Inter-Operability of Compression Techniques for Efficient Deployment of CNNs on Microcontrollers

Hamoud Younes, Hugo Le Blevec, Mathieu Léonardon, Vincent Gripon

Inter-Operability of Compression Techniques for Efficient Deployment of CNNs on Microcontrollers

Hamoud Younes[1,2], Hugo Le Blevec[1], Mathieu Léonardon[1], and Vincent Gripon[1]

[1] IMT Atlantique, Lab-STICC, UMR CNRS 6285, F-29238, France
firstname.lastname@imt-atlantique.fr
[2] Department of Computer and Communication Engineering, Lebanese International University, Bekaa, Lebanon
hamoud.younes@liu.edu.lb

Abstract. Machine Learning (ML) has become state of the art for various tasks, including classification of accelerometer data. In the world of Internet of Things (IoT), the available hardware with low-power consumption is often microcontrollers. However, one of the challenges for embedding machine learning on microcontrollers is that the available memory space is very limited, and this memory is also occupied by the rest of the software elements needed in the IoT device. The problem is then to design ML architectures that have a very low memory footprint, while maintaining a low error rate. In this paper, a methodology is proposed towards the deployment of efficient machine learning on microcontrollers. Then, such methodology is used to investigate the effect of using compression techniques mainly pruning, quantization, and coding on the memory budget. Indeed, we know that these techniques reduce the model size, but not how these techniques interoperate to reach the best accuracy to memory trade-off. A Convolutional Neural Network (CNN) and a Human Activity Recognition (HAR) application has been adopted for the validation of the study.

Keywords: CNN, Quantization, Pruning, Coding, Microcontrollers

1 Introduction

Convolutional neural Networks (CNNs) are state-of-the-art solution for computer vision tasks including image classification, object detection, etc. In the last decade we have been witnessing an increased adoption of CNNs in a wider range of applications such as voice recognition, near-sensor intelligence, human activity recognition and more. These applications demand the availability of smart devices that can perform inference in a faster and more energy efficient fashion [1]. However, CNNs are both computationally intensive and memory intensive, which makes it challenging for their deployment on the edge. This is specifically the case for the deployment on memory-constrained devices such as

Microcontroller Units (MCUs). The main challenge in embedding CNNs on microcontrollers is that the available volatile and non-volatile memories are in the order of few hundred/thousand kilobytes [2]. For applications on the edge, the available memory is also occupied by the rest of the software stack, e.g. WiFi, Bluetooth, etc.

One of the widely used methods to reduce the computational and memory requirements of CNNs is compression. Compressing a CNN can be achieved through a set of techniques such as quantization, pruning, and coding (in our work, coding refers to using lossless compression techniques when storing the network parameters into memory). In principle, quantization and pruning are known to reduce the memory footprint at the expense of accuracy reduction. As for coding, it reduces the CNN memory requirement at the expense of adding a latency overhead during inference to reconstruct the initial model parameters. Applying the aforementioned techniques is not a straightforward process as it is not known which quantization level to use, the amount of parameters to prune, how to combine two or more techniques, and if coding is necessary or not. Also, the existing deep learning frameworks fall short in some scenarios, for example it is not possible to use any number of bits for quantization.

In this paper, such challenges and questions are tackled through the following main contributions:

- A methodology towards embedding efficient CNNs on microcontrollers using several compression techniques and open source frameworks for deep learning training is proposed.
- A detailed study on the interoperability of compression techniques to achieve a high accuracy to memory trade-off is presented.

The rest of the paper is organized as follows: Section 2 describes the compression techniques used in the study. Section 3 details the proposed methodology for efficient CNN deployment on microcontrollers. Section 4 presents the experiment performed during the study including, the CNN architecture, training procedure, and the case study adopted for verification. Section 5 highlights the findings and shows a thorough comparison between the different simulated scenarios. Section 6 concludes the paper and illustrates on future work.

2 Compression Techniques

Quantization has shown consistent reliability and success in addressing the issues of deploying neural networks on constrained hardware platforms by reducing the number of bits required for the representation of each weight. Several works have tackled quantizing the training phase [3, 4], while others tackled the quantization in the inference phase [5, 6]. However, it has been shown that going below half-precision is a challenging task and requires a lot of fine tuning to maintain an acceptable error rate [7]. This challenge has provoked the need for frameworks such as TensorFlow Lite Micro [8], Larq Compute Engine [9], CMSIS-NN [10], and CMIX-NN [11] for the use of 8-bit and below quantization for both training and deploying neural networks on microcontrollers.

Pruning is the process of systematically removing parameters/neurons from an existing network to reduce the overall size. Pruning methods usually differs in their choices regarding sparsity structure, scoring, scheduling, and fine tuning. Some methods adopt unstructured pruning i.e prune individual parameters, others prune a fraction of the parameters with the lowest score (absolute value) within a layer or prune all desired weights at once, etc [12]. During development, pruning can be applied using built-in functions in frameworks such as PyTorch and TensorFlow, or through a custom technique as the ones proposed in [13, 14].

Lossless Compression (Coding) as its name implies, permits the reconstruction of the original data from compressed data without any loss in information. Lossless compression is widely used in "ZIP" file format and "gzip" tool in Linux environments. One of the widely used lossless coding methods is the "Huffman Coding" [15]. It uses a variable-length code-words to encode symbols based on the probability of occurrence of each symbol. Huffman coding has been applied to several state-of-art neural networks such as: ResNet-20, VGG-16, LeNet, etc. and results showed that a compression ratio up to 31× can be achieved for such architectures [16, 17].

3 Methodology

Figure 1 presents the proposed methodology towards the efficient deployment of CNNs on microcontrollers with a set of compression techniques. This methodology will be followed in order to study the interoperability of different compression methods on the accuracy to memory trade-off. The process can be summarized in five phases:

- **P1**: Define a CNN model using PyTorch where the number of feature maps and kernel size can be altered at any time. This model will be the baseline used for float32 training. Then, a second model will be defined where each layer (convolution, linear, etc.) from the baseline model will be replaced with its quantized implementation. Existing frameworks (PyTorch, TensorFlow, etc.) offers quantization only for $nbits \in [8, 16]$. Thus, an ongoing open-source framework called *Brevitas* is adopted [18]. Brevitas allows the use of any number of quantization bits, offers the capability to quantize either weights or activations or both, and provides choosing any quantization method. However, using Brevitas for low bit quantization ($nbits = 1, 2$), the training process requires extensive fine-tuning to maintain an acceptable accuracy loss. Thus, a third model that extends Binary Connect [19] is defined. This model can quantize the weights to [-1,1] or [-1,0,1] for $1-bit$ and $2-bit$ quantization respectively.
- **P2**: Train all the models then fine tune them to obtain a comparable accuracy with respect to state-of-the-art solutions. As fine-tuning is not an easy task, techniques such as Neural Architecture Search (NAS) [20] can be used to define a set of hyperparameters among which some lead to the best possible accuracy.

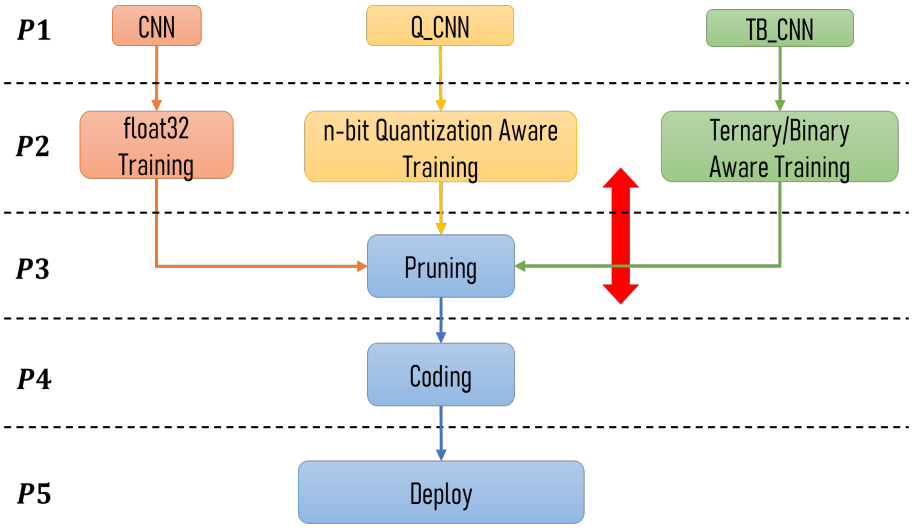

Fig. 1. Proposed Methodology Towards Efficient CNN Deployment on Microcontrollers

- **P3**: Apply a pruning method on the pre-trained models and fine tune to achieve a similar accuracy compared to the baseline model. It is worth mentioning that for low-bit quantization, pruning a CNN based on the weights values tends to be less effective as the set of possible weight values is very small (-1, 0, 1). A workaround could be to apply pruning before binarization.
- **P4**: For all the models, apply lossless compression using ZIP or gzip, which offers the same compression style as Huffman coding.
- **P5**: For deployment on microcontroller, a traditional deployment pipeline can be followed to obtain the flat buffer C array. Starting from a ".pth" file, Open Neural Network Exchange (ONNX) can be used to obtain a ".onnx" file, which is converted to a ".tflite" file. Finally, TFLite Micro library is used to export the C array. Note that for quantized models, Brevitas exports the weights in floating-point representation, thus it is necessary to apply post-training quantization to obtain the quantized weights.

Although frameworks such as PyTorch and Tensoflow offer out of the box pruning methods, they can only be combined with half-precision and 8-bit quantization. Moreover, CMIX-NN offers quantization with less than 8-bits, but the library is written in C and intended only for inference. Such problems are diminished with the use of the proposed methodology as it integrates several frameworks and provides insight on their capabilities. In addition, we applied this methodology to a case study and the results are thoroughly explained and discussed in Section 5.

4 Experimental Setup

This section describes the experimental setup performed to obtain an understanding on the interoperability of compression methods on the memory requirements of CNNs. In this paper, a Human Activity Recognition (HAR) problem [21] is adopted for derivation of the study. Such problem has been one of the targets for embedded machine learning on microcontrollers among the tiny Machine learning (tinyML) community as it helps understanding the human movement behaviour and how it could be extended to robotics and industrial applications [22].

4.1 Case study: Human Activity Recognition using CNNs

In this work, we used the dataset hosted by the university of California (UCI-HAR) [23]. The dataset describes 6 daily life activities: walking, walking upstairs, walking downstairs, sitting, standing, and lying. Each activity is a one-dimensional time series of 2.56 s corresponding to 128 samples obtained at 50 Hz sampling frequency. Each activity has 9 channels: 3 axes of total acceleration, 3 axes of angular velocity and 3 axes of body acceleration. The whole dataset contains a total of 10299 samples. For all training configurations, the UCI-HAR raw data has been normalized following the z-score of the training set.

As for the choice of CNN, a ResNet model with two blocks based on ResNet v1-6 presented in [24] is selected for the baseline model.

4.2 Training Procedure

The training process differs from one model to another due to the different representation of weights and hence different techniques could be required for fine tuning. However, there some notes to highlight:

- The kernel size has been fixed for all models to 3, while the number of feature maps (fmaps) is varied for 16, 32, 48, 64, and 96.
- L1-unstructured pruning from PyTorch has been applied to all models with a pruning rate $p \in [0.2, 0.4, 0.6, 0.8, 0.9, 0.95]$. This technique zeroes out the weights that have the lowest $l1$-norm.
- All models have been trained using 10-fold cross validation and the reported accuracy result is computed as the average among these runs.

Baseline Training The model is trained for 300 epochs using a batch size of 64. SGD has been chosen as an optimizer with a learning rate of 0.05 , momentum of 0.9, and a weight decay of 0.0005. A MultiStepLR scheduler is used to decrease the learning rate by a factor of 0.13 at epochs 100, 200, 250. Mixup [25] has been adopted for training with a factor of 0.2.

Quantized Training The training procedure is similar to the baseline model with few changes. Training is performed through Quantization Aware Training (QAT) with uniform scale quantizer [26], where gradients are computed using the quantized versions of the weights, while non-quantized parameters are updated using them. For a weight value w, the quantized value w_q is defined as:

$$w_q = quantize(w, b, s) = clip(round(w_q * s), -2^{b-1} + 1, 2^{b-1} - 1) \quad (1)$$

where $nbits$ is the quantization bits, s is the quantization scale set as a learnable parameter, and the $clip$ function is defined as:

$$clip(w, l, u) = \begin{cases} l & w < l \\ w & l \leq w \leq u \\ u & x > u \end{cases} \quad (2)$$

We trained three models where the quantization $nbits \in [4, 8, 16]$.

Ternary/Binary Training For the training of these two models, the weight decay is reduced to 0.004 while preserving other hyperparametrs from the baseline model. To compensate for the accuracy drop introduced from precision loss, batch normalization is applied after all convolution layers. Also, the model is trained for 120 epochs using float32 weights, then we clip the weights to [-1, 1]/[-1, 0, 1] for the remaining epochs. As for Mixup, it hasn't been used as the performance of low-bit quantized CNNs tends to be unstable.

5 Results and Discussion

Figure 2 presents the error rate (computed as $e = 1 - accuracy$, shown in log scale) versus the compressed model size for the models with different compression techniques. The compressed model size refers to the required non-volatile memory, which is to be uncompressed (decoded) during inference. It is worth mentioning that PyTorch framework keeps a value of zero in place of the pruned weights, thus the pruned model size is the same as the baseline. Hence, it is important to apply coding in order to get the real pruned model size. Moreover, models with a compressed model size greater than 400 KB (e.g. float32 and 16-bit models with 96 feature maps) have been left out as such size, when uncompressed, is already close to the maximum available volatile memory on mainstream microcontrollers. The plot shows the configurations with a compressed model size of 100 KB and less, such value is the target for most applications [1, 27]. The following observations can be highlighted by analyzing the plot:

- In most of the configurations, quantization offers a reduced error rate compared to pruning for the same model size. As the number of feature maps decreases, the error rate increases rapidly when pruning is applied regardless of the quantization level.

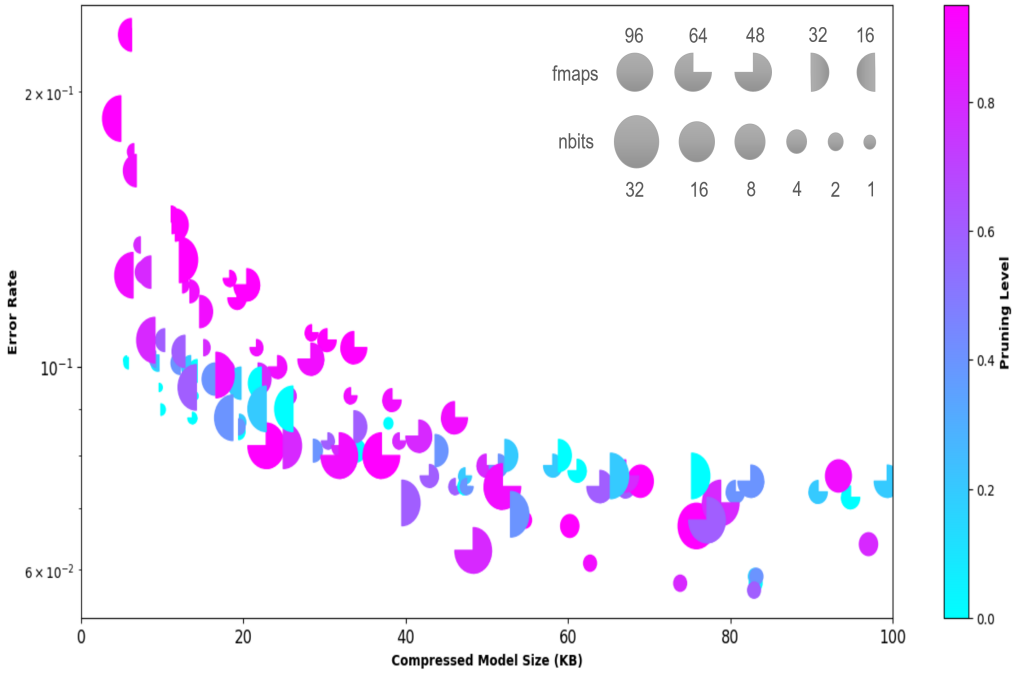

Fig. 2. Simulation Results for Models under Different Compression Techniques

- Low-bit quantization could achieve similar model size compared to high pruning levels but with a much reduced error rate. For example, a binarized model has the same size as a 95% pruned float32 model for about half the error rate.
- The model size decreases as the pruning level increases for models with float, 16-bit, and 8-bit quantization. However, for 4-bit quantization, a high pruning level ($> 60\%$) must be applied to obtain a noticeable decrease in the model size. This is due to the fact that as the number of bits decreases, the possible weight values decrease, hence weight-magnitude-pruning techniques effect tends to be negligible.
- Reducing the number of feature maps often offers a better error to model size trade-off compared to increasing the pruning level. For instance, a model with 16 feature maps has a lower model size compared to a 95% pruned model with 96 feature maps, with a slight difference in the error rate.
- Models with small number of feature maps offer a smaller size compared to quantized models with $nbits > 2$. However, the error rate is lower when quantization is adopted.
- Going for a region with a model size less than 20 KB is accompanied with an high error rate. Typically, the models in this region have small number of feature maps, a high pruning level, and 4-bit or less quantization.

- Combining a quantization level n with a pruning rate p offers a smaller model size compared to a model with $n/2$ quantization level at the expense of increased error rate. Thus, as a trade-off, it is better to use a lower quantization level instead of a higher one with pruning, but with an increased number of feature maps.
- Regardless of the number of feature maps and quantization level, applying more than 60% pruning results in an huge increase in the error rate. Thus, the best pruning level to achieve an acceptable error rate is around 40%. The model size of such configuration is then obtained based on the selected quantization level.
- 16-bit and 8-bit quantized models offer a much reduced model size compared to float32 with a similar error rate. The same could be said about lower quantization levels but with a noticeable increase in the error rate.
- Combining pruning with low-bit quantization (1 and 2-bits), leads to huge increase in error rate, hence such configurations are not present in the plot. So, it is advisable to apply low-bit quantization after pruning.

Based on the target application and the available memory budget, one could find several models that fits the need. For instance, an 8-bit quantized model with 32 feature maps has the same size as a 16-bit quantized model with 60% pruning. However, the error rate is much lower for the 8-bit model.

6 Conclusion

This paper presented a methodology for quantization aware training accompanied with pruning and coding to reach an efficient model design for the deployment on microcontrollers. The methodology has been applied on a human activity recognition task to study the interoperability between different compression techniques. The study is thoroughly discussed to highlight various design strategies for an error to model size trade-off. The findings paves the way towards more extensive experimental setup in the future, including different task, CNN architecture, and quantizer types. Moreover, for real-time applications, the error to latency trade-off is another aspect that could be investigated.

Acknowledgement

This work has been partly supported by the grant ADEME PERFECTO 2021. The authors would like to thank the GoodFloow company for the financial support and technical guidance to this project.

References

1. M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, "TinyML: Current Progress, Research Challenges, and Future Roadmap," *Proceedings - Design Automation Conference*, vol. 2021-December, pp. 1303–1306, Dec. 2021. ISBN: 9781665432740 Publisher: Institute of Electrical and Electronics Engineers Inc.

2. F. Sakr, F. Bellotti, R. Berta, A. D. Gloria, and J. Doyle, "Memory-Efficient CMSIS-NN with Replacement Strategy," *Proceedings - 2021 International Conference on Future Internet of Things and Cloud, FiCloud 2021*, pp. 299–303, Aug. 2021. ISBN: 9781665425742 Publisher: Institute of Electrical and Electronics Engineers Inc.
3. R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.
4. Y. Choi, J. Choi, M. El-Khamy, and J. Lee, "Data-Free Network Quantization With Adversarial Knowledge Distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.
5. R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.
6. Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "ZeroQ: A Novel Zero Shot Quantization Framework," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
7. A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," 2021. Publisher: arXiv Version Number: 3.
8. 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," Oct. 2020.
9. L. Geiger and P. Team, "Larq: An Open-Source Library for Training Binarized Neural Networks," *Journal of Open Source Software*, vol. 5, p. 1746, Jan. 2020. Publisher: The Open Journal.
10. L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," *ArXiv*, vol. abs/1801.06601, 2018.
11. A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, pp. 871–875, May 2020.
12. D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the State of Neural Network Pruning?," in *Proceedings of Machine Learning and Systems* (I. Dhillon, D. Papailiopoulos, and V. Sze, eds.), vol. 2, pp. 129–146, 2020.
13. D. Molchanov, A. Ashukha, and D. Vetrov, "Variational Dropout Sparsifies Deep Neural Networks," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 2498–2507, PMLR, Aug. 2017.
14. H. Tessier, V. Gripon, M. Léonardon, M. Arzel, T. Hannagan, and D. Bertrand, "Rethinking Weight Decay for Efficient Neural Network Pruning," *Journal of Imaging*, vol. 8, p. 64, Mar. 2022.
15. D. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, pp. 1098–1101, Sept. 1952.
16. S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

17. R. R. Gajjala, S. Banchhor, A. M. Abdelmoniem, A. Dutta, M. Canini, and P. Kalnis, "Huffman Coding Based Encoding Techniques for Fast Distributed Deep Learning," in *Proceedings of the 1st Workshop on Distributed Machine Learning*, (Barcelona Spain), pp. 21–27, ACM, Dec. 2020.
18. A. Pappalardo, "Xilinx/brevitas," 2021.
19. M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, (Cambridge, MA, USA), pp. 3123–3131, MIT Press, 2015. event-place: Montreal, Canada.
20. G. Kyriakides and K. Margaritis, "An introduction to neural architecture search for convolutional networks," *arXiv preprint arXiv:2005.11074*, 2020.
21. A. Anguita, Davide, A. Ghio, L. Oneto, X. Parra Perez, and J. L. Reyes Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proceedings of the 21th International European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, (Bruges), pp. 437–442, 2013.
22. Y. L. Coelho, F. A. S. Santos, A. Frizera-Neto, and T. F. Bastos-Filho, "A Lightweight Framework for Human Activity Recognition on Wearable Devices," *IEEE Sensors Journal*, 2021. Publisher: Institute of Electrical and Electronics Engineers Inc.
23. G. Chetty, M. White, and F. Akther, "Smart Phone Based Data Mining for Human Activity Recognition," *Procedia Computer Science*, vol. 46, pp. 1181–1187, 2015.
24. P. E. Novac, G. B. Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and Deployment of Deep Neural Networks on Microcontrollers," *Sensors 2021, Vol. 21, Page 2984*, vol. 21, p. 2984, Apr. 2021. Publisher: Multidisciplinary Digital Publishing Institute.
25. H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond Empirical Risk Minimization," 2017. Publisher: arXiv Version Number: 2.
26. H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation," 2020. Publisher: arXiv Version Number: 1.
27. J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-Efficient Patch-based Inference for Tiny Deep Learning," Oct. 2021.