Trade-offs in Neural Network Compression: Quantized and Binary Models for Keyword Spotting

Bruno Lovison-Franco, Jonathan Miquel, Aymen Romdhane, Guillaume Prenat, Lorena Anghel, David Novo, Pascal Benoit

Trade-offs in Neural Network Compression: Quantized and Binary Models for Keyword Spotting

Bruno Lovison Franco,[1] Jonathan Miquel,[1] Aymen Romdhane,[1] Guillaume Prenat,[2]
Lorena Anghel,[2] David Novo,[1] Pascal Benoit[1]
[1]LIRMM, University of Montpellier, CNRS - Montpellier, France - {firstname}.{lastname}@lirmm.fr
[2]UGA/CNRS/CEA - SPINTEC Grenoble, France - {firstname}.{lastname}@cea.fr

***Abstract*—Enabling smart and independent IoT devices often requires to run complex Machine Learning (ML) workloads at the edge. Such systems usually operate with memories in the order of tens of kilobytes and low processing power. To fit within these constraints, model designers typically rely on low-precision integer representation of operations down to 1-bit, i.e., Binary Neural Networks (BNN). In this paper, we investigate the tradeoffs available to model designers between memory footprint and accuracy and the challenges to overcome for effective use of BNN. We show that designing BNN architectures is not a straightforward process. To overcome this, we propose a methodology based on design guidelines and Neural Architecture Search (NAS) to adapt traditional model architectures into BNN variants. As a case study, we apply this methodology to a ResNet-based model for a keyword spotting (KWS) application. Our results demonstrate that, contrary to 8-bit quantization, direct binarization significantly impacts accuracy. However, careful architecture redesign and hyperparameter tuning helps bringing BNNs performances on par with their quantized counterparts.**

***Index Terms*—Edge Computing, Exploration, TinyML, Binary Neural Networks, Quantization**

I. Introduction

Recent years have been characterized by the explosion of Internet-of-Things (IoT) devices. Common IoT applications require gathering and processing sensor data to act on their environment accordingly. Traditionally, such systems transmit gathered data to a High-Performance centralized computing node to handle the bulk of the processing task through Machine Learning (ML) based algorithms. However, in many resource-constrained applications, the continuous transmission of high-frequency-sampling data (e.g., audio) over a low-power RF link (e.g., LoRa) is hardly achievable within the given constraints of available bandwidth and energy. It is possible to limit transmissions to a strict minimum by moving the processing to the edge device. This allows not only for saving energy and reducing the latency of the system but also comes with the added benefit of a reduced risk of data leaks.

However, executing ML workloads on Microcontroller Unit (MCU) class devices is not trivial due to the limited embedded resources. MCUs usually operate with memories in the order of tens of kilobytes and low processing power. In the TinyML field, ML models designers take into account such constraints at the cost of moderate loss of models accuracy. Quantization (i.e., reducing a model's operations precision) is the most popular optimization available to reduce the requirements of high-performance models. It allows for encoding full precision floating point parameters (32-bit) into an integer (typically 8-bit) representation, bringing three main benefits [1] : (i) MCUs do not require a Floating Point Unit (FPU) to execute the inference without relying on software emulation, (ii) The memory footprint of the model is reduced (by a factor of 4 for 8-bit quantization) and, (iii) larger bit-width MCU can speed-up execution by using single instruction multiple data (SIMD) operations. For all of these benefits, it is widely available in state-of-the-art portable frameworks such as Tensorflow Lite for Microcontrollers.

More aggressive quantization approaches, namely Binary Neural Networks (BNN), leverage the up-stated benefits to an extreme by reducing the encoding precision down to a single bit [2]. Due to their low memory footprint, it is possible to combine multiple BNN models predictions within a single system to improve the overall performances.

As such, TinyML models designers have a plethora of low memory-footprint options to bring existing models to the edge. In these conditions, comparing and selecting the best option is not obvious and exhaustive comparison is time-consuming.

In this work, we present this selection process by using a Keyword Spotting (KWS) application as a case study. We compare the effect on model size (i.e., model weights memory-footprint) of the above-stated options, namely 8-bit quantization, binarization, and ensembles. The target baseline model of this study is a ResNet-based model, namely *res8-narrow*, which was first introduced in [3]. This model was the result of a first accuracy vs model size exploration tailored for small footprint KWS applications. Our objective is to further reduce the size of this model at a minimal cost on accuracy to enable deployment on lower-end MCU-class devices. In summary, this work makes the following contributions:

- We evaluate the quantization of a ResNet-based model into both 8-bit and binarized weights and activations representation. We show that proper binarization of the baseline model require additional tweakings as opposed to 8-bit quantization.
- We adapt the design and training pipelines of a BNN model to improve its accuracy, our results show that such techniques allow BNNs to perform similarly with respect to the baseline model.
- We investigate binary ensembles to combine multiple model outputs in a voting system, we show that this straightforward implementation underperform architecture search approaches.
- We compare different low-bit representations of ML models to find Pareto-efficient *size vs accuracy* architectures, our results show that 8-bit quantification outperforms other options at extremely compact model sizes (less than 26KiB) and binary models becomes more memory-

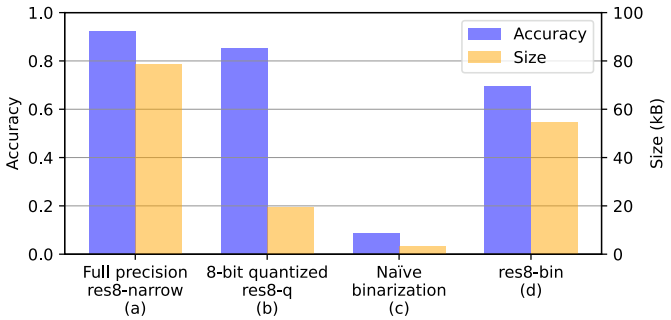

Fig. 1: Accuracy results obtained on test dataset vs model size for (a) the baseline model, (b) 8-bit quantization, (c) unchanged binarized architecture. The last plot (d) show the results of a model implementing basic BNN design guidelines.

efficient above this threshold.

II. Background

The following briefly summarizes the techniques we employed in the paper to balance accuracy and model size.

Quantization is a widely spread model compression technique that encodes full precision weights and intermediate outputs of a model to a lower precision (usually 8-bit integer).

Binarization is an extreme form of quantization in which parameters and intermediate outputs of a model are represented either as -1 or 1. The training of such models opens up new issues to tackle [2]. Nonetheless, the intrinsically low memory footprint of such a model makes it a good match for edge systems.

Binary Ensemble Neural Networks (BENN) leverage the low memory footprint of BNNs to combine in a single system multiple models with relatively low accuracy to improve the overall reliability. In this work, we investigate the tradeoff between memory and performance that such technique enables.

Neural Architecture Search (NAS) is an automated model architecture designing heuristic that operates within a specified search space [4]. We apply this design methodology in this work to efficiently explore tradeoffs between full precision, quantized, and binarized models.

III. Motivation

We base our exploration on a small memory-footprint model, namely *res8-narrow*, first introduced by Tang et al. [3] (See Figure 2). We first investigate the full-precision, 8-bit quantization-aware and binary training of this unchanged architecture on a KWS task. Figure 1 (a), (b), and (c) shows the accuracy results and model sizes obtained after the three different trainings.

The full-precision variant (a) reaches 92.14%, as it was tailored for mobile-class devices, this baseline already has a low memory-footprint of 80 KiB.

The 8-bit quantization variant (b), namely *res8-q*, achieves degraded but still comparable accuracy (85.43%) with the added benefit of a reduced model size of a factor $\approx 4\times$.

The binary variant (c) proves to be more challenging to train. This baseline binarized architecture does not meet fiability requirements reaching only 8.28% accuracy, which is on par with a random prediction.

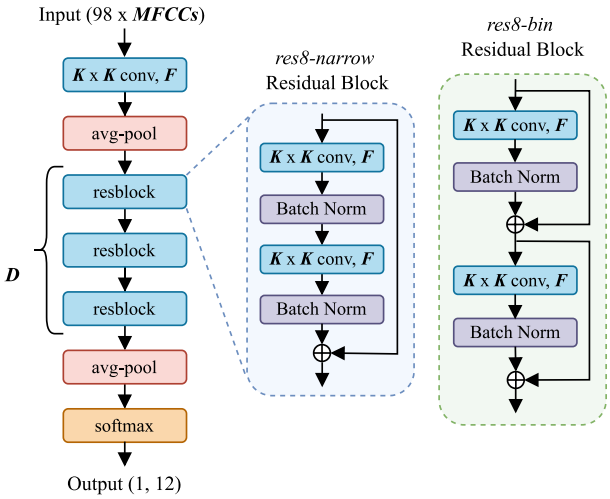

Fig. 2: On the left side we show the *res8-narrow* architecture, it is composed of an initial convolutional layer, three Residual Blocks (detailed in the center), and a global downsampling layer before the final softmax layer. On the right side, we show the updated Residual Block used for BNNs.

Following the BNN design principles presented by Bethge et al. [5], we adapt the *res8-narrow* architecture to perform better with binary parameters. Practically, this translates to two main changes : (i) we replace the downsampling (i.e., average pooling) layers with convolution layers, and (ii) we introduce additional residual connections in the residual blocks (See Figure 2). In the following, we refer to this updated architecture as *res8-bin*.

In Figure 1 (d), we show the architecture and training results of *res8-bin*. It achieves a much higher 69.50% accuracy compared to a direct binarization of *res8-narrow*. However, the removal of the downsampling layers leads to a model size of 56.06 KiB, higher than the 8-bit quantized variant *res8-q*. This highlights the need for further optimization to balance the trade-off between accuracy and model size for binarized architectures.

IV. Methodology

Bethge et al. [5] showed that BNNs require deep and complex architectures to counterweight their low weight-encoding precision. According to our previous experiment, the same applies to a lesser extent for 8-bit quantization. The data preparation step, computing Mel-Frequency Cepstral Coefficients (MFCC) features from audio clips, has also a significant impact on the models: For our architectures, the number of inputs and model size have a linear relationship. Finding size-efficient architectures with these newly added parameters is a three-step problem to solve through a trial-and-error process. First, we need to tweak models hyperparameters (from the model itself or data preparation) to identify architectures of interest (i.e., low-memory footprint). Second, we need to train selected architectures. Finally, we can merge low memory-footprint architectures displaying good accuracy into ensembles to form more complex and more accurate systems. The following details each steps of this process.

TABLE I: Hyperparameters explored during the iterative Neural Architecture Search. Bold numbers are the default value.

Iteration	Parameter	Values
1	MFCCs	10, 20, **40**
2	Residual Depth	2, **3**, 4, 6, 8, 10
3	Kernel Size	2, **3**, 5, 7, 9
4	Filters	4, 8, 16, **19**, 32, 64

A. Neural Architecture Search Parameters

In this work, we keep NAS space small for the sake of clarity. Table I summarize the hyperparameters tunned, with bold numbers representing values of the reference design *res-bin*. From early experimentations, these following four hyperparameters had the most significant impact on both model size and accuracy results. The four hyperparameters tunned are in order, (i) the number of MFFCs computed per 30ms window, (ii) the residual depth i.e. number of residual blocks (noted *D* in Figure 2), (iii) the convolutions' kernel size (noted *K*), and (iv) the number of filters for convolution layers (noted *F*).

Our approach employs an iterative NAS strategy to investigate, in order, the above-stated hyperparameters. Within this already contrained search space, we only consider model architectures whose size is inferior to the baseline *res8-narrow*, i.e., 80 KiB. For the retained architectures, we repeat model trainings three times and save the best-performing hyperparameter for subsequent explorations.

B. Model training

The following details the specificities for 8-bit quantized and binary models training.

Quantization Training. Two options are available to train quantized models. The first method is post-training quantization: We train a full-precision model and then use a few audio clips to evaluate quantization parameters (zero-point and scale). The second method is Quantization-Aware Training (QAT). For this approach, following TensorFlow recommendations [6], we also start by training the full-precision baseline model, then we run 15 additionnal fine-tunning epochs. During the fine-tuning, weights' 8-bit representation (and quantization parameters) are emulated and re-evaluated at each epochs.

Our early experimentations showed that QAT performs better than post-training on *res8-narrow*. For the sake of clarity, we only present the QAT results in this work.

Binary Training. Directly applying the gradient updates to binary weights during backpropagation is not feasible due to the sign function's zero derivative. To tackle this issue, we use full-precision latent weights during training. During the backward pass, these latent weights are updated using the Straight-Through Estimator (STE) [7]. During the forward pass, the latent weights are binarized as -1 or 1 using the sign function. Since full-precision latent weights are not needed for inference, we only save the binarized parameters in the final model so that latent weights do not impact memory-footprint.

C. Binary Ensembles Evaluation

The idea behind the binary ensembles is to combine multiple low-memory footprint models to counterweight their individually low accuracy. Each model within an ensemble has the same voting weight. Practically, this means that the prediction of the ensemble is the average of the predictions of each individual model. We select the three binary architectures with the lowest memory footprint found through our NAS exploration regardless of their accuracy. In our case study, these models' size are lower than 15 KiB. As such, we can combine multiples instances of these architectures in an ensemble (while still fitting memory constraints) to improve accuracy. For each selected architecture, we train two additional models and evaluate their combined accuracy and memory size in a 2-model ensemble. Note that no further selection is performed on trained models based on their individual performance. We repeat this process to build and evaluate 3-model ensembles.

V. Results and Discussion

In this section, we compare all above-stated model compression techniques as accuracy vs model size Pareto plot.

A. Experimental Setup

The models are trained and evaluated using Google's Speech Commands V2 dataset [8], composed of 105K speech samples of 35 words. We classify 10 keywords, namely: "yes", "no", "up", "down", "left", "right", "on", "off", "go", "stop" along with the classes "unknown" (sampled from the remaining 25 keywords) and "silence" (generated from non-vocal background noises). The audio clips are preprocessed in the form of MFCC features extracted from 30ms frames with a 10ms time shift (for a total of 98 sets of MFCCs). Following the recommendations of [8], we mix every keywords with background noise (at random amplitudes) and we add random timing jitter for input feature enhancement. An 80:10:10 split is used for training, validation, and testing subsets. For a fair evaluation of models, we use the hashed id associated with each audio clip to ensure that every audio excerpt from a common speaker remain in a single subset.

Model training and evaluation are made in Python using Google's Tensorflow framework [9]. To build and train the binary architectures, we use the Larq Library [10].

The models are trained using the Adam optimizer with an initial learning rate of $1e^{-3}$ (except binary whose learning rate starts at $1e^{-2}$) polynomially decaying down to $1e^{-5}$. We use a batch size of 64 clips, each model is trained for 30 epochs with 600 steps per epoch.

B. Pareto Front Evaluation

In Figure 3 we compare all trained models during the NAS based on their accuracy and model size. As a general rule of thumb for both 8-bit quantized and binarized models, the higher the number of parameters, the better subsequent classification performances. Both model size reduction approaches lead to a logarithm-like increase in accuracy results with diminishing benefits as the model size increases. Given the 80 KiB model size limitation, the number of quantized architectures available within our search space is more limited. As a direct consequence, more quantized architectures from the NAS are discarded, reducing the number of quantized models. Overall, small-sized models (i.e., models whose size is less than 26 KiB), tend to benefit more of a higher weight precision, and as such quantized models outperform binarized

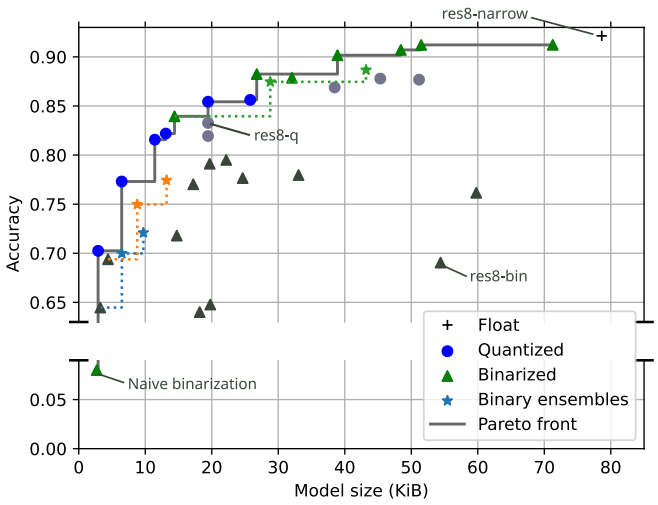

Fig. 3: Pareto plot obtained by comparing *accuracy vs size* for every model architectures explored through the NAS strategy.

approaches. Past this 26 KiB threshold, the tendency starts inverting, with binarized approaches leveraging their deeper and more complex architectures, consistently reaching higher accuracy thresholds. Explored binary ensembles consistently underperform NAS approach at comparable model sizes.

8-bit Quantized Models. The number of quantized architectures available within our limited search space is limited due to the empirical effect of hyperparameters on model size. Nonetheless, we observe through the applied NAS strategy that highly constrained size (less than 13KiB) quantized models leverage their relatively precise weight representation. This allows such models with low parameter counts (10 MFCCs, a kernel size of 3, and 4 filters per convolution) to reach 70.25% accuracy at just 2.91KiB. Doubling the filter count to 8 boosts accuracy to 81.57%. However, the effect on model size is not linear as it increases to 11.43KiB.

As model size grows (13 to 26KiB), the Pareto-optimal configurations shift towards shallower networks with more filters. At this range, models have a maximum of 4 residual blocks depth and keep the baseline number of filters (19). This suggests that for quantized models, a wider architecture becomes more effective at higher sizes potentially due to a better ability to capture complex features.

Past the 26 KiB range, accuracy increases become very incremental. As a matter of fact, roughly doubling the model size (from 25.78 to 51.17KiB) only yields an accuracy increase of 2% (from 85.65 to 87.68%).

Binary Models. For our KWS application, binarized models suffer from the low-precision weight representation at low model sizes. Practically, explored BNN architectures below the 26KiB threshold consistently achieve 5 to 10% less accuracy than similar-size quantized variants.

This trend inverts past 26KiB, at this point binarized models form the Pareto front of the 26 to 72KiB explored range. The very low-precision encoding of binary architectures allows us to explore more deep architectures, with higher filter counters and wider convolutional kernel size.

Increasing the size of binarized architectures also leads to diminishing returns on accuracy increase. This occurs at a higher model size (around 40 KiB) compared to quantized architectures. At this threshold roughly doubling the model size (from 38.90 to 71.27KiB) only yields an accuracy increase of 1% (from 90.17 to 91.23%).

Binary Ensembles. The memory cost of relying on ensemble approaches is a straightforward multiplication of the size of the model used as a baseline: a 2-model ensemble doubles the overall system memory footprint. A 2-instance ensemble increases on average 5.5% accuracy for the base binary models under 4.5KiB. However, the ensemble approach also suffers from diminishing returns on accuracy increase as we add additional models to the voting system. Diminishing returns are also noticeable as base model accuracy increases: For a 83.95% base performance, a 2-model ensemble only yields 3.5% more accuracy. For all of these reasons, in our experiments, accuracy from the ensemble approach proves to consistently underperform the NAS approach.

VI. Conclusion

In this work, we presented the trade-offs between accuracy and model size in the context of keyword spotting on resource-constrained devices. We used a ResNet-based model as a case study. We compared model size reduction techniques using 8-bit quantization, binarization, and binary ensembles. Our results showed that quantization yields better per-byte performance at tiny model sizes (less than 26KiB) while binarisation reaches higher accuracies past this threshold for our target baseline model.

Acknowledgment

This work was supported by the French National Research Agency (ANR) through the government grants EMCOM (ANR-22-PEEL-0009) and CHOOSE (ANR-22-PEEL-0013).

References

[1] L. Heim, A. Biri, Z. Qu, and L. Thiele, "Measuring what really matters: Optimizing neural networks for tinyml," 2021. [Online]. Available: https://arxiv.org/abs/2104.10645

[2] H. Qin, X. Ma, Y. Ding, X. Li, Y. Zhang, Z. Ma, J. Wang, J. Luo, and X. Liu, "Bifsmnv2: Pushing binary neural networks for keyword spotting to real-network performance," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 10 674–10 686, 2024.

[3] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5484–5488, 2018.

[4] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, pp. 1–21, 2019.

[5] J. Bethge, H. Yang, M. Bornstein, and C. Meinel, "Back to simplicity: How to train accurate bnns from scratch?" 2019. [Online]. Available: https://arxiv.org/abs/1906.08637

[6] TensorFlow Documentation, "Training quantized models," https://www.tensorflow.org/model_optimization/guide/quantization/training_comprehensive_guide, 2024, accessed: 2024-06-05.

[7] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013. [Online]. Available: https://arxiv.org/abs/1308.3432

[8] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016. [Online]. Available: https://arxiv.org/abs/1603.04467

[10] L. Geiger and P. Team, "Larq: An open-source library for training binarized neural networks," *Journal of Open Source Software*, p. 1746, 2020.