On the Automatic Exploration of Weight Sharing for Deep Neural Network Compression

Etienne Dupuis[1], David Novo[2], Ian O'Connor[1], and Alberto Bosio[1]

[1]*Ecole Centrale de Lyon, Institut des Nanotechnologies de Lyon, France*
[2]*LIRMM, Université de Montpellier, CNRS, France*
{etienne.dupuis, ian.oconnor, alberto.bosio}@ec-lyon.fr, and david.novo@lirmm.fr

Abstract—Deep neural networks demonstrate impressive levels of performance, particularly in computer vision and speech recognition. However, the computational workload and associated storage inhibit their potential in resource-limited embedded systems. The approximate computing paradigm has been widely explored in the literature. It improves performance and energy-efficiency by relaxing the need for fully accurate operations. There are a large number of implementation options with very different approximation strategies (such as pruning, quantization, low-rank factorization, knowledge distillation, etc.). To the best of our knowledge, no automated approach exists to explore, select and generate the best approximate versions of a given convolutional neural network (CNN) according to the design objectives. The goal of this work in progress is to demonstrate that the design space exploration phase can enable significant network compression without noticeable accuracy loss. We demonstrate this via an example based on weight sharing and show that our method can obtain a 4x compression rate in an int-16 version of LeNet-5 (5-layer 1,720-kbit CNNs) without re-training and without any accuracy loss.

Index Terms—Deep Neural Networks, Approximate Computing, Model Compression, Weight Sharing, Design Space Exploration, Embedded System, Hardware Accelerator

I. Introduction

Deep Learning, and in particular Convolutional Neural Networks (CNNs), are currently one of the most widely used predictive models in the field of machine learning. CNNs are used today in various applications such as object recognition, drug discovery and natural language processing, as well as safety-critical applications like autonomous driving. Unfortunately, the computational cost of CNNs is often out of reach for low-power embedded devices [1].

Novel computing paradigms and emerging technologies are under investigation to render CNNs sustainable and really usable by edge computing end users. Among them, the Approximate Computing (AxC) paradigm leverages the inherent resilience of CNNs to errors to improve energy efficiency, by relaxing the need for fully accurate operations.

CNNs have a high degree of redundancy in terms of their architecture and parameters. This observation has paved the way for a number of highly recognized approximation techniques [1]. The most popular ones include pruning [2], quantization [3], low-rank factorization [4], knowledge distillation [5] and weight-sharing [2]. We focus on the latter technique, which has already demonstrated impressive results.

The basic idea of weight sharing is to merge similar values and replace them with a single one, thus leading to a reduction in memory footprint. Weight sharing has already proved its efficiency to correctly represent weight ranges and values. In the literature, there are many practical applications of weight sharing. For example, Hashed net [6] proposes to randomly group weights into buckets sharing the same value prior to the training step. A more efficient method, involving a 3-step process and based on the retraining of a model, is presented in DeepCompress [2], achieving impressive results due to the use of various levels of redundancy in deep neural networks through pruning, clustering and Huffman coding. Deep K-means [7] uses regularization terms to encourage weights to concentrate at re-training time and proposes interesting data re-shaping techniques optimized for higher data reuse with row-stationary dataflow [8] at inference time. It is also possible to reduce complexity by encoding both weights and features map, then compute multiplications using a lookup table. Both LookNN [9] and Quantized CNNs [10] are based on this concept and achieve good results at inference time, particularly if energy efficiency is taken into account.

While the direct training approach shows excellent results in the literature [11], it is often inconvenient due to the necessity to adapt the training framework and methods. Furthermore, re-training the model requires access to the full training data and framework, which is not always possible due to the potential size of the entire dataset, and also for legal and privacy compliance reasons. This is where conversion algorithms offer an alternative by working directly on pre-trained models. An instance of this approach can be found in vector quantization [12], where a K-means clustering of fully connected (FC) layers is proposed with a systematic approach.

Despite the promising results, all existing approaches suffer from important limitations: requiring retraining or the use of several steps, or not covering the whole network. Our goal is to compress a neural network targeting embedded devices. We choose to address this problem using a systematic exploration to convert a pre-trained model, without requiring retraining (which can be costly or even impossible) and targeting both FC and CONV layers to leverage redundancies where they exist.

The rest of the paper is organized as follows: Section II describes the proposed method for design space exploration, while Section III presents the preliminary results obtained. Finally, Section IV concludes the paper.
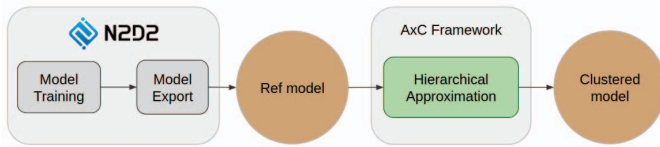
Fig. 1. Overall flow of the proposed method

II. PROPOSED METHOD

Fig. 1 sketches the overall flow of the proposed method. First of all, we start from a trained network (i.e., the reference model in the figure). For the training phase, we use the open-source framework N2D2 [13] (however, any available framework can be used). It is important to notice that we export the trained network in an already quantized form, rather than with the usual full precision (i.e., with weights represented by 32-bit floating-point data). This is due to the fact that we want to further compress the network after the quantization step. Our AxC framework applies a so-called "hierarchical approximation" to the reference network. Within the framework, all the layers of the network are processed. The result is an approximate network characterized by its accuracy loss and its compression rate as compared to the reference. This section firstly presents a well-known and suitable weight sharing approximation technique, and a new design space exploration method.

A. Weight Sharing Approximation Method

As seen in previous sections, neural network approximation can be achieved using different approaches. The weight sharing technique identifies values that will be shared between weights.

In this way, instead of storing the weight values, only the indexes for accessing them are required and thus stored in the memory. Accordingly, the size of the weight matrix can be reduced from B bits for each value to $log2(K)$ with K being the number of different values. The size of the network then becomes $N * log2(K) + K * B$ instead of $N * B$. Further compression can be achieved using K-means clustering to reduce K.

Fig. 2 shows an example to clarify the benefits of weight sharing. The first matrix corresponds to a 5×5 convolutional kernel whose values were computed during training. The matrix contains $N = 25$ values in the range from 0 to 20, which can be quantized in 5bits, resulting in a total $Size = N * B = 25 * 5 = 125bits$.
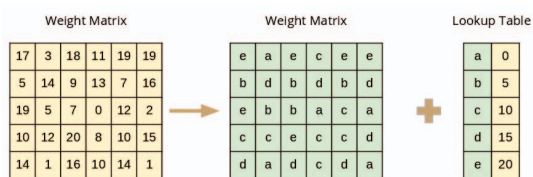

Fig. 2. Network compression using lookup table

In this example we identified 5 clusters, namely 'a', 'b', 'c', 'd' and 'e', replacing the 25 original values as shown in the second matrix. Instead of still storing 25 values, we store only the indexes of the clusters. 5 clusters only require three bits for the encoding. The cluster index is finally used to access a lookup table (LUT, as shown in the figure) to retrieve the weight value. Accordingly, with 5 clusters ($K = 5$), we obtain $WeightMatrixSize = N * log2(K) = 75bits$ and $LUTSize = K * B = 25bits$ resulting in a total $Size = 100bits$. We can thus save 25 bits with respect to the original kernel matrix, achieving a 20% compression ratio.

According to the literature, the K-means based weight sharing method has achieved outstanding performance compared to other methods [7], by using a scalar pool of values instead of data-representation based values. This maintains a very low accuracy loss and benefits from good network compression rates.

The K-means algorithm results are variable and based on random initialization. Some works try to find better initialization methods, such as using linear initialization in order to improve weight range representation [2]. We choose to maintain standard random initialization for the sake of simplicity, but this will be further explored in future work.

B. Hierarchical Exploration of Weight Sharing Opportunities

Weight sharing can be applied at different granularities, ranging from the level of a single convolutional kernel (see Fig. 2), up to the level of the whole network. The simplest approach is to perform clustering on all network weights in a single step, but this technique lacks a clear representation of the data range, which is a crucial element to avoid accuracy losses [1].

In order to identify the most promising granularity, we apply the weight sharing at the following levels:

- *2D-Kernel*: targets a single layer of a 3D convolutional kernel matrix (2D kernel);
- *Channel*: targets the 3D convolutional kernel matrix;
- *Layer*: targets all the 3D convolutional kernel matrices of a network layer.

Each of the above levels of granularity lead to different compression ratios and accuracy loss figures with respect to the reference network. The accuracy is evaluated by checking the top-1 accuracy; hence, an approximate network is considered to have an accuracy loss if the top-1 classification accuracy differs from the reference. Fig. 3 plots the trade-off obtained at different levels of granularity. Each solution corresponds to a network with a certain number of clusters. The results clearly show that the "kernel" and "channel" granularity do not lead to a good tradeoff between compression and accuracy. On the other hand, the "layer" granularity allows up to 3x higher compression ratios for the same accuracy loss. For the remainder of the paper, we thus exploit the **layer granularity** in our weight sharing technique.

As reported in the previous section, weight sharing allows the memory footprint to be compressed because we reduce the number of stored values. However, we have to also include a
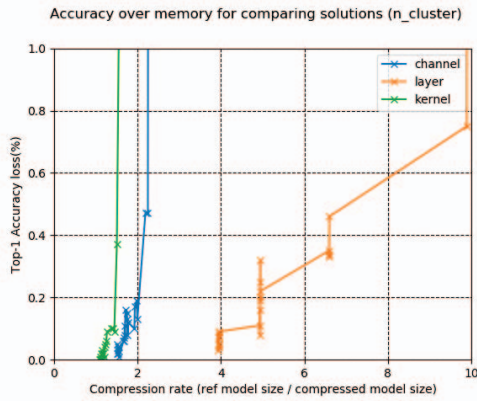
Fig. 3. Top-1 accuracy loss and memory compression over varying numbers of clusters at different clustering levels

Look-up Table (LUT) in order to associate indexes to weight values. The designer has thus two possibilities:

1) Resort to a LUT for each layer;
2) Resort to a LUT for the whole network.

Once the level of granularity has been identified, we explore the impact of different clusters among the layers. We thus develop a simple exploration framework to identify the best number of clusters per each layer.

Fig. 4 gives the basic concept of the proposed exploration framework, where we target one layer at a time. For the current layer, we apply the K-means algorithm several times by varying the number of clusters from 1 to N. Each solution is characterized by its accuracy loss.

For each solution, we thus need to run the approximate network on the test set to quantify the accuracy loss. In our case, the database is the MNIST and we run the network on the 10k test database. We use a greedy approach aimed at minimizing the accuracy loss.

As the number of distinct combinations of suitable high-level approximation parameters for a network is very large, it is probable that the first solution to be found is not optimal. For this reason, we use several iterations to improve the exploration of the solution space by attempting to escape from potential local optimization extrema. At each iteration, the combination found in the previous iteration is used as a base, and the hierarchical exploration algorithm is executed. This
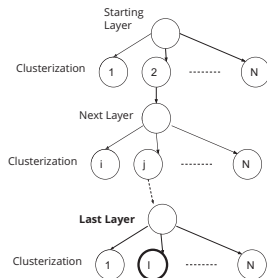
improves the high-level approximation parameter values by adapting them to each other, and the initialization is a non-approximated network. We do stop the process after a user-defined maximum number of iterations.

For each identified solution (i.e., one per iteration) we compute the compression ratio with respect to the reference network.

III. EXPERIMENTS

This section presents the preliminary results obtained by the proposed approach. The target CNN is LeNet-5 [14], composed of 3 convolutional layers (CONV) followed by 2 fully connected (FC) layers, with a total of 61,470 parameters (of which 50% are in the convolutional layers). One difference with our own LeNet is the removal of the last SoftMax layer in order to bind the last FC layer to the classification output. We trained on the MNIST handwritten digit dataset using 28x28 cropped pictures. The training set contained 48,000 images, with an additional 12,000 for the validation set, and 10,000 for the testing set. The learning rate started at 0.05, with the decay of 5.10^{-4} every $375(*128)$ iterations, and momentum was set to 0.9.

The training was carried out using the open-source framework N2D2 [13]. The LeNet-5 model description we used is available in the framework itself. It is important to mention once again that the proposed approach is independent of the adopted training tool. We used the top-1 classification accuracy as an evaluation metric: at the end of the training, we reached a 0.89% error rate. We vary the number of clusters of the K-means algorithm from 1 to $N = 25$, where the maximum value corresponds to the number of values in a LeNet-5 kernel and thus, the minimum number of weights in a layer. The exploration algorithm was set to iterate using previous values as initialization until it found an already identified solution, with a limit of 30 iterations. We explored in two different modes: natural order, from the first layer to the last; and reversed order. Execution of the optimization algorithm took 7.2 hours on an Intel I7 (8 core CPU).

Table I shows the obtained results. The first column specifies the type of network. For each result, we compute the accuracy loss and the compression ratio (CR) as reported in the second column. The latter is calculated by dividing the size of the int-16 trained model by the size of the approximate network, and the accuracy loss corresponds to the degradation induced

Fig. 4. Layer by layer hierarchical design space exploration graph

TABLE I
COMPRESSING LeNet-5 ON MNIST

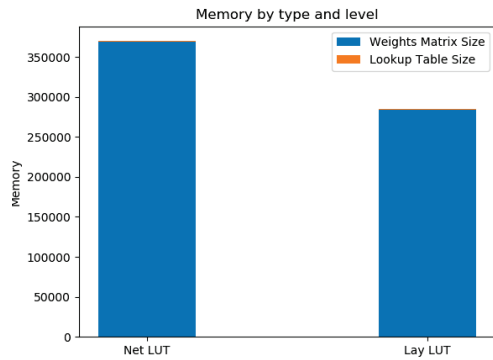| Id	Type	Network	Top-1 accuracy loss (%)	CR
1	N2D2 export	16-bit (ref)	0.00	1
2	N2D2 export	8-bit	0.05	2
3	AxC	16-bit first to last	0.02	4.06
4	AxC	16-bit last to first	0	4.04
5	AxC	8-bit first to last	0.05	4.37
6	AxC	8-bit last to first	0.05	4.83

Fig. 5. Comparison of different LUT scopes

by the compression. The first two rows correspond to our reference networks. The first is the 16-bit quantized network used as reference (i.e., accuracy loss = 0 and compression ration = 1), while the second is the same network quantized using 8-bits. For this case, we only obtained a 2x compression at the cost of a slight accuracy loss. with respect to the 16-bit network.

The last four rows correspond to the network approximated using the proposed approach. AxC networks 3 and 4 are obtained starting with the 16-bit network. Using our exploration tool, we can achieve a compression rate over 4x from the 16-bit version, without any accuracy loss on the top-1 classification accuracy performance. The difference between 3 and 4 is the order of analysis of the layers. Indeed, we investigated the impact of layer ordering during the approximation (i.e., from the first to the last and *vice versa*). We obtained slightly better results when we perform clustering from the last layer to the first layer, but we observe in each case that the resulting approximation network is more approximated in the first layers, which correspond to feature extractors and are less sensitive to approximation degradation. AxC networks 5 and 6 are obtained by starting from an int-8 network, allowing the clustering to be more efficient as the range of values is smaller than the int-16 counterpart. This results in a slightly higher compression rate, but the accuracy loss is higher. In this case, the approximate layer order does not impact the accuracy loss.

Finally, we also compare the memory footprint required by using a single LUT for the whole network versus the use of one LUT for each layer. Fig. 5 represents the different memory requirements, showing that it is more advantageous to use one LUT per layer. Moreover, we can see that the LUT size is negligible with respect to the size of the weight matrix. We cannot draw a generic conclusion based on a single CNN, but this can nevertheless be considered as an promising way to evaluate the impact of given design parameters.

IV. Conclusion

This paper presented a smart design space exploration method for compressing trained deep neural networks. We have proven that our method is able to compress a deep neural network with negligible accuracy loss. The most important advantage is the fact that our approach is a one-shot conversion, and thus, we are able to avoid the prohibitive cost and constraints tied to network re-training.

Future work will first target the application of the proposed approach to larger networks. Then, we intend to extend the framework to support other types of approximate computing techniques like pruning, as the use of a combination of approximation methods allows more levels of redundancy inherent to neural networks to be leveraged. We also plan to evolve our greedy algorithm to include an optimization function taking compression and other hardware implementation dedicated metrics into account. For the approximation method itself, further parameters will be added such as initialization method for K-mean allowing the design space to be enlarged and to explore potentially better parameter combinations.

V. Acknowledgement

References

[1] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, pp. 2295–2329, Dec 2017.

[2] W. J. D. Song Han, Huizi Mao, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv*, 2016.

[3] C. Baskin, E. Schwartz, E. Zheltonozhskii, N. Liss, R. Giryes, A. M. Bronstein, and A. Mendelson, "UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks," *arXiv*, Apr. 2018.

[4] A. Acharya, R. Goel, A. Metallinou, and I. Dhillon, "Online Embedding Compression for Text Classification using Low Rank Matrix Factorization," *arXiv*, Nov. 2018.

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv*, Mar. 2015.

[6] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *CoRR*, vol. abs/1504.04788, 2015.

[7] J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, and Y. Lin, "Deep k-Means: Re-Training and Parameter Sharing with Harder Cluster Assignments for Compressing Deep Convolutions," *arXiv*, June 2018.

[8] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–379, June 2016.

[9] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "LookNN: Neural Network with No Multiplication," in *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 1779–1784, 2017.

[10] Y. W. Q. H. Jiaxiang Wu, Cong Leng and J. Cheng, "Quantized convolutional neural networks for mobile devices," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[12] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," *arXiv*, Dec. 2014.

[13] CEA-LIST, "N2D2." https://github.com/CEA-LIST/N2D2. [Accessed: Dec-2019].

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.