An Accuracy-Driven Compression Methodology to Derive Efficient Codebook-Based CNNs

Flavio Ponzina, Miguel Peón-Quirós, Giovanni Ansaloni, David Atienza
Embedded Systems Laboratory, Swiss Federal Institute of Technology, Route Cantonale, 1008 Lausanne, Switzerland
Emails: {flavio.ponzina, miguel.peon, giovanni.ansaloni, david.atienza}@epfl.ch,

***Abstract*—Codebook-based optimizations are a class of algorithmic-level transformations able to effectively reduce the computing and memory requirements of Convolutional Neural Networks (CNNs). This approach tightly limits the number of unique weights in each layer, allowing the storage of employed values in codebooks containing a small number of floating-point entries. Then, CNN models are represented as low-bitwidth indexes of such codebooks. This work introduces a novel iterative methodology to find highly beneficial schemes trading off accuracy and model compression in codebook-based CNNs. Our strategy can retrieve non-uniform solutions driven by an accuracy constraint embedded in the optimization loop. Our results indicate that, for a 1% accuracy degradation, our methodology can compress baseline floating-point CNN models up to 19x. Moreover, by reducing the number of memory accesses, our strategy increases energy efficiency and improves inference performance by up to 91%.**

***Keywords*—CNN compression, Clustering, Ensembling.**

I. Introduction

In the last years, Convolutional Neural Networks (CNNs) have been increasingly effective in addressing challenges related to Artificial Intelligence (AI). Nonetheless, highly accurate CNN models usually employ millions of parameters and require the execution of billions of multiply-and-accumulate (MAC) instructions per-inference [1]. These heavy workloads and memory requirements make state-of-the-art CNNs unfit to be executed on constrained edge devices, where their deployment could lead to a plethora of application scenarios where low latency, high efficiency, and privacy are key.

Recent research works have addressed this challenge from different perspectives, either providing edge devices with dedicated powerful and efficient hardware resources [2] or optimizing CNN models to reduce their complexity [3]. Among the latter strategies, most approaches are based on pruning [4], quantization [5], or a combination of these two methods. A further avenue towards the realization of efficiency CNN models is that of codebook-based methods. These are classes of CNN compression approaches that employ look-up tables to store a limited number of unique weights. Compression is achieved by exploiting the small size of codebooks, as low-bitwidth codebook indexes suffice to describe the actual CNN parameters. In contrast to quantization, where the CNN weights values are represented in low-bitwidth formats, codebook-based strategies preserve high-precision floating-point arithmetic. Therefore, the resulting efficiency gains are not linked to the availability of specialized, low-bitwidth hardware resources performing the MAC operations at the core of a CNN model. Instead, indexes of arbitrary bitwidth can be easily retrieved with standard mask and shift operations. Hence, several indexes can be stored in each memory word, extracted at run-time, and employed to fetch the corresponding floating-point codewords.

Previous works on codebook-based methods for model compression have showcased memory reductions of up to 8x, with limited impact on classification accuracy [6] [7]. Nevertheless, they present major shortcomings. They adopt uniform approaches, in which the same number of codewords is imposed in each layer, thus reducing a vast design space to a few (possibly sub-optimal) alternatives. Moreover, they do not include an accuracy constraint in their compression procedure. As the impact on accuracy of cannot be determined a priori, state-of-the-art strategies require a trial-and-error approach when having to comply with a user-defined accuracy threshold.

This paper addresses these shortcomings. First, our design flow employs non-uniform codebook-based representations to efficiently explore the complex trade-off between accuracy and compression. Second, our compression methodology includes an accuracy constraint in the optimization loop. Thus, we compress the baseline model while abiding by a target accuracy level. Third, our strategy improves run-time performance in addition to reducing the size of CNN models. Indeed, as codebooks have limited size, they fit in high-level (fast) processor caches, while multiple small-bitwidth indexes can be retrieved with a single (costly) access to main memory. Finally, in contrast to previous studies that either compress fully connected [8] or convolutional [9] layers, our approach targets both, thus enabling higher optimization possibilities in a more vast range of CNNs.

In summary, the contributions of this paper are:

- We introduce a novel CNN compression methodology that employs variable-size weights' codebooks to reduce memory requirements while meeting a user-defined accuracy level.
- We show that our optimized CNN design outperforms uniform codebook-based strategies, increasing compres-

This work has been supported by the EC H2020 WiPLASH (GA No. 863337), the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657) and the Swiss NSF ML-Edge (GA No. 200020_182009) projects.

978-1-6654-8356-8/22/$31.00 ©2022 IEEE

sion up to 56% for an accuracy threshold of 1%.
- We evaluate the inference performance of our optimized benchmarks, observing speedups up to 91% when compared to baseline implementations.

The rest of the paper is organized as follows: Section II introduces concepts and research efforts related to our work. Next, Section III details our accuracy-driven compression strategy. Section IV describes the adopted experimental setup, and Section V discusses our achieved results. Finally, Section VI summarizes our findings.

II. Background

A. CNN model compression

Due to their intrinsic redundancy [10], CNN models are amenable to optimizations. Such techniques reduce memory requirements and computing costs, with little impact on quality-of-service metrics such as accuracy.

To this end, CNN *pruning* approaches remove individual weights/connections [3] or, in some cases, entire filters [4] from a target architecture. Pruning can be applied on CNN structures before (and independently of) training, discarding a percentage of weights to meet the desired compression level. Alternatively, trained models can be pruned by removing specific weights/filters based on their magnitude [11] or entropy [12] to limit accuracy degradation. Pruning is adopted jointly with replication in [13]. In this paper, the authors first prune an initial CNN structure to obtain a compression factor of *I*. Then, the obtained pruned structure is replicated and independently trained *I* times to build an ensemble of CNNs (named E^{2}CNNs). At inference time, each ensemble's instance outputs are averaged, resulting in increased accuracy and robustness. Since these characteristics are highly desirable when dealing with model compression, we also employ ensembling as a pre-processing step in our framework.

Another way to reduce the memory requirements is *quantization* [14], where floating-point weights and activations are replaced by integer or fixed-point numbers. Usually low-bitwidth formats are adopted, with extreme examples represented by binary or ternary models [15]. Recent studies [16] [17] propose heterogeneous quantization strategies. In these works, bitwidths are tailored depending on the robustness of each CNN layer towards narrow data representations. Highly quantized CNNs require specialized hardware resources able to leverage low-bitwidth representations to reduce computing cost and memory transfers [17].

B. Codebook-based compression

Codebook-based compression strategies [8] [6] restrict the number of admissible weights values. These are stored (as floating-point numbers) inside small-sized look-up tables (codebooks). CNN models are then represented in terms of low-bitwidth indexes accessing the codebooks. Fig. 1 illustrates the different compression approach adopted in codebook-based methods with respect to quantization.

The use of codebooks results in large model compression ratios, as the bits required to index a codebook are only $log_2 k$, being k the number of codewords (i.e., admissible weight values). As an example, the use of k= 256 codewords only requires 8 bits to encode an index. A compression ratio of 4x is hence achieved by representing weights as codebook indexes instead of employing their value, when considering standard 32-bits floating-point representations. Note that the memory required for the codebook is negligible, since it has only a limited number of elements.

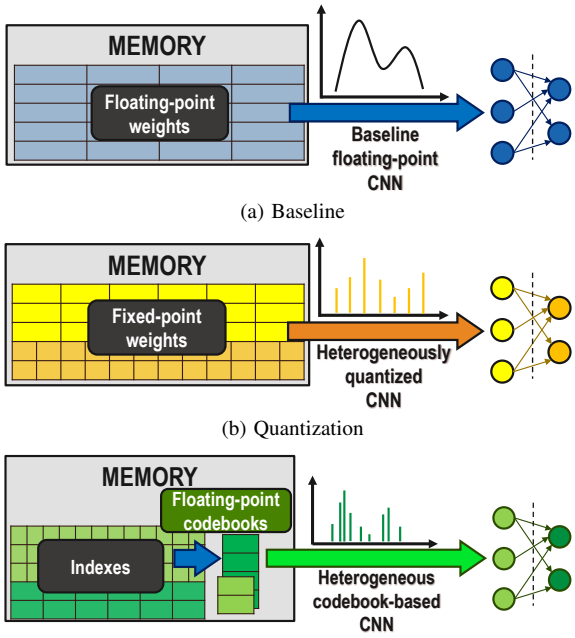

Fig. 1. CNN weights represented as (a) floating-point parameters, (b) quantized fixed-point values and (c) indexes in codebooks. As codewords are stored as floating-point values, accuracy is preserved to a large degree.

To preserve accuracy, the weights stored in codebooks must be judiciously chosen. We herein base our selection of codewords on clustering [18]. In more detail, we employ the k-means algorithm to group together similar weights in a cluster, and assign the cluster centroid to a codebook entry.

III. CNN compression methodology

An overview of the overall approach is summarized in Fig. 2. A detailed description of its different phases is presented in this section.

A. Optimization of ensembles of CNNs

Input of the framework is a single-instance CNN. As a first step of our methodology, an equivalent ensemble model is derived to increase robustness (Fig. 2.I), following the methodology described in [13]. The heuristic in [13] is also employed to determine the number of instances in the ensemble.

Each optimization step targets one instance at a time, processing them in a round-robin fashion. This approach enables a careful control of accuracy degradation and, by considering different instances in consecutive optimization steps, results in similar compression levels among the considered CNNs. Therefore, it prevents the final ensemble-based design

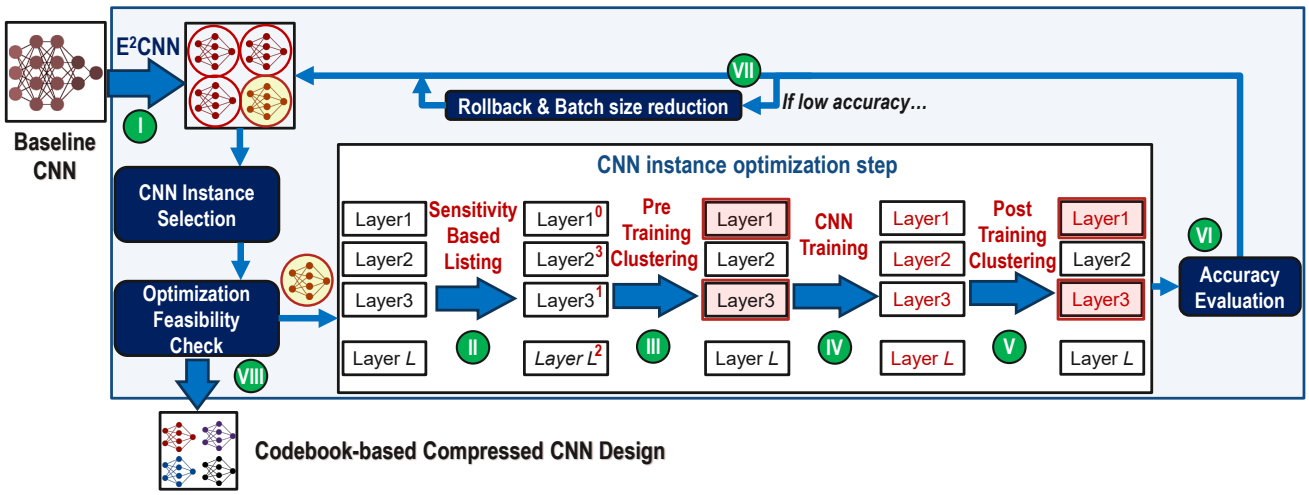

Fig. 2. Block scheme of the proposed methodology. The CNN model to be optimized is first converted into an ensemble of CNNs (I). Then, the ensemble instances are iteratively compressed via optimization steps (II, III, IV, V), which operate on a batch of layers in parallel, driven by an user-defined accuracy constraint (VI). The batch size is reduced (VII) when a rollback is performed due to the violation of the accuracy threshold (VIII).

from being composed of both highly-optimized and poorly-optimized CNNs, ultimately increasing the overall ensemble compression.

B. Sensitivity analysis for layer selection

To minimize the impact on accuracy of a compression step, each iteration of the methodology assesses the robustness of the convolutional and fully connected layers of the ensemble's instance being optimized. To this end, we consider as metric the *clustering sensitivity* S, which measures the range-normalized variance of weights (Fig. 2.II). S is defined as:

$$S(l) = \frac{\sigma^2(W_l)}{max(W_l) - min(W_l)} \quad (1)$$

where $S(l)$ indicates the clustering sensitivity of layer l and W_l represents the corresponding set of weights. Layers having a minor sensitivity are considered first as candidates for compression. Indeed, a low S value indicates that range-normalized weights are concentrated in a small region of the available data range, suggesting that reducing the number of codewords in the layer's codebook does not introduce large absolute-magnitude deviations.

C. Clustering-based codebook generation

Our methodology uses the k-means algorithm to group the parameters of each convolutional and fully-connected layer of an ensemble instance. Weights and biases are clustered independently, thus generating two codebooks per layer. If a layer, according to its computed sensitivity, is selected as a possible target for optimization, its number of codewords is halved. In that case, the bits required to index the corresponding codebook are reduced by one.

Note that only a subset of layers are targeted at each optimization iteration, and a rollback is performed if the accuracy degradation threshold is violated. Hence, our strategy results in non-uniform codebook sizes, both in different layers of the same CNN instance (internal heterogeneity) and in the same layers belonging to different instances (external heterogeneity).

We set $K = 32$ (i.e., 5 bits indexes) as the number of codewords in the first optimization step, as we empirically observed no accuracy degradation for this codebook size in all experiments described in Section V. Then, subsequent iterations limit the number of codebook entries to 16, 8, 4 or 2 values, corresponding to 4, 3, 2 or 1 bit indexes, respectively.

D. Logarithmic batching for layer compression

To reduce complexity and, therefore, compression/training time, our methodology applies the compression procedure to N layers of an ensemble instance in parallel (Fig. 2.III), selected as the ones having the lowest sensitivity ($S(l)$) values.

The value of N identifies the optimization batch size. Large choices of N produce a significant speedup of the optimization process by enabling its parallelization, but introduce coarser approximation steps that can extensively affect accuracy. On the contrary, small values of N better preserve accuracy at the cost of longer execution times for the optimization loop. Therefore, as detailed later in this section, batch sizes are progressively decreased during the optimization process, as less and less leeway for further compression is available.

The compression step is followed by a retraining phase (Fig. 2.IV). The retraining phase fine-tunes the weight values of the entire ensemble instance. This approach is key to preserving accuracy, as it largely compensates for the accuracy degradation due to compression. On the other hand, it has the adverse effect to cause weight values to deviate from those of the codewords, both in the layers being optimized in the current iteration and the ones in which the size of the codebook is unchanged. Hence, after re-training, the k-means algorithm is run a second time to fix weight values to those admitted by the instance codebooks (Fig. 2.V).

At this point, the accuracy of the entire ensemble of CNNs is evaluated (Fig. 2.VI). If its accuracy is acceptable according to the user-specified threshold, the optimization is retained and

the algorithm continues by starting a new iteration, selecting a new batch of layers to be compressed belonging to a selected CNN instance. On the contrary, if the accuracy degradation exceeds the threshold, the performed optimization is discarded, the previous instance implementation is restored and, for the targeted CNN instance only, the batch size N is halved. Hence, a more fine-grained approximation will be applied when the same instance is selected again (Fig. 2.VII). When $N = 1$ for a CNN instance (i.e., a batch size of 1), no further optimization is considered for the targeted layer if a compression iteration produces an unacceptable accuracy. The optimization loop terminates when no layer can be selected for further compression (Fig. 2.VIII).

E. Complexity analysis

The complexity of our methodology is linear in the number of instances I, as well as in the number C of considered compression levels. In our case, $C = log_2 K$ (K being the biggest considered codebook size), since the number of codebooks levels is halved at each optimization iteration. In the worst case, all batch clustering steps fail in compressing the target layers while preserving accuracy, and the batch size N always reaches 1, meaning that each layer is considered in isolation. In this case, the complexity becomes linear also in the number L of layers of each CNN, resulting in $\mathcal{O}(ICL)$.

Nevertheless, our sensitivity-based layers selection and a relatively large initial batch size N support effective run-time reductions. Indeed, in the best case, the model can be effectively compressed, preserving the initial batch size in all iterations, and reducing complexity to $\mathcal{O}(\frac{ICL}{N})$. Finally, the imposed accuracy threshold plays an important role in this analysis, since very limited acceptable drops (i.e., less than 1%) reduce the likelihood of successful compressions of a large batch of layers in parallel. Hence, multiple batch size reductions affect the run-time of the optimization algorithm from early iterations.

F. Effects on performance

Counter-intuitively, the presented methodology results in considerable reduction in the run-time required for inference, as showcased in Section V. Indeed, mask and a shift operations are required to retrieve an index of arbitrary bitwidth from a word (usually composed by 32 or 64 bits) read from memory. Then, an additional level of indirection is introduced, as this index is used as an address in a look-up table storing the appropriate codebook. However, codebooks are usually small, and are therefore retrieved from local cache memories. Moreover, thanks to their reduced bitwidth, multiple indexes are read at each word access in main memory, thus reducing the number of slow read operations.

IV. Experimental Setup

We consider benchmarks of different complexity to evaluate our methodology. We tested LeNet-5 [19], AlexNet [20], VGG16 [20], GoogLeNet [21], ResNext [22] and MobileNet [23] on the CIFAR100 dataset [24]. The proposed benchmarks have been implemented, trained and evaluated in PyTorch [25]. The training phase consists in 200 epochs using the Adam optimizer [26] and fixed hyper-parameters common to all benchmarks, including a weight decay of 10^{-4} and a learning rate of 10^{-3}. For all the evaluated benchmarks, the obtained baseline accuracy is in line with the one reported in literature and reflects the non-triviality of the proposed classification tasks.

TABLE I
Accuracy of single-instance and E^{2}CNNs models evaluated on the CIFAR100 dataset.

Benchmark	Accuracy		Accuracy Gain
	Single Inst.	E^{2}CNNs	
LeNet-5	42.56%	43.63%	1.07%
AlexNet	55.84%	61.16%	5.32%
VGG16	69.24%	74.18%	4.94%
GoogLeNet	72.62%	76.48%	3.86%
ResNext	75.33%	78.09%	2.76%
MobileNet	64.61%	67.74%	3.13%

We tested E^{2}CNNs implementations composed of $I = 2, 4,$ and 8 instances, showing in Section V only configurations that maximise classification accuracy. Table I motivates our choice by comparing the accuracy of baseline CNN models with ensemble-based designs. It showcases E^{2}CNNs accuracy improvements from 1% up to more than 5%, in line with the results presented in [13]. The pruning step has a critical impact on the very small structure of LeNet-5, largely reducing the individual instances' accuracy and consequently limiting the gains of the ensemble.

Weights clustering is performed adopting the k-means function provided by Scikit-learn [27], using an initial uniform clusters distribution that better protects the initial classification accuracy by preserving high magnitude parameters, as suggested in [7]. Ten fine-tuning epochs are run in each clustering-based compression phase. A second execution of k-means is then employed to cluster weight values to codewords after fine-tuning. We set the initial layers batch size N equal to half of the total number of layers, a setting resulting in a good compromise between achieved compression and execution time. Besides the aforementioned Python implementation, a C++ inference solver is adopted to measure performance using the built-in *clock_gettime()* function.

As a first baseline, we consider uniform quantization, in which all weights are represented as values with the same number of bits. We also compare with an alternative employing uniform codebooks, having the same codebook size for each layer. Such baseline strategy is based on the methodology described in [28], removing the constraint of having symmetric weights and employing a layer-based incremental strategy.

V. Experimental results

We first compare a uniform codebook-based compression approach with an equivalent alternative based on quantization. Then, we showcase the benefits of adopting the heterogeneous accuracy-driven approach characterizing our methodology. To this end, we evaluate compression and inference performance.

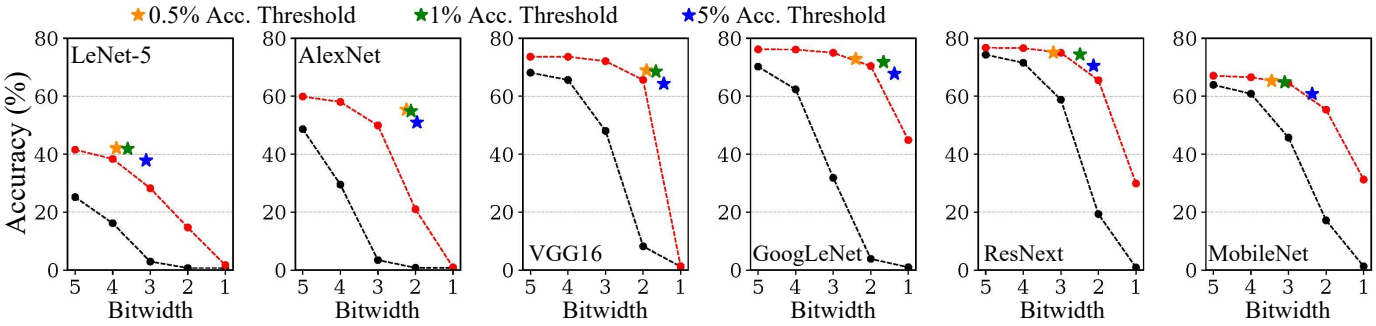

Fig. 3. Accuracy/bitwidth trade-offs for different compression methods. Uniform quantization (black lines), compared to uniform codebooks (red lines) and our heterogeneous compression methodology (stars, evaluated for three different accuracy thresholds). Bitwidths refer to values for quantization approaches and indexes for codebook ones. Average values are depicted for the heterogeneous strategy.

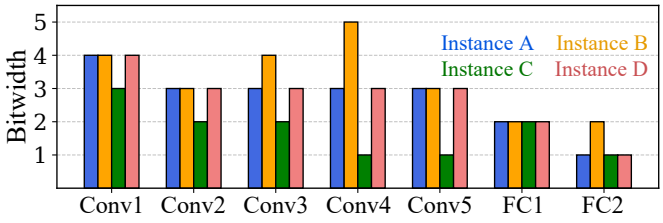

Fig. 4. AlexNet-based 4-instances optimized ensemble. Each instance (A,B,C,D) has layers with differently sized codebooks (internal heterogeneity). The codebook sizes also varies among instances (external heterogeneity).

A. Homogeneous codebook-based compression.

Fig. 3 compares uniform codebook-based compression (red lines) with uniform quantization (black lines) on different benchmark CNNs. Results highlight that codebooks-based solutions outperform quantization, resulting in superior accuracy levels for the same bitwidths employed to represent either values (for quantized solutions) or indexes (for codebook ones). In turn, bitwidths determine compression levels, and ultimately model sizes. For an accuracy threshold of 1%, the uniform codebook-based strategy improves compression by an average of 29.16% when compared to quantization.

The superior accuracy of codebook-based approach derives from their higher flexibility, as codewords adapt to the distribution of the weight values during the clustering process. Indeed, one codeword can cluster a few values in a large data range (typically at the tails of the weights values distribution). Conversely, another codeword may cluster many values in a small data range.

Such behaviour is induced by the k-means algorithm and is not achievable with quantization, since in this case any touple of consecutive weights differ by the same fixed extent.

B. Heterogeneous codebooks: compression/accuracy trade-offs.

Since it does not provide a way to control accuracy in the compression loop, the uniform codebook-based approach results in sudden accuracy drops when the size of all codebooks is aggressively reduced beyond a critical point. Moreover, the maximum compression preserving a certain accuracy level cannot be discerned a priori, being it tightly related to the CNN being optimized. Our proposed heterogeneous and sensitivity-driven strategy addresses these shortcomings.

To showcase its performance, we evaluate it considering three accuracy thresholds of 0.5%, 1% and 5%. Results are shown in Fig. 3 as yellow, green and blue star markers, respectively. Such implementations improve compression across all benchmarks by an average 22.27%, 8.31% and 13.13% for accuracy thresholds of 0.5%, 1% and 5%, when compared to uniformly compressed alternatives. Moreover, considering single-instance floating-point baselines, our approach enables average compressions of 11x, 13x and 15x, for the three evaluated accuracy thresholds.

Improvements in accuracy/compression trade-offs derive from the adoption of different compression schemes in the CNN layers. Thus, a deeper compressions (i.e., smaller codebooks) is only applied in layers exhibiting a higher degree of robustness layers. To exemplify the outcome of our framework, Fig. 4 showcases the achieved weight bitwidths in each layer of an optimized four-instances ensemble based on AlexNet.

C. Heterogeneous codebooks: inference run-time.

We measured the time required for an inference of the baseline floating-point implementation of the benchmarks, and compared results to heterogeneous codebook-based implementations with an accuracy degradation threshold of 1%. Speedup results are presented in Fig. 5 for five different systems-on-chip: SiFive Freedom U740 (based on the RISC-V ISA), ARM Cortex A9 MPCore, AMD EPYC 7551, Intel Xeon Gold 6154, Intel i7-4790, and Intel i9-10900K. The measured gains are strongly correlated to the reduction of memory accesses in our optimized benchmarks, which, on one side, decreases the energy-per-inference and, on the other side, reduces the probability of cache misses, with evident benefits in terms of execution times. Improvements are more substantial in high-performance CPUs, for which the clock cycles cost of memory accesses is far more expensive than MAC execution. Therefore, the achieved reduction of memory operations limits the number of cache misses, thus producing very high speedups. Average improvements range from less than 5% on RISC-V and ARM processors, larger than 80% on high-performance Intel CPUs.

Our results consider the overhead of the shift and mask operations required to extract the bit fields corresponding to low-bitwidth indexes from memory words, as well as the time

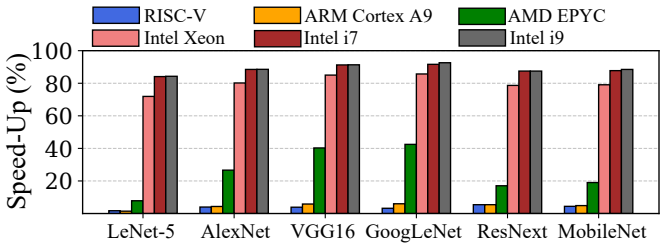

Fig. 5. Speedups of our methodology with respect to floating-point implementations, as measured on different processors.

TABLE II
COMPARISONS WITH PREVIOUS STATE-OF-THE-ART WORKS.

Benchmark	Reference			Our Solution	
	Work	Drop	Compr.	Drop	Compr.
LeNet-5 - MNIST	[9]	0.89%	1.72 bits	0.89%	**1.52 bits**
LeNet-5 - MNIST	[7]	0.2%	3 bits	0.89%	**1.52 bits**
VGG16 - CIFAR10	[6]	0.84%	3 bits	0.82%	**1.31 bits**

required to look up floating-point codewords corresponding to an index. The obtained performance gains suggest that the impact on performance of indexes extraction is low compared to the improvements achieved by reducing memory accesses.

D. Comparison with state-of-the-art.

A comparative analysis with state-of-the-art approaches is provided in Table II. Note that, only for results in this table, we targeted less challenging datasets (MNIST and CIFAR10) to allow for an apples-to-apples comparison. Reported results related to our methodology have been retrieved by imposing a 1% accuracy threshold. We report the most comparable figures for Wu et al. [9], Han et al. [7], and Jain et al. [6], as in these studies the accuracy degradation is not directly controlled. By employing a layer-based compression strategy, our methodology outperforms the uniform approaches in [6] and [7], where a further reduction of weight bitwidths (i.e., 2 bits) in all layers would significantly reduce classification accuracy. Note that our methodology improves on [7], which also uses pruning and weights encoding. Moreover, the compression achieved in [9] only refers to the second convolutional layer, while the remaining layers are not optimized.

VI. CONCLUSIONS

In this work, we have presented a novel accuracy-driven CNN design methodology that applies an accuracy-driven strategy where variable-size codebooks are employed to compress and speedup CNNs. First, we constrain the number of unique weights values employed in each layer so that values can be stored in codebooks. Then, compression is achieved by representing the weights of CNN models as low-bitwidth indexes in codebooks. Our methodology leverages the increased robustness provided by ensembling. It also supports heterogeneous codebook sizes both in different layers of an ensemble instance and across instances. Indeed, we showcased that, for 1% accuracy degradation, our approach compresses CNN model sizes up to 19x. Moreover, our methodology improves inference run-time performance up to 91.6% compared to uncompressed floating-point implementations.

REFERENCES

[1] S. Bianco *et al.*, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, 2018.

[2] A. Rodríguez *et al.*, "Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework," *MDPI. Sensors*, 2018.

[3] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015.

[4] H. Li *et al.*, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[5] Y. Wei *et al.*, "Quantization mimic: Towards very tiny cnn for object detection," in *Proceedings of the European conference on computer vision (ECCV)*, 2018.

[6] A. Jain *et al.*, "Symmetric k-means for deep neural network compression and hardware acceleration on fpgas," *IEEE Journal of Selected Topics in Signal Processing*, 2020.

[7] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[8] Y. Gong *et al.*, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[9] J. Wu *et al.*, "Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions," in *PMLR. International Conference on Machine Learning*, 2018.

[10] B. W. Denkinger *et al.*, "Impact of memory voltage scaling on accuracy and resilience of deep learning based edge devices," *IEEE Design & Test*, 2019.

[11] Y. LeCun *et al.*, "Optimal brain damage," in *Advances in neural information processing systems*, 1990.

[12] J.-H. Luo *et al.*, "An entropy-based pruning method for cnn compression," *arXiv preprint arXiv:1706.05791*, 2017.

[13] F. Ponzina *et al.*, "E2CNNs: Ensembles of convolutional neural networks to improve robustness against memory errors in edge-computing devices," *IEEE Transactions on Computers*, 2021.

[14] J. Wu *et al.*, "Quantized convolutional neural networks for mobile devices," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, 2016.

[15] M. Courbariaux *et al.*, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.

[16] C. N. Coelho *et al.*, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, 2021.

[17] F. Ponzina *et al.*, "A flexible in-memory computing architecture for heterogeneously quantized cnns," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021.

[18] L. Rokach *et al.*, "Clustering methods," in *Springer. Data mining and knowledge discovery handbook*, 2005.

[19] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann.lecun.com/exdb/lenet*, 2015.

[20] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012.

[21] C. Szegedy *et al.*, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI Conf. on Artificial Intelligence*, 2017.

[22] X. S *et al.*, "Aggregated residual transformations for deep neural networks," 2017.

[23] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[24] A. Krizhevsky *et al.*, "The CIFAR-10 dataset," *URL: http://www.cs.toronto.edu/kriz/cifar.html*, 2014.

[25] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, 2019.

[26] D. P. Kingma *et al.*, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *JMLR. The Journal of machine Learning research*, 2011.

[28] A. Zhou *et al.*, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.