Reproducing the sparse Huffman Address Map compression for deep neural networks

Giosuè Cataldo Marinò$^{1[0000-0003-1386-6770]}$,
Gregorio Ghidoli$^{1[0000-0002-0421-8512]}$, Marco Frasca$^{1[0000-0002-4170-0922]}$, and
Dario Malchiodi$^{1[0000-0002-7574-697X]}$

[1] Dipartimento di Informatica, Università degli Studi di Milano
Via Celoria 18, 20133 Milan, Italy
{marco.frasca,dario.malchiodi}@unimi.it, {giosue.marino,
gregorio.ghidoli}@studenti.unimi.it

Abstract. Deploying large convolutional neural networks (CNNs) on limited-resource devices is still an open challenge in the big data era. To deal with this challenge, a synergistic composition of network compression algorithms and compact storage of the compressed network has been recently presented, substantially preserving model accuracy. The proposed implementation, which we describe in this paper, offers different compression schemes (pruning, two types of weight quantization, and their combinations) and two compact representations: the Huffman Address Map compression (*HAM*), and its sparse version *sHAM*. Taken as input a model, trained for a given classification or regression problem (as well as the dataset employed, which is necessary for the fine-tuning of weights after network compression), the procedure returns the corresponding compressed model. Our publicly available implementation provides the source code, two pre-trained CNN models (retrieved from third-party repositories referring to well-established literature), and four datasets. This implementation includes detailed instructions to execute the scripts and reproduce the obtained results, in terms of the figures and tables included in the original paper.

Keywords: CNN compression · Weight pruning · Weight sharing · Probabilistic quantization · Entropy coding.

1 Introduction

This paper focuses on the reproducibility of results obtained in [5]. The aim of the original work is twofold: a) to evaluate the impact of lossy CNN compression techniques (pruning and quantization) on prediction accuracy, and b) to provide a compressed and compact representation of a given trained CNN for classification or regression problems. Step a) has been carried out by considering two publicly available CNNs (see Sect. 2.2) trained respectively for image classification and for protein-ligand affinity prediction (regression). The weights of these models have been pruned and/or quantized, considering in particular

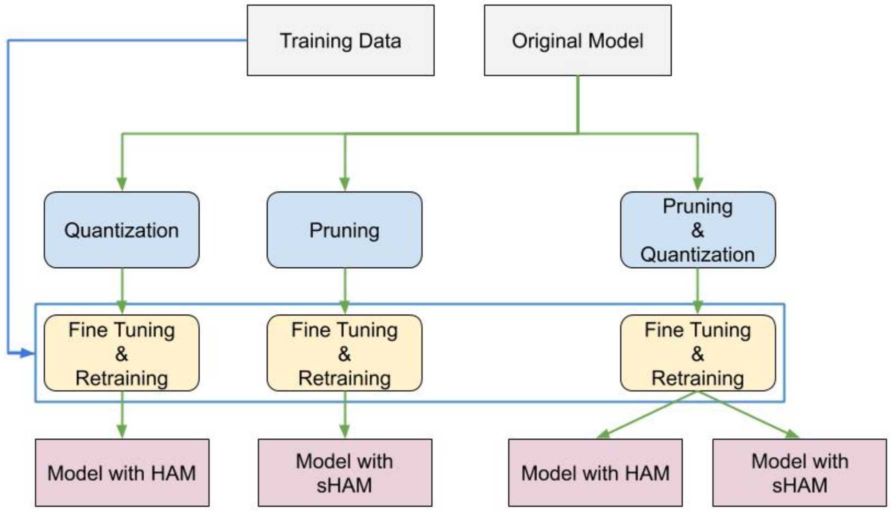

Fig. 1: Sketch of the proposed compression framework. The last level reports the best representation format for the corresponding weight compression strategy. For pruning and quantization both HAM and sHAM are shown, meaning that the format achieving the best compression rate depends on the proportion of original connection pruned (with low pruning HAM is preferred).

two different quantization procedures, namely weight sharing and probabilistic quantization (cfr. Sect. 2.3). The prediction performance of the compressed models has been assessed on four benchmark data sets: MNIST [4], CIFAR-10 [3], DAVIS [2] and KIBA [7] (see Sect. 2.1). Step b) leverages two novel compression formats specifically designed to benefit from the pruning and quantization of the connection weights, called HAM and sHAM, described in Sect. 2.4. Finally, Sect. 3 describes how to run the experiments discussed in the original paper, depicted in Fig. 1 and consisting of:

- input data retrieval (pre-trained CNN, as well as the corresponding training set);
- network pruning and/or quantization;
- model retraining;
- model transformation to HAM or sHAM formats;
- assessment of the compressed model performance.

2 Implementation

In this section we describe all stages of the processing pipeline underlying the results in [5]. Namely, Sect. 2.1 briefly introduces the processed datasets, while

	Size	Resolution
MNIST	70 k	28×28, grayscale
CIFAR	60 k	32×32, color

(a) Classification

	Proteins	Ligands	Interactions
DAVIS	442	68	30056
KIBA	229	2111	118254

(b) Regression

Table 1: Structure of the processed datasets.

Sect. 2.2 describes the models of neural networks used to test the performances of three compression schemes. The latter are detailed in Sect. 2.3, while Sect. 2.4 depicts the representation format of the compressed models.

2.1 Dataset

We validated our methodology on two problems, respectively in the classification and regression realms, employing in both cases two distinct datasets (see also Table 1).

Classification. The first application concerns the multiclass classification of handwritten digits, carried out on the MNIST [4] and CIFAR-10 benchmarks [3]. MNIST contains 70 k 28×28 grayscale images, whereas CIFAR-10 consists of 60 k 32×32 color images.

Regression. We considered the problem of predicting the affinity between drug (ligand) and targets (proteins), processing the DAVIS [2] and KIBA [7] datasets. Proteins and ligands are both represented through strings, respectively using the aminoacid sequence and the SMILES (Simplified Molecular Input Line Entry System) representation. DAVIS and KIBA contain, respectively: i) 442 and 229 proteins, ii) 68 and 2111 ligands, and iii) 30056 and 118254 total interactions.

2.2 State-of-the-art models

We considered two state-of-the-art CNN models as part of the input of our processing pipeline:

- *VGG19* [6], containing 16 convolutional layers and a fully-connected block, trained on the CIFAR-10 and MNIST datasets; we assumed that this model is likely over-dimensioned for the digit classification task, and indeed we showed that we can obtain a succint version (requiring significantly less than 1% of the original space) without accuracy loss;
- *DeepDTA* [8], composed of two convolutional blocks (three convolutional layers followed by a MaxPool layer) to process proteins and ligands separately, which are then joined through three fully-connected hidden layers; the network is trained in turn on the DAVIS and KIBA datasets: although the

original network was specifically tailored for the considered problem, also in this case we obtained a remarkable compression rate, still preserving model performance.

2.3 Network compression

In this section we shortly describe the considered compression schemes, namely pruning, weight sharing, and probabilistic quantization.

Pruning. Activation functions process the sum of the neuron inputs, each weighted according to its connection; a straightforward compression technique consists therefore in "cutting" all connections whose weight has a small absolute value. Indeed, nullifying such negligible weights should not sensibly change the above mentioned signal, as well as the global network output. We parameterized this technique on the threshold p used in order to deem a connection as negligible: in turn, this threshold was defined by considering a suitable set of empirical quantiles of connection weights. As a post-processing phase, we retrained the network, now ignoring the erased connections (that is, clamping the corresponding weights to zero).

Weight sharing. In analogy with the observations at the basis of network pruning, when several weights are close one another, they can all be set to a common value without significantly affecting network performance. We clustered all learnt weights using the k-means algorithm, obtaining k representative centroids which we used to replace the weight values. Also in this case, we subsequently retrained the network, now updating centroids through cumulative gradient. Note that this algorithm, in its original form, outputs a table of representative weights, as well as a matrix storing indices to this table, rather than the weights themselves. We substituted this format with those described in the next section.

Probabilistic quantization. An alternative approach to weight sharing is that of selecting the representative weights through a probabilistic algorithm. We adapted a technique used in the realm of bandwitdh reduction, having the desirable property that the above metioned representative values can be thought as an unbiased estimate of the original weights. For this technique we used the same retraining process and parameterization used for the weight sharing technique.

2.4 Compact network representation

In order to appropriately exploit the characteristics of the compressed connection matrices, the latter are stored by means of two novel formats, respectively called HAM and sHAM. Both formats represent each matrix element through Huffman coding, subsequently concatenating the corresponding codewords by column order, thus obtaining a unique binary string. HAM encodes all weights, and the lower the number of distinct weights in the matrix, the lower the average codeword length. To benefit also from the matrix sparsity, sHAM applies Huffman

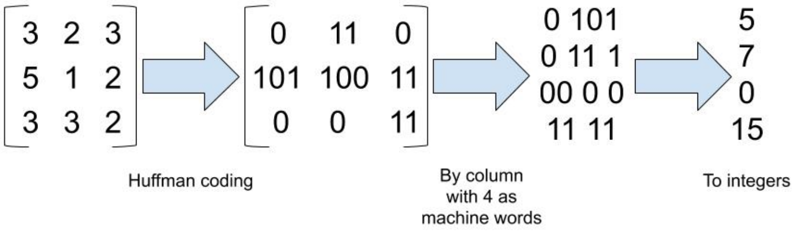

Fig. 2: Example of HAM storage format.

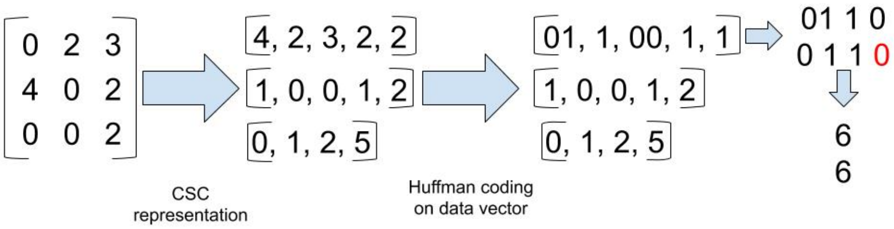

Fig. 3: Example of sHAM storage format. The additional 0 marked in red is the effect of padding required in order to work with machine words.

coding only to non zero elements, stored through Compressed Sparse Column (CSC) format. In both HAM and sHAM cases, the generated bit sequence is organized as a succession of machine words, interpreted as an array of integers. Figures 2 and 3 show an example of encoding for both formats, highlighting the various phases of the involved conversion.

3 Reproducibility

The results illustrated in [5] can be fully replicated by running the code in the repository freely available on GitHub[1]. Once this repository has been cloned, the user can launch the `runner.sh` shell script (available in the root directory), which automatically creates a virtual environment within which all required libraries are installed, to subsequently run all the experiments. As a result, several text files are created in the `time_space/results/` directory. These files can be postprocessed by the notebook `time_space/plot_from_file.ipynb` to get the best compression results and to generate summary plots. It is worthy pointing out that, however, some small fluctuations in the results are inherently due to the GPU utilization [1]. Running the experiments requires the availability of at least 10 GB of RAM, in order to load the selected CNN models; the execution time took roughly two weeks using a computing environment equipped with an Nvidia RTX 2060 GPU and an i7-9750H CPU. Note,

[1] https://github.com/giosumarin/ICPR2020_sHAM

however, that this time refers to the execution more than 860 different experiments, each involving compression and retraining of a CNN. To reproduce a single compression experiment and to quickly check its reproducibility, the reader can refer to the script `runner_single_exp.sh` in the repository root. Whereas, the implementation of specific compression schemes is available in the `compressionNN_package/compressionNN` folder of the above mentioned repository. More precisely, the scripts in files `pruning.py`, `weightsharing.py`, and `stochastic.py`, respectively implement pruning, weight sharing, and probabilistic quantization; analogously, the considered joint techniques are implemented in `pruning_weightsharing.py` and `pruning_stochastic.py` scripts.

Acknowledgements

This work has been supported by the Italian MUR PRIN project "Multicriteria data structures and algorithms: from compressed to learned indexes, and beyond" (Prot. 2017WR7SHH).

References

1. Cook, S.: CUDA Programming: A Developer's Guide to Parallel Computing with GPUs. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2012)
2. Davis, M.I., Hunt, J.P., Herrgard, S., Ciceri, P., Wodicka, L.M., Pallares, G.J., Hocker, M., Treiber, D.K., Zarrinkar, P.P.: Comprehensive analysis of kinase inhibitor selectivity. Nature Biotechnology **29**, 1046–1051 (2011)
3. Krizhevsky, A.: Learning multiple layers of features from tiny images. Master's thesis, University of Toronto (2009)
4. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
5. Marinò, G., Ghidoli, G., Frasca, M., Dario, M.: Compression strategies and space-conscious representations for deep neural networks. In: International Conference on Pattern Recognition, ICPR 2020 (2020), in press (arxiv:2007.07967).
6. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
7. Tang, J., Szwajda, A., Shakyawar, S., Xu, T., Hintsanen, P., Wennerberg, K., Aittokallio, T.: Making sense of large-scale kinase inhibitor bioactivity data sets: A comparative and integrative analysis. Journal of Chemical Information and Modeling **54**(3), 735–743 (2014). https://doi.org/10.1021/ci400709d
8. Öztürk, H., Özgür, A., Ozkirimli, E.: DeepDTA: deep drug–target binding affinity prediction. Bioinformatics **34**(17), i821–i829 (09 2018). https://doi.org/10.1093/bioinformatics/bty593