A Heuristic Exploration of Retraining-free Weight-Sharing for CNN Compression

Etienne Dupuis, David Novo, Ian O'Connor, Alberto Bosio

A Heuristic Exploration of Retraining-free Weight-Sharing for CNN Compression

Etienne Dupuis*, David Novo†, Ian O'Connor*, and Alberto Bosio*
*Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270
{etienne.dupuis, ian.oconnor, alberto.bosio}@ec-lyon.fr
†LIRMM, Univ Montpellier, CNRS, Montpellier, France
david.novo@lirmm.fr

***Abstract*—The computational workload involved in Convolutional Neural Networks (CNNs) is typically out of reach for low-power embedded devices. The scientific literature provides a large number of approximation techniques to address this problem. Among them, the Weight-Sharing (WS) technique gives promising results, but it requires carefully determining the shared values for each layer of a given CNN. As the number of possible solutions grows exponentially with the number of layers, the WS Design Space Exploration (DSE) time can easily explode for state-of-the-art CNNs. In this paper, we propose a new heuristic approach to drastically reduce the exploration time without sacrificing the quality of the output. The results carried out on recent CNNs (GoogleNet [1], ResNet50V2 [2], MobileNetV2 [3], InceptionV3 [4], and EfficientNet [5]), trained with the ImageNet [6] dataset, show over $5\times$ memory compression at an acceptable accuracy loss (complying with the MLPerf [7] quality target) without any retraining step and in less than 10 hours. Our code is publicly available on GitHub [8].**

***Index Terms*—Convolutional Neural Network, Deep Learning, Computer vision, Hardware Accelerator, Design Space Exploration, Approximate Computing, Weight-Sharing**

I. Introduction

Today, Deep Neural Networks (DNNs) are very powerful tools. They are mostly used in computer vision and natural language processing, in various daily life applications from smartphone facial recognition to voice assistants as well as safety-critical applications like autonomous driving. Recent DNNs are articulated around the use of multiple chained convolution layers, called Convolutional Neural Networks (CNNs), using a dot product between the inputs and the learned filter weights. However, the outstanding performance achieved by CNNs comes at the cost of very large computational requirements, making them out of reach for most of the low-power embedded devices [9]. As an example, the energy required to run real-time object classification on a smartphone drains the battery in one hour [10]. Usually, the CNN computation requirements are measured in terms of the memory footprint required to store the weights/activations values and the number of Multiply-Accumulate (MAC) operations required to compute neuron outputs during inference. However, most of the energy cost during CNN inference comes from memory access as analyzed in [9]. It is thus important to reduce memory footprint to improve energy efficiency.

Novel computing paradigms and emerging technologies are under investigation to make CNNs accessible to low-power devices [11], [12]. Among them, the Approximate Computing (AxC) paradigm [13] leverages the inherent error resilience of CNNs to improve energy efficiency by relaxing the need for fully accurate operations. CNNs have a high degree of redundancy in terms of their structure and parameters [14], and this redundancy is not always necessary for an accurate prediction.

Pruning, quantization and Weight-Sharing (WS) are among the most popular approximation techniques for CNN compression and acceleration. They focus on approximating the values and the representations of weights and/or activations (i.e., the inputs and outputs of the layers). Pruning techniques [15] remove the least important weights of a CNN to reduce memory footprint. In some cases, pruning is applied to a complete structure (e.g., a channel) providing more opportunities for acceleration [15]. On the other hand, quantization [16] and WS [14] techniques, focus on reducing the number of distinct values for weights and/or activations. Quantization allows acceleration (i.e., it reduces the computational cost of MAC operations) by using a more compact data type representation, such as 8-bit fixed-point numbers, while WS (the focus of this paper) alleviates the memory footprint by reducing the number of different weight values yet maintaining their original data type representation.

Although the standard application of WS [14], [17]–[22], requires the retraining of the network to recover accuracy loss, it has been proven [23] that it is also possible to optimize the number of shared values to each layer's resilience while avoiding the costly retraining step. However, due to the increasing complexity of modern CNN topologies, the Design Space Exploration (DSE) required to find optimal WS approximations is prohibitively costly. In this paper, we propose a heuristic approach to achieve a scalable retraining-free WS compression.

The main contributions of this work can be summarized as:

1) The study of the complexity of the design space exploration of weight-sharing in CNNs.
2) A novel automatic two-step heuristic optimization to retraining-free weight-sharing.
3) A comparison with the state-of-the-art WS techniques and a post-training pruning technique [24] in terms of compression vs. accuracy results.

The remainder of this paper is structured as follows: after giving background information and discussing the state of the art in Section II, the proposed method is detailed in Section III. Section IV presents the experiments that have been conducted and the obtained results. Finally, Section V concludes the paper.

II. Weight-sharing and related works

In this section, we provide a short background on WS, discuss the related work, and position our approach.

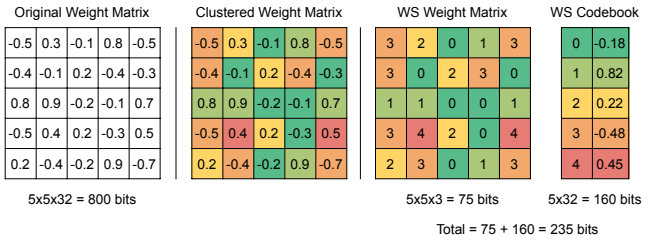

Fig. 1. Weight-Sharing example for a 5x5 matrix with weight values grouped in 5 different clusters.

WS background. WS aims to group weights into buckets or clusters sharing the same value. It allows a significant reduction of the CNN memory footprint by storing shared values in a dedicated codebook, where original weight values in the weight matrix are replaced by their corresponding indexes. The memory footprint reduction, defined as *compression*, is due to the indexes being encoded with fewer bits. For example, considering a FP32 baseline, if one can reduce the number of different shared values (k) to 256, the indexes can be encoded with only $log_2(k) = 8$ bits, allowing a $4\times$ compression compared to the original 32-bit counterpart. Shared weight values can be determined by using a clustering algorithm like K-means. Fig. 1 shows an example of applying the K-means clustering algorithm. From the initial set of 25 values, which are reduced to 5 shared values after applying clustering, there is a reduction from 800 to 235 bits due to the use of 3-bit indexes and only 5 32-bit weights, resulting in a $3.4\times$ compression. This naive example involves a very small weight matrix, and thus, the codebook size is twice the weight matrix with the indexes. However, in real-life CNNs, the codebook size is usually negligible compared to the size required to store all the indexes (in the order of millions) of a CNN layer.

Related works. In the literature, there are many practical applications of WS. Existing solutions differ by (1) the use of a meta-heuristic technique for clustering, (2) the implementation of the WS codebook used to store the shared values, and (3) the algorithm used to choose the number of shared values (clustering).

Studying the clustering algorithms used in previous work, it appears that most [14], [20], [25] rely on the K-means [26] approach and apply clustering after the training phase. Alternatively, another group of solutions [17]–[19], [21] act during the CNN training to force weights values to be concentrated and thus facilitate the clustering. There are also examples of non-K-means based clustering [14], [22]. In this paper, we target WS after training, and we use the K-means algorithm. However, since our main target is the automatic exploration of the optimal number of shared values, any clustering algorithm could be used.

Regarding the implementation, most methods [14], [25] focus on the compression of the weight matrix. Others also look for acceleration and thus propose to reduce the computational complexity by implementing look-up table-based multiplication or computation reuse, benefiting from the reduced number of different values [20], [22], [27], [28]. As the main focus of this work is the tuning of the number of shared values, only compression is taken into account.

Regarding the selection of the number of shared values, some existing methods [17], [22] manually tune the number of shared values to achieve the desired *Compression Rate* (CR). The main problem with manual tuning is that it is costly and requires expert knowledge. Other methods [14], [18], [19] use a fixed number of shared values. However, the most widely adopted solution is to homogeneously vary the number of shared values (i.e., use the same number for each layer) to explore the trade-off between compression and precision [20], [27]–[29]. However, this approach is clearly suboptimal, as the weight sharing factor is not adapted to the differences in tolerance-to-approximation of each layer.

Our work. In practice, the number of shared values needs to be tuned for each layer, requiring a complex design space exploration. The main goal of this paper is to automatically select the optimal number of shared values per layer. Our prior works [23], [30] address this problem incrementally by studying the resilience of each layer to WS, keeping previous layers approximated at their lowest *Accuracy Loss* (AL) (i.e., the absolute difference in top-1 accuracy between a baseline and an approximated CNN). However, the main issues with such local optimization approaches are the avoidance of local optima and the lack of network-wide metrics for CR and AL. Instead, in this work, we propose a completely new two-step heuristic approach able to significant speedup the exploration time of the global solution space.

An important particularity of our approach is its training-free nature. Almost all existing methods require retraining or fine-tuning after the WS step. Such post-processing has the disadvantage of being computationally intensive and thus costly, hard to set up, and sometimes not possible due to the lack of access to the training dataset. Indeed, the growing concerns on privacy and security will certainly restrain the access to training datasets in important application domains. Conversely, the proposed heuristic only requires a small validation dataset and significantly lower computation time. Another advantage of retraining-free methods is that due to their faster execution time, it is possible to explore multiple trade-offs between AL and CR. For comparison, DP-NET [18] requires 30 epochs to compress GoogleNet [1] on the ImageNet [6] dataset. Considering that each training epoch takes almost 1 hour on our Tesla V100 GPU using TensorFlow [31], the compression of a single approximated CNN version amounts to almost 30 hours. Instead, our proposed method takes only 10 hours to compress GoogleNet on the same GPU and outputs a full set of Pareto-optimal approximated versions.

III. Weight-sharing optimization

In this section, we formulate the WS optimization problem and explain the proposed two-step heuristic approach in detail.

For a given CNN with N layers, let k_i be the number of shared values of the layer i. k_i is bounded to a set of values k_{range}. Accordingly, a CNN approximated with WS can be characterized by its $k_{tuple} = \{k_i\} \forall i \in [1, N]$ representing the number of shared values for each layer. From the k_{tuple} it is possible to measure both (1) the AL, by scoring the CNN (i.e., running the inference of the approximated CNN and compute the difference between the top-1 accuracy of the approximated CNN w.r.t. the reference) on a large and representative test dataset, and (2) the CR, using Eq. 1:

$$CR = \frac{W \times b_{values}}{W \times b_{index} + |k_{tuple}| \times b_{values}}, \quad (1)$$

where W is the number of weights of the network, b_{values} the number of bits used to represent weight values, and $b_{index} = log_2(k_{tuple})$ the number of bits used to represent a WS codebook index.

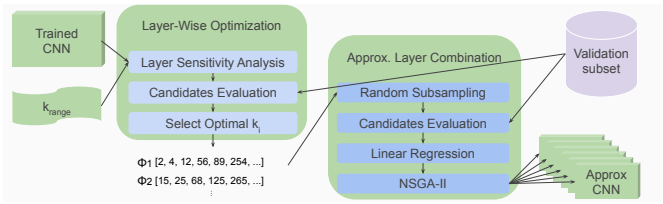

Fig. 2. Conceptual view of our weight-sharing optimization heuristic.

Exhaustively exploring every possible k_{tuple} results in $\mathcal{O}(|k_{range}|^N)$ complexity. As an example, a toy CNN like LeNet-5 [32], with $N = 5$ layers and a $k_{range} = [1, 256]$, results in a number of scoring steps equal to $256^5 = 1.1 \times 10^{12}$. Thus, the exploration of the complete solution space would require more than three decades when considering an optimistic $1\ ms$ evaluation step. To make things worse, the complexity raises exponentially with N, which is more than a factor of ten higher in recent CNNs.

To make the optimization problem tractable, we propose to adopt a divide and conquer strategy, splitting the exponential complexity problem of optimizing the number of shared values for each layer in two sub-problems: (1) finding ϕ_i, the set of optimal k_i for each layer of the network and (2) finding the optimal k_{tuple} as combinations of ϕ_i w.r.t. AL and CR. Although the second sub-problem still exhibits an exponential complexity, its solution space is significantly reduced after solving the first sub-problem. We propose using a known meta-heuristic to solve the second sub-problem. Fig. 2 shows a conceptual view of the proposed heuristic. The input is a trained CNN and a range of possible numbers of shared values. The output is a set of Pareto efficient approximated CNNs representing optimal trade-offs between AL and CR. The figure also shows how each sub-problem is solved in multiple sub-steps. In the *Layer-Wise Optimization* step, each layer is locally optimized. Then, the best approximate candidates of each layer (ϕ_i) are combined to find Pareto optimal approximated CNNs in the *Approximated Layer Combination* step.

Complementary to the DSE complexity reduction, it is also possible to use heuristic-based estimation methods to avoid the costly scoring step. Some proxies for the scoring step of individual layers have been evaluated in [23]. For instance, the *inertia*, defined as the sum of the distances between shared values and original values, shows good accuracy at quickly evaluating the quality of the shared weight values. Finally, to reduce the scoring time of the complete CNN, it is possible to use a subset of the evaluation dataset.

A. Layer-wise optimization

The first sub-problem (Layer-Wise Optimization in Fig. 2) targets the local optimization of each layer. The goal is to find ϕ_i, the set of optimal k_i values, for each layer of the CNN. ϕ_i is characterized by (1) the AL obtained when layer i is approximated with k_i shared values while the other layers are not approximated, and (2) the number of bits required to store each index, i.e., $b_{index} = ceil(log_2(k_i))$. We use b_{index} as an optimization objective as it is indirectly proportional to the CR, see Eq. 1. The AL is obtained using a small subset[1] of the validation dataset. Our algorithm explores one by one the k_i in the user-defined k_{range}

[1]Less than 10% works well for the tested CNNs.

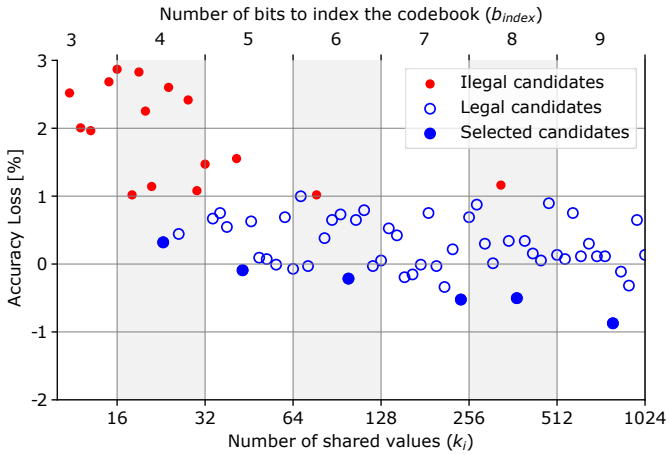

Fig. 3. Results of the Layer-Wise exploration on the first layer of the ResNet50V2 [2], AL is obtained on a small subset of the validation set (10%).

and for each, it evaluates both AL and b_{index} to then compose ϕ_i by selecting the candidate with the best AL per b_{index}. The complexity of this search is linear w.r.t. the number of layers, being $\mathcal{O}(N * |k_{range}|)$ and the size of the output set of optimal k_i is bounded as follows: $|\phi_i| \leq ceil(log_2(max(k_{range}))$.

Fig. 3 shows the output of the layer-wise approximation step for the first layer of the ResNet50V2 [2] as an example. We use MLPerf [7] to set a quality target (maximal allowed AL) at 99% of the baseline top-1 accuracy (98% for light models like MobileNetV2 [3]). In the figure, illegal candidates (red circles) are the solutions leading to an AL greater than the quality target, while the legal candidates (empty blue circles) are the solutions within the AL constraint. Among the legal candidates, the selected candidates (full blue circles) are those leading to the best AL within each b_{index}. As shown in the figure, the number of ϕ_i is significantly lower than the whole set of legal candidates.

B. Approximated layer combination

The second sub-problem targets the optimal k_{tuple} combinations of ϕ_i. Although in the previous step we have greatly reduced the number of candidates to be combined, the complexity of the exhaustive exploration is still $\mathcal{O}(\prod_{i=1}^{N+1} |\phi_i|)$ and thus exponential to the number of layers N. To address this problem, we propose the use of a meta-heuristic algorithm, like the genetic algorithm NSGA-II [33], to find a set of Pareto optimal k_{tuple} w.r.t. AL and CR.

The bottleneck of the genetic algorithm is the AL evaluation cost since it requires the scoring of the CNN. Thus, we propose to reduce this cost by using a proxy-based estimation. We use an analytical model to approximate AL estimation from k_{tuple} without requiring the CNN scoring. Although it is possible to use the inertia as a proxy metric for a single layer [23], we still need to combine the inertia values of each layer to provide an AL estimation that aggregates the contribution of each layer. Accordingly, we build a linear regression model that takes as input the inertia values of each approximated layer and gives as output an AL prediction. First, we train the regression model on a representative population of candidates (annotated with the AL and inertia of each layer) sampled from all the possible k_{tuple}. Then, we integrate our regression model in the NSGA-II algorithm. In order to evaluate the quality of the proposed method we compare the results of two versions of

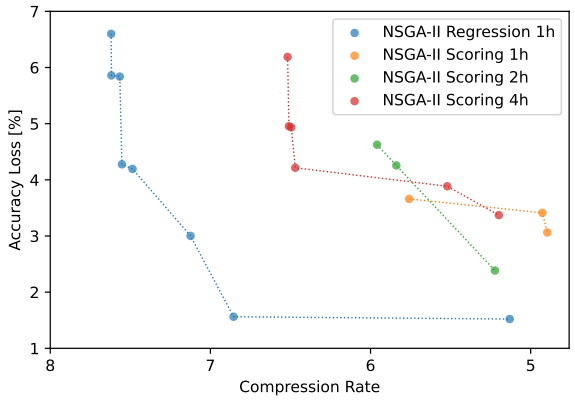

Fig. 4. Accuracy vs. compression results when executing the NSGA-II with and without the linear regression model. The latter is run multiple times with different time limits. The results are obtained using ResNet50V2 [2] on the ImageNet [6] dataset.

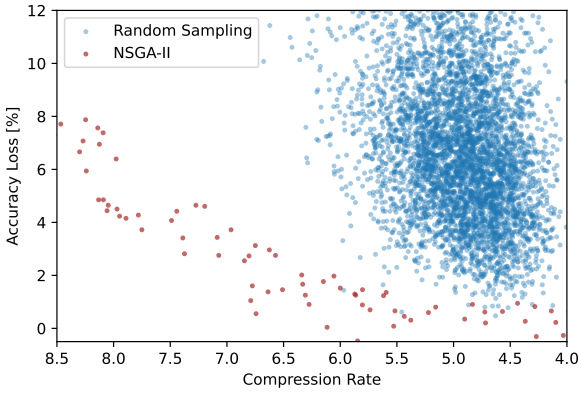

Fig. 5. Comparing WS ResNet50V2 [2] solutions obtained from random sub-sampling and from a NSGA-II optimization (population size = 100, number of generations = 500) with a linear regression model trained with the random solutions. The AL of each solution is obtained from conventional scoring.

the NSGA-II algorithm considering different limits in the execution time. One version uses our linear regression model and runs for 1 hour, while the other version uses the regular AL scoring method to select the best design candidates and includes three runs timed out at 1, 2, 4 hours, respectively. Fig. 4 shows the Pareto optimal CNN approximations found after running the two versions of the NSGA-II algorithm. Clearly, the version using the linear regression model is able to converge significantly faster towards a Pareto efficient front.

The linear regression model is described by the formula: $AL_{k_{tuple}} = \sum_{i=1}^{N+1} \alpha_i * inertia_{k_i}$, with $inertia_{k_i}$ being the reported inertia during the local optimization for the layer i, and α_i the prediction model coefficients. The linear regression model modulates the approximation error introduced at each layer by limiting the number of shared values. We have evaluated the use of the local AL from the layer-wise optimization step, the inertia, and the number of shared values k_i, as inputs to the model. Interestingly, there is little to no difference between the linear regression model using inertia or local AL as input, achieving both an r^2 score of 76% when testing using a validation split of 20% of the training samples. This demonstrates that a simple linear regression model using either inertia or local AL values as inputs can achieve a reasonably high modeling accuracy. Instead, when using the number of shared values k_i as input to the regression model, the r^2 score drops to 10.3%, indicating that this metric is poorly correlated with the CNN AL. Based on these results and considering its lowest computational cost, we select inertia as the proxy metric for the results shown in the rest of the paper.

Training a regression model requires a certain number of representative inputs. The simplest way to obtain these inputs is to randomly sample the solution space, apply WS to each sample, and score the resulting approximated CNNs. The results of such a random sub-sampling are presented in blue in Fig. 5. Each sample has been obtained by randomly selecting a k_{tuple} combination of $k_i \in \phi_i$ space. For each sample, the AL is evaluated by scoring on 10% of the validation dataset samples, while the CR is evaluated by using Eq. 1. The random samples are then ordered based on their AL and we select the most accurate samples[2] to train the regression model that is then used to quickly estimate the AL of candidates during the NSGA-II optimization. Using NSGA-II allows fast improvement of the optimal Pareto set of solutions as can be seen in Fig. 5. The figure presents the candidates obtained using both random sampling and the NSGA-II algorithm, illustrating that the proposed approach, based on the meta-heuristic optimization algorithm NSGA-II and using a linear regression model to avoid the costly scoring, can significantly improve the Pareto front.

IV. Evaluation

Our experiments were conducted on a single GPU server equipped with two Intel Xeon Silver 4210 and one NVIDIA Tesla V100. Most of the critical computations are accelerated by the GPU. Besides the ML functionality, the K-means algorithm is also executed on the GPU using the open-source code provided by Ding & al. [34] while the NSGA-II algorithm is executed on the host CPU.

A. Compression results of our weight-sharing technique

The same compression flow is applied to all CNNs with the following parameters: the dataset used for scoring during the exploration is a subset of the ImageNet [6] validation dataset containing 10% of the 50,000 samples. The k_{range} used during the layer-wise optimization is a logarithmic range with 100 values between 2 and 1024. The number of random samples used to train the linear regression model is 5,000 and the linear regression model is trained on the 30% best samples in terms of AL. The NSGA-II algorithm is set to run with a population of 100 candidates for 500 generations with default mutation values. All the reported AL values are measured on the entire validation dataset.

Table I presents the results on a representative set of image classification CNNs. They can be separated into two categories: (1) light CNNs are optimized to run on resource-constrained devices, while (2) heavy CNNs exclusively focus on top-1 accuracy. GoogleNet [1], ResNet50V2 [2] and InceptionV3 [4] belong to

[2]The top 30% works well for the tested CNNs.

TABLE I
COMPRESSION RESULTS ON DIFFERENT CNNS ON THE IMAGENET [6] DATASET UNDER MLPERF [7] QUALITY TARGET CONSTRAINTS

Network	MLPerf category	#Layer	Mem. [MB]	Top-1 Acc. [%]	CR min, max
GoogleNet	heavy	58	50	69.7	5.4
ResNet50V2	heavy	54	97	76.0	5.3, 5.6
InceptionV3	heavy	2.8	104	77.2	4.7, 5.3
MobileNetV2	light	53	13	71.9	4.4, 5.7
EfficientNetB0	light	82	20	76.4	4.5, 5.6
EfficientNetB1	light	116	30	78.4	4.3, 5.3
EfficientNetB2	light	116	35	79.8	3.5, 5.3

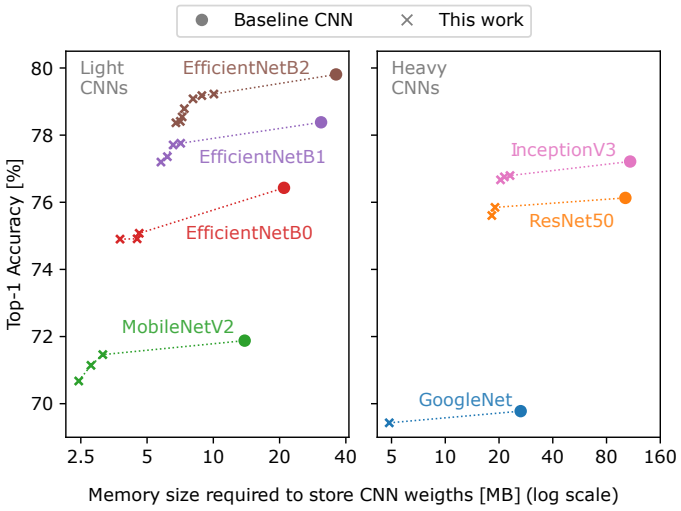

Fig. 6. Comparison of different approximated CNNs characterized by their top-1 accuracy and memory requirements on the ImageNet [6] dataset.

the heavy category, while MobileNetV2 [3] and the various EfficientNets [5] belong the light category. Following MLPerf [7] recommendations, each category has different quality targets (i.e., 99% and 98% of FP32 precision for heavy and light, respectively). The number of layers represents the number of fully connected and convolutional layers that are targeted by the proposed WS method, and indicates the complexity of the WS solution space, which grows exponentially with the number of layers. The reported memory is the size required to store the 32-bit weights of the baseline CNNs in memory. The top-1 accuracy indicates the baseline classification accuracy. Finally, the min. and max. CR values represent the minimum and maximum compression rates of the approximated CNNs found by our WS method within the corresponding quality target. On each of the tested CNNs, the proposed WS method is always capable of significantly compressing the baseline reference, by consistently achieving over $5\times$ higher CR. Importantly, the compression is achieved without requiring the intervention of an expert for manually tuning the k_is and without involving any retraining, fine-tuning or calibration steps.

A graphical view of the results is shown in Fig. 6, with the light CNNs on the left and the heavy ones on the right. The figure depicts the association between the minimum memory size required to store all the weights and the top-1 accuracy of each CNN instance. Achieving the best trade-offs between the AL and the CR, often requires exploring multiple CNN topologies with multiple levels of approximation. Accordingly, our method finds Pareto optimal trade-offs between accuracy and memory footprint to facilitate the selection of the most suitable CNN instance for any given application.

TABLE II
COMPARISON WITH OTHER WS TECHNIQUES ON GOOGLENET.

Method	**Retraining-Free**	**CR**	**Top-1 AL (%)**
Deep K-means (2018) [17]	**Yes**	1.5	1.22
	Yes	2.0	3.70
	Yes	3.0	13.72
	Yes	4.0	48.95
	No	1.5	0.26
	No	2.0	0.17
	No	3.0	0.36
	No	4.0	1.95
DP-Net (2020) [18]	No	7.0	-0.30
	No	10.0	1.56
FastWS [23]	**Yes**	4.6	0.83
This Work	**Yes**	5.4	0.35

B. Comparison with prior weight-sharing works

To assess the quality of the proposed method with respect to existing works, we have identified three recent WS methods using GoogleNet on the ImageNet [6] dataset. The first method, named Deep K-means [17], proposes the use of a regularization term to force weights to concentrate during the training, using a fixed cluster rate to obtain the number of shared values for each layer based on the layer weight counts. They report both the AL and the CR for GoogleNet using their method with and without retraining. The second method, named DP-Net [18], proposes the use of Dynamic-Programming instead of the K-means algorithm for the clustering as well as a regularization term. They use a fixed and homogeneous number of shared values for each layer and have only reported results using retraining. The third and most related method, named FastWS [23], optimizes the number of shared values layer by layer without retraining.

Table II summarizes the comparison. It shows that the result obtained with the proposed method Pareto-dominates the results obtained with retraining-free methods: our result simultaneously achieves the highest CR and the lowest AL. When compared with the results of Deep-K-Means [17] using retraining, our method is still Pareto-optimal and dominates all results with the exception of that achieving an AL of 0.17% and a CR of $2\times$. Arguably, our result still seems more attractive as it achieves $2.7\times$ higher compression for just 0.18% higher accuracy degradation. Finally, our result is Pareto-dominated by the results of the DP-Net [18] method. However, these results require extensive retraining, a very computationally costly task which is not always possible due to privacy and security constraints on training the datasets. Thus, our method remains the best option whenever retraining is not feasible.

C. Comparison with post-training pruning

In this section, we compare the proposed method with Post-Training Pruning (PTP) [24], a state-of-the-art technique that achieves compression via pruning. Among our selected CNNs, PTP includes compression results on ResNet50 [35] and MobileNetV2 [3]. In detail, PTP proposes a data-free weight pruning

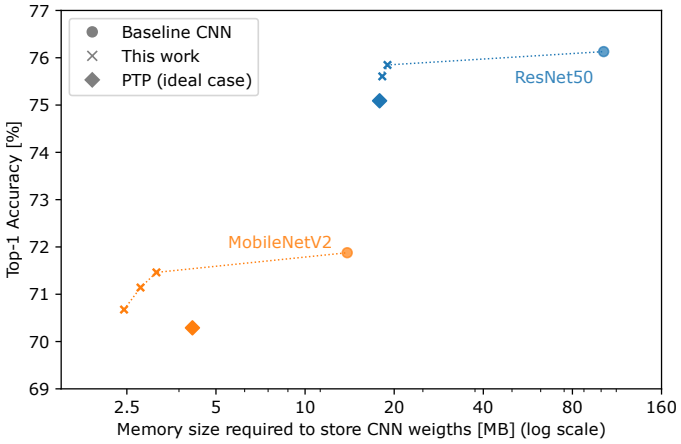

Fig. 7. Comparison of obtained approximated CNNs with the proposed method and PTP [24]. CNNs are characterized by the absolute AL measured on the ImageNet [6] and CR compared to the baseline full precision version.

approach based on automatically-generated synthetic fractal images for retraining. This pruning step is then followed by post training quantization, resulting in an 8-bit sparse matrix representation. Theoretically, this would lead to a $11.4\times$ and $6.7\times$ CR for ResNet50 and MobileNetV2, respectively. However, in practice, it is also necessary to store indexes next to the sparse weight. Without those indexes, it would be impossible to recover the position of the weights. Accordingly, in our comparison we consider the ideal case of just adding one 8-bit index per weight. Fig. 7 shows how PTP compares with respect to our method. The figure shows that the PTP solution is Pareto-dominated in MobileNetV2 and marginally Pareto-optimal in ResNet50. Considering that the PTP compression estimates are based on very optimistic assumptions, we conclude that our method provides better accuracy-compression trade-offs.

V. Conclusion

This paper presents a novel method to weight-sharing optimization for CNN compression. We show that our method can efficiently explore the large weight-sharing design space to produce a set of solutions that offer very interesting trade-offs between accuracy and model compression. These solutions achieve more than a $5\times$ compression over the baseline memory footprint in multiple state-of-the-art computer vision CNNs on the challenging ImageNet dataset while also complying with the MLPerf [7] quality target constraints. Importantly, these compression results are achieved while avoiding the prohibitively costly retraining step, which is commonly used in prior works. To facilitate the reproduction of our results, our code is publicly available on Github [8].

In future work, we plan to study how our method combines with structured pruning and quantization-aware training. We also plan to investigate the use of calibration for shared weights tuning and per-channel clustering to allow comparison with recent post-training quantization techniques.

VI. Acknowledgement

This work has been funded by the AdequatedDL project (ANR-18-CE23-0012) of the French National Research Agency (ANR).

References

[1] C. Szegedy *et al.*, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.
[2] K. He *et al.*, "Identity mappings in deep residual networks," *ArXiv*, vol. abs/1603.05027, 2016.
[3] M. Sandler *et al.*, "MobileNetV2: Inverted residuals and linear bottlenecks," *Proceedings of CVPR*, 2018.
[4] C. Szegedy *et al.*, "Rethinking the Inception architecture for computer vision," *Proceedings of CVPR*, 2016.
[5] M. Tan *et al.*, "EfficientNet: Rethinking model scaling for convolutional neural networks," *ArXiv*, vol. abs/1905.11946, 2019.
[6] J. Deng *et al.*, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of CVPR09*, 2009.
[7] P. Mattson *et al.*, "MLPerf training benchmark," 2020.
[8] E. Dupuis *et al.*, "A Heuristic Exploration to Retraining-free Weight-Sharing for CNN Compression," 11 2021. [Online]. Available: https://github.com/e-dupuis/retraining-free-weight-sharing
[9] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, 2017.
[10] T.-J. Yang *et al.*, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *Proceedings of CVPR*, 2017.
[11] Y. Ma *et al.*, "In-memory computing: The next-generation ai computing paradigm," in *Proceedings of GLSVLSI*, 2020.
[12] B. Shastri *et al.*, "Photonics for artificial intelligence and neuromorphic computing," *Nature Photonics*, vol. 15, pp. 102–114, 2020.
[13] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, vol. 48, no. 4, May 2016.
[14] W. J. D. Song Han, Huizi Mao, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv*, 2016.
[15] S. Anwar *et al.*, "Structured pruning of deep convolutional neural networks," *ACM JETC*, vol. 13, 2017.
[16] C. Baskin *et al.*, "UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks," *arXiv*, Apr. 2018.
[17] J. Wu *et al.*, "Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions," *ArXiv*, vol. abs/1806.09228, 2018.
[18] D. Yang *et al.*, "DP-Net: Dynamic programming guided deep neural network compression," *ArXiv*, vol. abs/2003.09615, 2020.
[19] Y. Hu *et al.*, "Cluster regularized quantization for deep networks compression," in *Proceedings of ICIP*, 2019.
[20] S. Son *et al.*, "Clustering convolutional kernels to compress deep neural networks," in *ECCV*, 2018.
[21] K. Ullrich *et al.*, "Soft weight-sharing for neural network compression," *ArXiv*, vol. abs/1702.04008, 2017.
[22] E.-V. Pikoulis *et al.*, "A new clustering-based technique for the acceleration of deep convolutional networks," 12 2020, pp. 1432–1439.
[23] E. Dupuis *et al.*, "CNN weight sharing based on a fast accuracy estimation metric," *Microelectronics Reliability*, vol. 122, 2021.
[24] I. Lazarevich *et al.*, "Post-training deep neural network pruning via layer-wise calibration," *ArXiv*, vol. abs/2104.15023, 2021.
[25] Y. Gong *et al.*, "Compressing deep convolutional networks using vector quantization," *ArXiv*, vol. abs/1412.6115, 2014.
[26] J. A. Hartigan *et al.*, "A k-means clustering algorithm," *JSTOR: Applied Statistics*, vol. 28, no. 1, pp. 100–108, 1979.
[27] J. Wu *et al.*, "Quantized convolutional neural networks for mobile devices," *Proceedings of CVPR*, 2016.
[28] M. S. Razlighi *et al.*, "LookNN: Neural Network with No Multiplication," in *Proceedings of DATE*, 2017.
[29] Y. Gong *et al.*, "Compressing Deep Convolutional Networks using Vector Quantization," *arXiv*, Dec. 2014.
[30] E. Dupuis *et al.*, "On the automatic exploration of weight sharing for deep neural network compression," *Proceedings of DATE*, 2020.
[31] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of OSDI*, 2016.
[32] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.
[33] K. Deb *et al.*, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, 2002.
[34] V. Markovtsev *et al.*, "src-d/kmcuda: 6.0.0-1," Feb. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.286944
[35] K. He *et al.*, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.