrated in PruNet [8], achieves an impressive 190 times memory compression. In [9], authors present a strategy for pruning filters with relatively low weight magnitudes to generate CNNs with lower computing costs while avoiding irregular sparsity. This approach reduces FLOP by around 30% for VGGNet (on CIFAR-10) and deep ResNets while maintaining the same accuracy.

The Q-Caps Nets framework [10] demonstrates quantization's efficiency in decreasing the memory requirements of complicated deep neural networks (DNNs) such as Caps Nets [11] by 6.2 times, with just a 0.15% accuracy drop. The authors in [12] propose a quantization approach that uses only integer arithmetic to approximate floating-point operations in neural networks. ARM NEON-based implementations boost inference efficiency while also reducing model size by four times.

The combination of quantization and pruning, as demonstrated by Deep Compression approach [13], shown significant benefits, achieving a tripled compression rate for models such as Alex Net and VGG-16. However, a method for compressing neural networks for TinyML applications is proposed in [14]. This method involves removing the final layers of a pre-trained model and replacing them with a smaller classifier head trained on an auxiliary network. This strategy requires no additional hardware, and results in a significant size reduction with minimal accuracy loss for TinyML applications.

On the other hand, authors in [15] evaluate the trade-off between accuracy and efficiency for several structured model pruning approaches and datasets (CIFAR-10 and ImageNet) using Tensor Processing Units (TPUs). Moreover, they developed a structured model pruning framework for TensorFlow2 that modifies models in place (rather than adding mask layers) by reloading weights from the original model, improving pruning accuracy.

3 Experimental Setup and Methodology

3.1 System Design

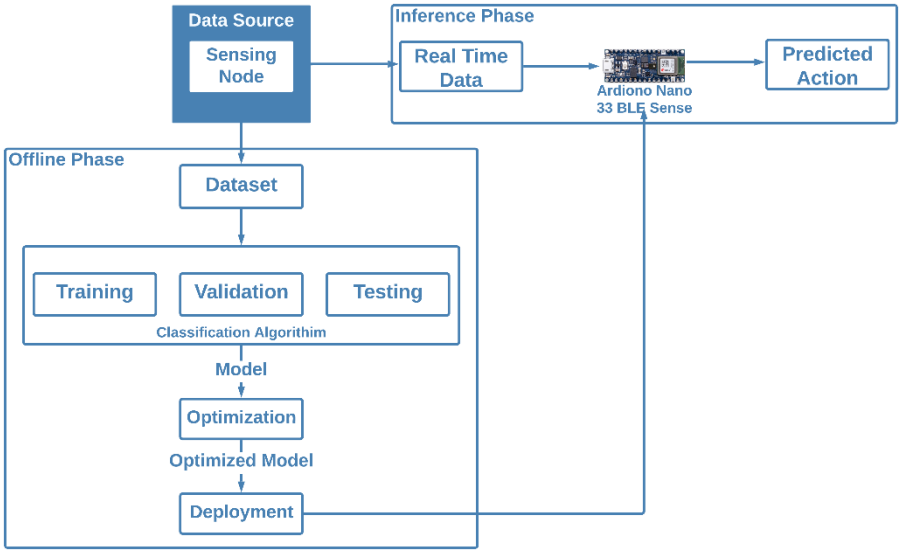

Fig.1. System Block Diagram

Fig. 1 illustrates a general flow of the methodology. Initially, the data is collected from a sensing node and saved on a PC. In the offline phase, a neural network like CNN or MLP receives this data as input and trains to identify patterns and correlations within the data. The training involves feeding data into the network to adjust its internal weights and biases to minimize the difference between expected and actual outputs. Subsequently, validation is performed, in which a distinct set of data is utilized to evaluate the network performance during training and prevent overfitting. Following training, the model performance is evaluated using unseen testing dataset, to assess its generalization performance. The resulting model is optimized aiming to reduce its complexity, thereby reducing memory occupation, and latency using multiple techniques and methods.

Finally, the optimized model will be deployed on the Arduino Nano 33 BLE sense, featuring nRF52840 microcontroller and equipped with embedded sensors. In the inference phase, the deployed model on the Arduino receives real-time sensor data as input for prediction.

3.2 Model Architecture and Training

Dataset. To evaluate the proposed methodology a dataset was collected from a multisensory glove equipped by five IMU MPU6050 6-axis sensor modules (3-axis gyroscope and 3-axis acceleration) sensors and five Force Sensitive Resistor SEN-09375 sensors with a diameter of around 12.5mm.

Eight different objects have been 3D printed for the data collection varying in shape and size: big cuboid, large cube, large sphere, large pyramid, tiny cuboid, small cube, small sphere, and small pyramid.
For each object the subject, where asked to grasp hold and then release, repeating this sequence 150 trials per object, resulting in a dataset compromising 1200 trials.

Classifiers. After performing a grid search, two machine-learning algorithms are evaluated on the collected data set: one dimensional convolutional neural network (1DCNN) and multilayer perceptron (MLP). The results indicates that the simple 1DCNN model is the best fit, having the smallest size and highest accuracy. This model consists of three layers, where the first layer is Conv1D, with a number of filters 16, and a kernel size of 3, and a Rectified Linear Unit (ReLU) activation function followed by GlobalAveragePooling1D. Finally, there is a dense layer with 8 neurons corresponding to the number of classes with the softmax activation function.

The data is split into 80% training and 20% testing. The model is compiled using the Adam optimizer with a learning rate of 0.001 and a sparse categorical cross-entropy loss function, with a batch size of 32. After stratified k-fold cross-validation with 10 folds, the model achieves an average training accuracy of 98.78% and an average testing accuracy of 94.92%.

3.3 Optimization Methods

Magnitude-based weight pruning was applied, a widely recognized model optimization technique promoted by TensorFlow that zeros out less significant model weights during the training process to achieve 50% model sparsity. Following pruning, the model is compiled using the Adam optimizer and sparse categorical cross-entropy loss function, then trained for 20 epochs with a batch size of 32. Then the model is stripped to remove the pruning wrappers, resulting in a streamlined model that retains the pruned structure but without the overhead of the pruning logic.

After that, the stripped model weights are retrieved and handed to eliminate the sparse weights and return the non-zero weights in flattened arrays. The first convolutional layer has two weight matrices: the first is reshaped by the same kernel size, input channels, and half the number of filters, while the second is reshaped by half the number of filters. The second layer, a global average-pooling layer, contains no weights to reshape. For the last dense layer, the first weight matrix is reconfigured using half the number of filters and units in the layer, while the second matrix stays unchanged. A new model is then built using the same architecture as the base model, but with the number of filters in the first layer lowered by half. The new reshaped weights are assigned to the new model. This model is then compiled using the same configuration as the base model but with learning rate 0.01. Finally, it is trained across

50 epochs with a batch size of 32. Then full integer post-quantization was done using TensorFlow lite API, which converted all weights and activation outputs into 8-bit integer data and sets inference input and output types to int8.

4 Results and Discussion

The optimized model is converted to TensorFlow Lite (. tflite file) which is an MCU-compliant library. The file is then uploaded to Edge Impulse [16], which is used to evaluate the models performance in the Arduino Nano 33 BLE sense.

Table 1 below shows the performance and characteristics of four models: Simple-1DCNN, Pruned-1DCNN (resulted from TensorFlow pruning), Reshaped-1DCNN, and post-quantized (PQT) Reshaped-1DCNN. The Simple-1DCNN model achieves testing accuracy of 97.08% for a model size of 10,160 bytes. Pruned-1DCNN show a slight reduction in the accuracy approximately 2.08% while reducing the zipped size significantly, from 8,445 bytes to 5,710 bytes, highlighting its effectiveness in enhancing model compressibility by eliminating sparsity.

The Reshaped-1DCNN model further reduces the model size to 6,596 bytes with an accuracy drop of approximately 3.34%. It also requires fewer floating-point operations (Flops) at 96,848, nearly half of what Simple-1DCNN and Pruned-1DCNN require. This reduction in computational complexity shows that the Reshaped-1DCNN is more efficient. Furthermore, Reshaped-1DCNN has the least zipped size among all the models, at 5,020 bytes.

The PQT-Reshaped 1DCNN model, optimized for potential efficiency, has a testing accuracy of 93.33%. This model has the smallest uncompressed size of 4,448 bytes.

Table 1. Measured Metrics for the Implemented Models.

Model	TF-Lite Testing Accuracy (%)	Size (bytes)	Zipped Size (bytes)	Number of Flops
Simple-1DCNN	97.08	10160	8445	193648
Pruned-1DCNN	95	10160	5710	193648
Reshaped-1DCNN	93.75	6596	5020	96848
PQT-Reshaped 1DCNN	93.33	4448	5810	-

Table 2. Performance of the Models on Arduino Nano 33 Ble sense.

Model	Processing Time (ms)	RAM Usage (KB)	Flash Usage (KB)
Simple-1DCNN	410	17.6	32.2
Pruned-1DCNN	410	17.6	32.2
Reshaped-1DCNN	328	17.6	28.6
PQT-Reshaped Model	9	6.2	33.8

On the Arduino board as shown in Table 2, the Simple-1DCNN and Pruned-1DCNN models exhibit the same processing time of 410 milliseconds and RAM usage of 17.6 KB, and flash use is 32.2 KB. This indicates that pruning alone does not immediately reduce resource requirements, except for compressed storage. In contrast, the Reshaped-1DCNN model achieves considerable advancements, minimizing processing time to 328 milliseconds while keeping RAM use at 17.6 KB. The flash use decreases to 28.6 KB, demonstrating the efficiency benefit realized with the sparsity removal approach.

The PQT-Reshaped model stands out for its very fast processing time of only 9 milliseconds, demonstrating a significant performance improvement. This model also exhibited the lowest RAM use, at 6.2 KB, significantly lower than the other models. However, its flash use has increased slightly to 33.8 KB due to the quantization overhead.

Giving a small neural network architecture, quantization involves including information such as scaling factors and quantization parameters to handle the conversion from full precision (float32) to lower-precision data types (int8). This new information increases the total model size. Despite this, the trade-off is justified by a significant reduction in processing time and RAM utilization, making the PQT-Reshaped model ideal for deployment on resource-constrained devices.

5 Conclusion

This paper focused on the development of optimized TinyML models for classification to be deployed on resource-constrained devices. Our primary objective was to optimize the model for efficient operation on the Arduino Nano 33 BLE Sense.

Several techniques were employed including full-integer post-quantization and low-magnitude weight pruning. Subsequently, we applied a method to eliminate the sparsity and reshapes the weights, resulting in an optimized model. This study introduced a solution that significantly improves the model performance. The optimized model process information approximately 98% faster and use 65% less RAM, although it takes up slightly more flash space (5%) while retaining high accuracy.
Future work involves integrating this method with techniques such as neural architecture search (NAS) to automatic the discover of optimal model architecture, thereby further improving efficiency and performance.

Acknowledgments

This project has received funding from the European Union's Horizon Europe research and innovation program under the Marie Sklodowska-Curie grant agreement No 101086359.

References

1. Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, Mohsen Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials ,* vol. 20, no. 4, pp. 2923-2960, 2018.
2. Sakr, F.; Berta, R.; Doyle, J.; Capello, A.; Dabbous, A.; Lazzaroni, L.; Bellotti, F. CBin-NN: An Inference Engine for Binarized Neural Networks. Electronics 2024, 13, 1624.
3. Lina Zhou, Shimei Pan, Jianwu Wang, Athanasios V. Vasilakos, "Machine learning on big data: Opportunities and challenges.," *Neurocomputing,* vol. 237, pp. 350-361, 2017.
4. Khalife, R., Mrad, R., Dabbous, A., Ibrahim, A. (2024). Real-Time Implementation of Tiny Machine Learning Models for Hand Motion Classification. In: Bellotti, F., et al. Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2023. Lecture Notes in Electrical Engineering, vol 1110. Springer, Cham.
5. Fresta, M.; Bellotti, F.; Capello, A.; Dabbous, A.; Lazzaroni, L.; Ansovini, F.; Berta, R. End-to-End Dataset Collection System for Sport Activities. Electronics 2024, 13, 1286.
6. Dabbous, A., Fresta, M., Bellotti, F., Berta, R. (2024). Neural Architecture for Tennis Shot Classification on Embedded System. In: Bellotti, F., et al. Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2023. Lecture Notes in Electrical Engineering, vol 1110. Springer, Cham.
7. Sajid Anwar, Kyuyeon Hwang, Wonyong Sung, "Structured Pruning of Deep Convolutional Neural Networks," *ACEM,JETC,* vol. 13, 2017.

8. Marchisio, Alberto; Hanif, Muhammad Abdullah; Martina, Maurizio; Shafique, Muhammad, "PruNet: Class-Blind Pruning Method For Deep Neural Networks," *International Joint Conference on Neural Networks (IJCNN),IEEE,* pp. 1-8 , 2018.
9. Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, Hans Peter Graf , "Pruning Filters for Efficient ConvNets," *arXiv,* 2016.
10. Alberto Marchisio, Beatrice Bussolino, Alessio Colucci, Maurizio Martina, Guido Masera, Muhammad Shafique, "Q-CapsNets: A Specialized Framework for Quantizing Capsule Networks," in *IEEE Design Automation Conference (DAC)*, 2020.
11. Sara Sabour, Nicholas Frosst, Geoffrey E Hinton, "Dynamic Routing Between Capsules," in *Advances in neural information processing systems*, 2017.
12. Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A.G., Adam, H., & Kalenichenko, D. (2017). Quantization and 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition,* pp. 2704-2713, 2017.
13. Song Han, Huizi Mao, William J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding," *arXiv preprint arXiv:1510.00149,* 2015.
14. Josen Daniel De Leon, Rowel Atienza , "Depth Pruning with Auxiliary Networks for Tinyml," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 3963-3967.
15. Kongtao Chen, Ken Franko, Ruoxin Sang , "Structured Model Pruning of Convolutional Networks on Tensor Processing," *arXiv,* 2021.
16. "Edge Impulse," [Online]. Available: https://edgeimpulse.com/. Accessed 16 May 2024.