Neurocomputing 520 (2023) 152–170

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Deep neural networks compression: A comparative survey and choice recommendations

Giosué Cataldo Marinó, Alessandro Petrini, Dario Malchiodi, Marco Frasca *

Dipartimento di Informatica, Università degli Studi di Milano, Via Celoria 18, 20133 Milano, Italy

ARTICLE INFO

Article history:
Received 22 March 2022
Revised 6 September 2022
Accepted 21 November 2022
Available online 25 November 2022
Communicated by Zidong Wang

Keywords:
CNN compression
Connection pruning
Weight quantization
Weight sharing
Huffman coding
Succinct Deep Neural Networks

ABSTRACT

The state-of-the-art performance for several real-world problems is currently reached by deep and, in particular, convolutional neural networks (CNN). Such learning models exploit recent results in the field of deep learning, leading to highly performing, yet very large neural networks with typically millions to billions of parameters. As a result, such models are often redundant and excessively oversized, with a detrimental effect on the environment in terms of unnecessary energy consumption and a limitation to their deployment on low-resource devices. The necessity for compression techniques able to reduce the number of model parameters and their resource demand is thereby increasingly felt by the research community. In this paper we propose the first extensive comparison, to the best of our knowledge, of the main lossy and structure-preserving approaches to compress pre-trained CNNs, applicable in principle to any existing model. Our study is intended to provide a first and preliminary guidance to choose the most suitable compression technique when there is the need to reduce the occupancy of pre-trained models. Both convolutional and fully-connected layers are included in the analysis. Our experiments involved two pre-trained state-of-the-art CNNs (proposed to solve classification or regression problems) and five benchmarks, and gave rise to important insights about the applicability and performance of such techniques w.r.t. the type of layer to be compressed and the category of problem tackled.

1. Introduction

The methodology behind deep neural networks (DNNs) dates back to more than forty years ago. However, the availability of dedicated hardware (such as GPUs or TPUs) and of huge datasets recently allowed to maximize the performance of several DNN-based predictors, setting in practice the state-of-the-art for several problems of image processing, financial forecasting, and so on. Convolutional neural networks (CNNs) played a key role in this advancement, and several pre-trained models, like for instance AlexNet [1] and VGG16 [2], to mention earlier works, or T5, proposed in the context of natural language processing [3], are available as base models for the applications of transfer learning techniques [4,5]. Such models are often overparameterized [6] and have a considerable memory footprint: for instance, the above mentioned VGG16 demands around 500 MB, whereas T5 (variant 11B) requires about 20 GB. Their ever-increasing size has given rise to major challenges, including a critical ascent in energy consumption and its detrimental effect on the environment [7], the need of trustworthy artificial intelligence systems, i.e., systems functioning in the most environmentally friendly way possible during both development and deployment (ALTAI guideline 6) [8], and the limitation or even the unfeasibility of their on-device training and inference on low-resource devices, e.g., an edge device with limited computational resources and battery life [9].

As a matter of fact, the need of *space-conscious* models is consequently emerging in multiple machine learning applications [10]. Plenty of approaches have been proposed in the last decade to tackle this problem, ranging from methods which adopt learning strategies directly producing succinct neural networks (see, e.g., [11,12]), to techniques aiming to appropriately compress existing models. Unfortunately, most of the proposed methodologies in this field are application- or model-specific, thereby making rather difficult to identify the approach or the combination of compression methods most suitable for a given problem and a deep neural model specifically designed to solve it. The purpose of this work is to try to identify some consolidated trends that help practitioners in selecting the most appropriate way to compress their pre-trained networks. To this end, we focus on the post-training compression, that is the problem of reducing the computational

* Corresponding author.
E-mail addresses: giosue.marino@studenti.unimi.it (G.C. Marinó), alessandro.petrini@unimi.it (A. Petrini), dario.malchiodi@unimi.it (D. Malchiodi), marco.frasca@unimi.it (M. Frasca).

https://doi.org/10.1016/j.neucom.2022.11.072
0925-2312/

resources needed by existing CNNs while possibly preserving their accuracy. This choice is motivated by the rapid increase of the number of pre-trained CNNs made publicly available in various application contexts, and by the fact that frequently such models are heavily redundant [13]. In doing so, we focus on structure-preserving compression methods, thus excluding all approaches, e.g., *knowledge distillation* [14] or *skeletonization* [15], that change the topology of the network (number of layers, neurons, filters, channels, and so on). Not altering the model structure best suits the case of post-training compression, requires no prior knowledge about the interdependence between the different layers in the network, and does not represent a limitation, as other compression approaches varying the network topology can be applied only in earlier stages. Moreover, there are some specific network structures which cannot easily undergo to layer/filter/channel elimination (for instance, those forming residual blocks).

That being said, the vast domain of structure-preserving CNN compression methodologies can be roughly arranged into three main categories: connection pruning, weight quantization, and (weight) matrix/tensor low-rank decomposition. Specifically, in addition to standard magnitude-based connection pruning we examined four top-performing quantization methods, four variants of them, and a low-rank factorization method recently proposed to compress fully-connected (FC) layers in CNNs, by combining both sparsity and singular value decomposition. The ultimate aim is to extensively compare, in terms of accuracy and compression ratio, the effectiveness of each methodology in relation to the layer type, the problem category (classification or regression), and the data characteristics. To the best of our knowledge, this is the first study investigating several CNN compression techniques not for survey purposes, but rather for detecting their relevant and specific features in relation to the type of problem, data and model to which they are applied, so as to assemble a vademecum for users who need to reduce the computational demand of their deep neural systems. Accordingly, we included in our experimental comparison two publicly available CNNs and five heterogeneous benchmark datasets: respectively, three are related to classification and the remaining two to regression. Although having evidence only in relation to the considered data, problems and neural models, our experiments:

(i) provide important indications about the behaviour of individual compression schemes when applied both to convolutional and FC layers,
(ii) give some important clues for the choice of the compression design in relation to a specific application, and
(iii) can serve as a reference for who is approaching this field for the first time.

In particular, we have initially found that quantizing globally rather than by individual layers typically yields a higher compression rate, and in most cases the same or higher accuracy. This is independent of the layer type and the considered task. On the other side, our experiments suggest that it is not convenient to apply weight quantization to convolutional layers (mainly for classification problems) and that some quantization methods sometimes show unstable behaviour, thus discouraging their application under specific conditions (related to quantization level, problem, and layer type). On FC layers, instead, the application of weight pruning along with quantization is always convenient when the layer topology allows a relatively high pruning level. On such layers, low-rank factorization is a faster and valid alternative to pruning/quantization methods, but only on classification problems. In terms of absolute performance, our results corroborate those obtained in previous studies, namely that it is possible to sensibly reduce the model occupancy, up to around 20× in our case, while not worsening or even improving the baseline performance. Finally, our contribution also includes a public repository containing instructions on how to apply the compression techniques described in the paper also to CNNs and/or datasets not included in this study, thus allowing generic users to approach, design and execute the appropriate compression scheme for their own models.

For sake of readability, we summarize here the notation adopted henceforth. A matrix $\boldsymbol{W}^o \in \mathbb{R}^{n\times m}$ is used to denote the connection weights of a generic network layer (flattened in case of tensors) with n input and m output neurons, whereas its compressed version, resulting from the application of a given compression scheme, is denoted by $\boldsymbol{W} \in \mathbb{R}^{n\times m}$. Symbols w^o and w will denote generic entries of $\boldsymbol{W}^o$ and $\boldsymbol{W}$, respectively. The *occupancy ratio* of $\boldsymbol{W}$ is defined as $\psi = \frac{\text{size}(\boldsymbol{W})}{\text{size}(\boldsymbol{W}^o)}$, where size$(x)$ denotes the memory size of x. A value $\psi < 1$ means that the compressed matrix occupies less than the original one. For instance, $\psi = 0.1$ means that the compressed matrix needs just 10% of the storage space of its uncompressed counterpart. When it will be more appropriate for the discussion, we will refer to the *compression ratio*, that is the inverse of ψ. Boldface and italic boldface will be used for matrices and vectors (e.g., $\boldsymbol{W}$ and $\boldsymbol{x}$), while $|\cdot|$ will be an abstract cardinality operator returning the length of a string or the number of elements in a vector. The log function refers throughout the manuscript to the binary logarithm, while for *sparsity coefficient* of a matrix we mean the ratio of the number of its null elements over the total number of entries. Finally, note that in the general context of neural networks compression the terms DNN and CNN are often used interchangeably. Henceforth we almost always use the latter to highlight the fact that our study is extended to convolutional models.

The paper is organized as follows: Section 2 is dedicated to the related works, while Section 3 describes the compared compression techniques. Section 4 illustrates the experimental comparison and the obtained results. Some concluding remarks end the paper.

2. Related work

The main studies belonging to the considered categories of structure-preserving CNN compression methods, namely connection pruning, weight quantization, low-rank matrix and tensor decomposition, are separately described hereafter, including techniques based on weight sharing in the broader category of weight quantization. For the sake of completeness, the main structural compression methods are also sketched in Section 2.4. The literature abounds with thorough reviews of compression methods for NNs: the interested reader can refer for instance to [16,17].

2.1. Connection pruning

Connection pruning is likely the most common technique to lower the number of parameters in a pre-trained network (and likely also the oldest one, as it dates back to 1990, see for instance [18]). Pruning consists in eliminating connections deemed as irrelevant w.r.t. the overall network behavior, while clamping the weight matrix dimension (differently than in structural compression, see Section 2.4). Libraries handling sparse matrix multiplication (SM) can be subsequently exploited to take full advantage of the memory reduction. However, SM tends to be slower than its dense counterpart; as a consequence, structural compression is often applied along with connection pruning. The simplest pruning strategy involves setting a threshold that decides which connections are removed: typically, those whose weights are in absolute value lower than the threshold are pruned [19]. The threshold can be layer-specific, or it can be set globally for the whole

network. This criterion is often referred to as *magnitude-based* pruning, and it involves a subsequent fine-tuning (retraining) of the remaining weights. In another large class of pruning approaches, regularization terms (e.g., using L_1 or L_2 norms) are applied to drive the learning algorithm to output a network in which several weights have negligible values, so that pruning can be applied as a straightforward post-training step [20]. Connection pruning has also been performed via alternative optimization strategies, such as genetic algorithms [21] and particle swarm optimization [22]. The benefit of these techniques is that weights removed at first can be reintroduced later on; notwithstanding, their high complexity allows their application only to CNNs of limited size.

2.2. Weight quantization

In neural networks, *quantization* refers to the process of fixing the number of bits used in order to represent weights, activation or gradient values, with the effect of saving space. For instance, when using single-precision floating point (FP32, as each value is encoded using 32 bits) in place of standard double-precision (FP64), the space used to store connection weights is halved. Half-precision for floating point (FP16) and integer arithmetic (INT16) are also commonly considered. To reduce memory footprint, meanwhile speeding up training time, recent researches have investigated the use of even smaller precision for the training and inference phases of CNNs, considering short integer (INT8, INT4, INT2), or just 1-bit representation [23]. In the latter, extreme form of quantization also called *binarization*, weights and activations are represented using a single bit and floating points operations are converted into binary gates [24]. Usually, the use of less than 32 bits can yield to a severe decay in performance, and a standard approach in this case is to replace floating-point operations with their integer equivalent, sometimes adding a further training step that ameliorates the consequent accuracy compromise [25]. Hybrid approaches have also been developed to apply quantization only to the weight and activation values concentrated around a narrow region, whilst representing the remaining values using a higher precision, thus leading to reduced errors [26].

Another category of compression methods considers the quantization as minimization of a penalty term included in the loss function [27,28]. Due to its independence on the underlying architecture and their low complexity, *weight sharing* quantization has found a large application: here, the weights are first partitioned into multiple categories, then within each category a representative value is selected and used to replace all weights in that category. Such methods mainly differ in the way they subdivide the network weights, e.g., by means of clustering techniques [29], statistical methods [30,31], uniform schemes [32], or by minimizing the distortion and the entropy of the coded source [33]. We will describe these methods in detail in Section 3.

2.3. Low-rank matrix and tensor decomposition

CNNs can also be compressed by decomposing the weight matrices (or tensors reshaped to 2 dimensions) into a lower rank approximation. For instance, a given matrix $\boldsymbol{W} \in \mathbb{R}^{n\times m}$ of full rank r can be decomposed as $\boldsymbol{W} \simeq \boldsymbol{AH}$, where $\boldsymbol{A} \in \mathbb{R}^{n\times r}$ and $\boldsymbol{H} \in \mathbb{R}^{r\times m}$, moving from a space complexity $\mathcal{O}(nm)$ to $\mathcal{O}(r(n+m))$, to the detriment of some approximation error coming from the estimate of low-rank matrices. Singular Value Decomposition (SVD) has been widely used in this context to achieve such a factorization [34,35]: in this case $\boldsymbol{W} = \boldsymbol{U\Sigma V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{n\times r}$ and $\boldsymbol{V} \in \mathbb{R}^{m\times r}$ are orthogonal matrices, and $\boldsymbol{\Sigma} \in \mathbb{R}^{r\times r}$ is the diagonal matrix of singular values. The nonzero elements of $\boldsymbol{\Sigma}$ are sorted in decreasing order (along with the rows of $\boldsymbol{U}$ and $\boldsymbol{V}$). The truncated SVD approximates $\boldsymbol{W}$ by computing the product of the matrices obtained by selecting the first $q < r$ columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ and the top left square submatrix of $\boldsymbol{\Sigma}$ of dimension q. Similar approaches apply Canonical Polyadic and Tucker decompositions to tensors [36], convolutional filters decomposition into a linear combination of rank-1 (separable) filters [37] and channel-wise low-rank decompositions endeavoring to reduce redundancy in the channel [38], or a combination of both approaches [39].

2.4. Other approaches

Other relevant compression techniques include knowledge distillation (KD) and several approaches to structural pruning. The former encompasses the learning of a 'thinner' NN model, called *student*, from a larger *teacher*, whose outputs act as soft targets for the training process. The teacher output should be a probability distribution, and the idea is to exploit the corresponding logits to 'distill' information to the student, which is trained minimizing the cross entropy between the logits of teacher and student [14,40,41].

In structural compression instead, inadequate components (e.g., units or layers) are pruned, usually through iterative procedures [42,43]. Adequacy is related to the loss change incurred when a component is removed [44]. *Skeletonization*, for instance, consists in coupling each connection to an importance coefficient. The more general class of *loss sensitivity* methods computes importance coefficients via measures of loss variation using first [45] or second order derivatives [43], and after a joint training of connections and coefficients, all units having the lowest importance coefficients are discarded. Both FC [46,47] and convolutional [48–50] parts can be pruned, however there are no general criteria to detect useless components, and often one should just rely on problem-dependent heuristics. Finally, of undoubted interest are some attempts to simultaneously learn a model along with its compression via constrained optimization frameworks, which propose alternated training and compression steps via the existing compression schemes mentioned so far [51], or their combinations [52].

3. Materials and methods

In the following sections we describe the compression strategies considered in this comparative study.

3.1. Connection pruning

We implemented connection pruning (see Section 2.1) removing weights that are small in absolute value. After having fixed an empirical percentile w_p of the entries of $\boldsymbol{W}^o$, we defined $\boldsymbol{W}$ by setting $w = w^o$ if $|w| > w_p, 0$ otherwise. The time complexity is $\mathcal{O}(nm\log(nm))$, as the sorting step needed for obtaining w_p dominates the overall computation. The procedure ends with a fine-tuning for non-null weights in $\boldsymbol{W}$. The only hyperparameter is the percentile level p, related in turn to the sparsity coefficient (see Section 4 for a description of how hyperparameters of all considered compression methods have been selected).

3.2. Quantization via weight sharing

This quantization strategy consists in reducing the space needed to store individual weights via weight sharing (WS), expressly by casting connection weights into categories and substituting all weights in a category with a representative for them. Three logic phases can be detected: a) *detecting shared weights*, by partitioning the latter into k categories, and transforming all

the weights in i-th category into a unique representative value c_i; b) *cumulative retraining of the weights*; c) *storage of the shared weights*, adopting a suitable format leveraging the quantization. The various state-of-the-art quantization approaches considered in this work mainly differ for the way they realize phase a), as detailed in the following.

3.2.1. Share weights

Clustering-based WS (CWS) This strategy, fixed the number k of clusters $C_1, \ldots, C_k$, aims at gathering similar values in $\boldsymbol{W}^o$ via the k-means algorithm, obtaining the corresponding centroids $\{c_1, \ldots, c_k\}$, and subsequently using the latter as representatives which will replace each weight in $\boldsymbol{W}^o$ [29]. The time complexity is $\mathcal{O}(k(mn)^2)$, due to k-means application. To achieve a higher compression, in [29] pruning and CWS have been applied in chain, with weight sharing considering non-null weights identified by pruning.

Probabilistic WS (PWS) This approach is based on a weight sharing technique named *Probabilistic Quantization*, recently proposed in [30] and relying on a probabilistic transformation analogous to those mapping weights onto special binary or ternary values proposed in [53,54]. Given $w_{\min} = \min \boldsymbol{W}^o$, $w_{\max} = \max \boldsymbol{W}^o$, PWS is based on the following probabilistic rationale[1]: suppose that each weight w^o is the specification of a random variable W^o with support $\mathscr{W} := [w_{\min}, w_{\max}]$. If W denotes the two-valued random variable defined by

$$\begin{aligned} \mathrm{P}(W = w_{\min}) &= \tfrac{w_{\max} - w}{w_{\max} - w_{\min}}, \\ \mathrm{P}(W = w_{\max}) &= \tfrac{w - w_{\min}}{w_{\max} - w_{\min}}, \end{aligned} \tag{1}$$

the specifications of W approximate a weight w through an extreme form of weight sharing in which $k = 2$ and the two representatives $w_{\min}$ and $w_{\max}$ are chosen randomly according to (1). Denoting as usual with $\mathbb{E}$ the expectation operator, it is easy to show that $\mathbb{E}(W|W^o = w) = w$, so that independently of the distribution of W^o,

$$\begin{aligned} \mathbb{E}(W) &= \int_{\mathscr{W}} \mathbb{E}(W|W^o = w) f_{W^o}(w)\mathrm{d}w = \int_{\mathscr{W}} w f_{W^o}(w)\mathrm{d}w \\ &= \mathbb{E}(W^o). \end{aligned} \tag{2}$$

As a consequence, pseudorandomly extracting a specification of W for each entry w^o, we get a quantized matrix $\boldsymbol{W}$ which is a highly compressible *unbiased* estimator of $\boldsymbol{W}^o$. This method is extensible to $k > 2$ as follows: partition $\mathscr{W}$ in $k-1$ intervals $\mathscr{W}_i = [c_i, c_{i+1})$, for $i = 1, \ldots, k-1$ (where obviously $c_1 < c_2 < \ldots < c_k$), so that $\{c_1, \ldots, c_k\}$ will be the set of representative weight values. A generic w is obtained exactly as above, now considering the interval $\mathscr{W}_i$ containing w and replacing $w_{\min}$ and $w_{\max}$ with c_i and c_{i+1}, respectively. It is easy to show that the matricial estimator of $\boldsymbol{W}^o$ will be unbiased regardless of the chosen partition. Here we follow [30] to set c_i as $c_i = \chi_{\frac{i}{k}}$, with χ_q the q-quantile of the entries of $\boldsymbol{W}^o$, which induces the representatives to be scattered evenly over the support in case the elements of $\boldsymbol{W}^o$ are uniformly distributed over $\mathscr{W}$. The overall time complexity amount to $\mathcal{O}(nm\log(nm))$, due to quantile computation.

Uniform Quantization (UQ). This quantization scheme, which selects representative weights uniformly in the weight domain, has been proven yielding an entropy asymptotically smaller than that of any other quantizer, regardless of the source statistics, under the assumption that the latter has a reasonably smoothed density function [55]. Banking upon this result, in [32] UQ has been formalized so as to transform the weight w^o so that $w = \delta \cdot \mathrm{round}((w^o + d)/\delta)) - d$, where $\delta > 0$ is the interval size, $d \in [-\frac{\delta}{2}, \frac{\delta}{2}]$ is a constant bias, and round is the rounding function. The parameter δ must be selected on the basis of compression ratio and/or of accuracy requirements, also taking into account the desired number k of distinct weights: the compression ratio increases, accuracy degrades and k decreases, as δ grows. *Entropy Constrained Scalar Quantization (ECSQ)* This compression strategy (also known as Entropy Coded Scalar Quantization) [33,56] transforms a weight w^o into the representative c_i associated to the partition of "similar" weights C_i to which the weight w^o belongs to. Both C_i and c_i are learned by jointly optimizing the expected value for the quantization distortion D (using a prefixed distortion measure such as MSE) and the entropy H of the resulting distribution of representative weights. The optimal ECSQ scheme has been found minimizing the distortion while constraining entropy to not exceed a prefixed threshold [55], considering the optimization of the Lagrange cost

$$D + \lambda H = \frac{1}{nm} \sum_{i=1}^{k} \sum_{j \in C_i} \left(\left| w_j^o - c_i \right|^2 - \lambda \log p_i \right), \tag{3}$$

where λ is a Lagrange multiplier, and $p_i = |C_i|/(nm)$. ECSQ and UQ have shown the best trade-off accuracy/compression ratio in a recent state-of-the-art comparison [32]. Here the hyperparameter λ impacts on the number of sets in the final partition, since some of them might collapse during the optimization; accordingly, λ must be tuned to obtain exactly k sets in the partition.

3.2.2. Cumulative retrain of weights

Once weights have been partitioned into categories $C_1, \ldots, C_k$ with representative weights $c_1, \ldots, c_k$, a retrain phase follows, ensuring weights always assume values in the set of representatives. This is achieved by using the cumulative gradient defined by $\frac{\partial \mathscr{L}}{\partial c_l} = \frac{1}{|C_l|} \sum_{w \in C_l} \frac{\partial \mathscr{L}}{\partial w}$ to update c_l, with $l \in \{1, \ldots, k\}$. This might end up in using less than k weights, if two or more representatives converge to a same value during retraining.

3.2.3. Storing the shared weights

Once the matrix/tensor $\boldsymbol{W}$ having only k distinct weights $c_1, \ldots, c_k$ has been obtained, different possibilities are available to store it in a lossless compressed format. For the purpose of an equitable comparison, however, the adopted format is less relevant, provided that it is used fairly for all quantization methods. Among the formats specifically proposed for CNNs, in the experimental section we will leverage those yielding the highest compression ratio (to our knowledge) for the specific case. In particular, for FC quantized layers, matrices are represented via the Huffman Address Map (HAM) [30,31], and via its extension for sparse matrices, *sparse* HAM (sHAM), when the sparsity in $\boldsymbol{W}$ is high enough [30]. Both methods construct the Huffman code of the k source symbols, and use the codewords to compress the matrix in a unique binary string. Due to the dependence on the source statistics, it is not possible the know in advance the exact occupancy ratio of HAM and sHAM, whilst quite loose upper bounds are $\psi_{\texttt{HAM}} \leqslant \frac{1+\log k}{b} + \frac{6k}{nm}$ and $\psi_{\texttt{sHAM}} \leqslant \frac{s(1+\log k)}{b} + \frac{6k+m+1}{nm} + s$, where b is the number of bits used to store one entry of $\boldsymbol{W}^o$, and s is the ratio of non-zero elements in $\boldsymbol{W}$.

Being HAM and sHAM not applicable to convolutional layers, their quantized matrices/tensors are represented through the *Index Map* (IM) [29] format. In case of matrices, IM stores the representative weights in a vector $\boldsymbol{c}$, whose indices populate the index matrix $\boldsymbol{\Pi}$, such that if w_{ij}^o is associated with centroid, say, c_1, then $\pi_{ij} = 1$. Denoted by $\bar{b}$ the number of bits used to store one entry

[1] in this subsection we slightly modify the used notation, respectively denoting random variables with upper case letters (W), specifications with lower case letters (w), and sets of specifications with calligraphic upper case letters ($\mathscr{W}$). When speaking of an estimator, in some cases we refer to a univariate random variable modeling a matrix entry, and in other cases to a random matrix: as the context easily allows to discriminate between the two cases, we prefer to not further extend the notation.

of $\mathbf{\Pi}$, the occupancy ratio is $\psi_{\text{IM}} = \frac{\bar{b}nm+kb}{bnm} = \frac{\bar{b}}{b} + \frac{k}{nm}$. Thus, when $k \leqslant 256$ and $\bar{b} = 8$, assuming FP32 for $\boldsymbol{W}^{\text{o}}$ ($b = 32$) implies $\psi_{\text{IM}} \simeq 1/4$. This approach can be easily extended to tensors of order > 2. Since the same storage format is used for all compression techniques, and our aim is to provide a comparison of the latter, in the rest of the paper we consider the ideal case $\bar{b} = \lceil \log k \rceil$, reminding that on most architectures the minimum value for $\bar{b}$ is 8 (one byte). Finally, for FC layers which underwent only to connection pruning, we use the classical Compressed Sparse Column (CSC) format [57] when the sparsity induced in $\boldsymbol{W}$ is high enough, leaving it uncompressed otherwise.

3.3. Sparse low-rank factorization (SLR)

This is a recently proposed method to compress FC layers in DNNs, which achieved top performance in this category of compression techniques [58]. It exploits low-rank factorization of rank q via truncated SVD of $\boldsymbol{W}$, i.e. $\boldsymbol{W} \simeq \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \boldsymbol{U} \in \mathbb{R}^{n\times q}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{q\times q}$, and $\boldsymbol{V} \in \mathbb{R}^{m\times q}$. The truncated matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are further compressed by assuming some input and output neurons in the layer are less important, utilizing for them a reduced rank $\bar{q} < q$. This means that the corresponding row and column entries in $\boldsymbol{U}$ and $\boldsymbol{V}^T$, respectively, are set to zero (*sparsification*). This sparsification preserves the importance of latent concepts contributing to reconstruction of $\boldsymbol{W}$, unlike 'classical' low-magnitude pruning. The authors proposed three strategies to select irrelevant neurons, however the weighted sum of their absolute connections resulted the best criterion when taking into account both performance and efficiency. It is worth noting that no post-factorization fine-tuning of the weight is needed for this methodology. The reduced rank is computed as $\bar{q} = \lceil c_r q \rceil$, whereas the number of rows and columns to be sparsified is $\bar{n} = \lceil c_s n \rceil$and $\bar{m} = \lceil c_s m \rceil$, respectively. The quantities $c_r, c_s \in [0, 1]$, called *reduction* and *sparsity coefficient*, are hyperparameters of the method. When considering classical low-rank factorization via truncated SVD, the number of parameters needed is nq (for matrix $\boldsymbol{U}$), mq (for matrix $\boldsymbol{V}$) and q (for the q entries of the diagonal of $\boldsymbol{\Sigma}$), for a total of $q(n + m + 1)$ parameters. In SLR, $\bar{n}$ rows and $\bar{m}$ columns are instead represented using a rank $\bar{q}$, needing $\bar{q}(\bar{n} + \bar{m})$ parameters, while the remaining $n - \bar{n}$ rows and $m - \bar{m}$ columns will still have rank q. Thus, diving by the number nm of entries in the original matrix, we get the following occupancy ratio:

$$\psi_{SLR} = \frac{q(n - \bar{n} + m - \bar{m} + 1) + \bar{q}(\bar{n} + \bar{m})}{nm}.$$

4. Experimental analysis

In this section we describe the experimental setting used to empirically compare the illustrated techniques in several scenarios, including five datasets and two uncompressed neural networks, as detailed here below.

4.1. Data

- *Classification*. MNIST [59], a benchmark of handwritten digits, containing a train set of 60 K 28x28 grayscale images and a test set of 10 K analogous images; CIFAR10 [60], a dataset of 50 K (train set) + 10 K (test set) 32x32 color images. Both datasets refer to ten classes (one for each digit) and their labels are balanced; CIFAR100 [61], extension of CIFAR10 to 100 classes, containing 500 training images and 100 testing images per class.
- *Regression*. DAVIS [62] and KIBA [63], datasets containing the evaluation of the affinity between drugs (ligands) and targets (proteins), respectively represented using the amino acid sequence and the SMILES (Simplified Molecular Input Line Entry System) string encoding. DAVIS and KIBA contain, respectively, 442 and 229 proteins, 68 and 2111 ligands, 30056 and 118254 total interactions between them, with 1/6 of the data composing the test set.

4.2. Benchmark models

We used two publicly available pre-trained, top-performing CNN models:

(i) *VGG19* [2], consisting of 16 convolutional layers followed by a FC block, in turn containing two hidden layers of 4096 neurons each, and a softmax output layer.[2], originally trained on CIFAR10 (and fine-tuned in order to work with MNIST and CIFAR100 datasets[3]);
(ii) *DeepDTA* [64], having two separate blocks for proteins and ligands, both containing three convolutional layers followed by a max pool layer and merged in a FC block consisting of three hidden layers, respectively containing 1024, 1024, and 512 units, followed by a single-neuron output layer[4].

Using pre-trained networks allows a fair analysis of compression and storage techniques, without introducing potential biases in the model selection and training procedures. Moreover, when fine-tuning weights after quantization, we preserved the same training configuration setup described in the original papers proposing the model. Finally, the uncompressed model is also employed as reference performance, and named hereafter as *baseline*.

4.3. Evaluation metrics

We performed comparisons focusing on the following metrics:

1. *accuracy* for classification and *mean squared error* (MSE) for regression (as in original papers);
2. occupancy ratio ψ (cfr. the end of the Introduction).

When only partly compressing the model, time and space performance only account for the actually compressed layers. The rest of the paper assesses the effectiveness of compression techniques in three scenarios, namely compressing:

1. only FC layers,
2. only convolutional layers, and
3. both layer types.

4.4. Software implementation

The code retrieved for baseline NNs was implemented in Python 3, using Tensorflow and Keras. We used the same environment for implementing compression and retraining. The software is distributed as a standalone Python 3 package. Source code, datasets and trained baseline networks are available at https://github.com/giosumarin/compare_dnn_compression. The provided code allows to replicate our experiments and compute the corresponding performance metrics. Moreover, the repository contains instructions for applying the compression techniques described

[2] https://github.com/BIGBALLON/CIFAR10-cnn.
[3] Precisely, by preserving training configuration we adapt the model input (for MNIST) and output (for CIFAR100) layer in order to deal with different data dimensions.
[4] The repository https://github.com/hkmztrk/DeepDTA. contains a script that produces the original model trained on KIBA; also in this case, we fine-tune this model on DAVIS by preserving training configuration.

in the paper to user-defined models/datasets and evaluate the results. The software processing pipeline for the compression of a generic DNN is composed of three steps, each corresponding to a dedicated script:

1. model training (skipped in the case of a pre-trained model);
2. model compression (selecting the desired combination of pruning, quantization, LRF, memorization format via instantiation of dedicated classes);
3. evaluation of the resulting occupancy ratio ψ.

5. Results

Before introducing the results obtained throughout the different experiments, we recap some recent findings that are useful for the purposes of our study.

5.1. Preliminary results from previous studies

In this subsection we summarize our previous results obtained when compressing only FC layers via pruning, CWS, and PWS methods, considering each layer separately, that is when each layer has its own k distinct weights [30]. They serve as a base reference for the different analyses presented here. *Compression techniques setup* Four of the benchmark datasets described in Section 4.1 (CIFAR100 is absent) and the two models introduced in Section 4.2 were used to compare the following compression schemes.

Pruning
The percentile level p was chosen in the set $p \in \{30, 40, 50, 60, 70, 80, 90, 95, 96, 97, 98, 99\}$ where levels smaller than 50, although not guaranteeing an occupancy < 1, were included because potentially useful in the combinations Pr-CWS and Pr-PWS (see last item in this list);

CWS The number k of representatives for VGG19 was selected in the set $\{2, 32, 128, 1024\}$ for the first two FC layers and in $\{2, 32\}$ for the (smaller) output layer; as DeepDTA is more compact, $k \in \{2, 32, 128\}$ in the three FC layers and $k \in \{2, 32\}$ for the output layer are considered.
PWS To have a fair comparison, k was set as in CWS.
Pr-X The combined application of pruning followed by the quantization $X \in \{\text{CWS}, \text{PWS}\}$ was tested in two variants: a) selection of best p in terms of performance, and then tuning of X as in previous points; b) the vice versa.

Fine-tuning of compressed weights Post-compression retraining was done using the same configuration as in original training. Data-guided tuning was applied only to learning rate ($3 \cdot 10^{-4}$ for pruning, 10^{-3} and 10^{-4} for PWS, CWS, and combined schemes), and maximum number of epochs, set to 100.

Performance assessment The top performance for each compression technique, and its configuration, is shown in Table 1, whereas the configuration improving the baseline (when existing) having the smallest memory requirement is shown in Table A.1 in the Appendix. Excluding discussions about these results, which are already present in the original paper, we only underline the fact that.

1) compression techniques providing the lowest occupancy, i.e., those combining connection pruning and quantization, still achieved to be competitive or better than the baseline w.r.t. performance;
2) no compression technique better than the other ones emerged.

The next sections extend and enrich these results, providing instead a better characterization of the different methods and their effectiveness in different scenarios.

Table 1
Top testing performance achieved by compression techniques. *Type* is the compression technique, while *Perf* contains Accuracy for VGG19 and MSE for DeepDTA (baseline is shown in brackets in the first column). ψ is the occupancy ratio. In bold the best results on each couple Net-Dataset.

Net-Dataset	Type	Config	Perf	ψ
VGG19-MNIST (0.9954)	Pr	96	0.9954	0.0800
	CWS	128–32-32	0.9957	0.3210
	PWS	32-32–2	**0.9958**	0.3090
	Pr-CWS a	96/128–32-32	0.9956	0.0390
	Pr-CWS b	96/128–32-32	0.9956	0.0390
	Pr-PWS a	96/32-128–32	0.9956	**0.0260**
	Pr-PWS b	50/32-32–2	**0.9958**	0.1870
VGG19-CIFAR10 (0.9344)	Pr	60	0.9365	0.8000
	CWS	32-32–2	**0.9371**	0.3060
	PWS	32–2-32	0.9363	0.0910
	Pr-CWS a	60/2–2-32	0.9366	0.0880
	Pr-CWS b	50/32-32–2	0.9370	0.2160
	Pr-PWS a	60/2–2-32	0.9363	0.0880
	Pr-PWS b	98/32–2-32	0.9365	**0.0120**
DeepDTA-KIBA (0.1756)	Pr	60	**0.1599**	0.8000
	CWS	128–128-32–2	0.1679	0.3900
	PWS	32–128-128–32	0.1761	0.4250
	Pr-CWS a	60/32–128-2–32	0.1666	**0.1870**
	Pr-CWS b	30/128–128-32–2	0.1644	0.3300
	Pr-PWS a	60/128–128-128–32	0.1769	0.2070
	Pr-PWS b	40/32–128-128–32	0.1683	0.2910
DeepDTA-DAVIS (0.3223)	Pr	80	**0.2242**	0.4000
	CWS	128–2-128–2	0.2320	0.2120
	PWS	128–32-32–32	0.2430	0.3240
	Pr-CWS a	80/32–128-2–32	0.2341	**0.1050**
	Pr-CWS b	40/128–2-128–2	0.2826	0.1910
	Pr-PWS a	80/128-128–32	0.2302	0.1220
	Pr-PWS b	60/128–32-32–32	0.2353	0.1600

Table 2
Comparison between unified and non-unified quantization. Same notations as in Table 1.

Net-Dataset	Type	Config	Perf	ψ
VGG19-MNIST (0.9954)	CWS	128–32-32	0.9957	0.3210
	uCWS	192	0.9957	0.2344
	PWS	32-32–2	**0.9958**	0.3090
	uPWS	66	0.9955	**0.1857**
VGG19-CIFAR10 (0.9344)	CWS	32-32–2	**0.9371**	0.3060
	uCWS	66	0.9370	0.1856
	PWS	32–2-32	0.9363	**0.0910**
	uPWS	66	0.9366	0.1857
DeepDTA-KIBA (0.1756)	CWS	128–128-32–2	0.1679	0.3900
	uCWS	290	**0.1609**	**0.2516**
	PWS	32–128-128–32	0.1761	0.4250
	uPWS	320	0.1631	0.2642
DeepDTA-DAVIS (0.3223)	CWS	128–2-128–2	0.2320	**0.2120**
	uCWS	260	0.2291	0.2496
	PWS	128–32-32–32	0.2430	0.3240
	uPWS	224	**0.2253**	0.2469

5.2. Evaluation of global and per-layer quantization

The experiments in the previous section made use of a *non-unified* quantization, selecting a specific k per layer and quantizing layers separately. In essence, all weight matrices could be quantized at the same time using a unique value of k, leading to a variant that we name hereafter *unified* quantization. The non-unified case has higher flexibility, counterbalanced by an increase of memory overhead, as each layer requires its own vector of representatives. On the other hand, the unified case entails only one dictionary, but the choice of k represents an 'one size fits all' solution, which might reduce the versatility and impact on the performance. To better understand it, here we extend the experiments of Section 5.1 by comparing these two approaches while preserving the experimental setting, that is compressing only FC layers on the same models and datasets, and measuring performance in terms of accuracy/MSE and occupancy ratio. Since in principle the non-unified variant uses distinct weights for each layer, a fair comparison is obtained by allowing the unified variant to use a number of distinct weights obtained summing up the number of distinct weights used by the non-unified approach across layers. The best non-unified models in terms of performance from Table 1 have been considered, and the corresponding unified variants have been trained. The resulting four experiments are summarized in Table 2, where the modalities exploiting the unified choice of k are marked as "uCWS" and "uPWS". The results point up that such variants tend to compress more than their non-unified counterparts, having in some cases negligible performance decay (classification), or even improvements (CIFAR10-uPWS and regression). Specifically, uCWS compresses up to 1.64× more than CWS (VGG19-CIFAR10), and uPWS up to 1.66× more than PWS (VGG19-MNIST), while almost preserving the accuracy (−0.03% and −0.01%, respectively). Besides, uCWS improves the MSE up to 4% on DeepDTA-KIBA (while compressing 1.55× more), and uPWS improves up to 8% on DeepDTA-DAVIS, even with a compression ratio of 1.31× higher. Nonetheless, the above-mentioned flexibility of non-unified variants in two cases leads to higher compression than the unified counterparts, specifically when they use lower k values in large layers. For instance, PWS has less than half occupancy compared to uPWS on VGG19-CIFAR10 by using only two distinct weights for the central hidden layer of VGG19 (the largest one). In order to confirm these trends, we repeated this experiment using the non-unified models reported in Table A.1 (best compression preserving the baseline performance), as shown in Table 3. The trend of Table 2 is preserved, although here the higher flexibility for non-unified methods is more marked (indeed they always compress more on classification data). For this reason and to better evaluate the behavior of unified versions, in the same table we also reported the additional results when varying $k \in \{2^i | 1 \leqslant i \leqslant 7\}$, and reporting the setting achieving the best compression ratio among those performing not worse than the non-unified counterparts. With this "non-constrained" setting, in most cases the occupancy is much lower than in non-unified variants (e.g., 4× smaller on VGG19-MNIST data). This is a nice result, suggesting that unified variants, apart being easier to configure (non-unified counterparts present a large number of combinations of k values across layers), tend to compress more and to perform better, mainly for regression problems. Accordingly, in the rest of the paper this variants will be used. Anyhow, the non-unified variants remain valid alternatives, able to attain better compression ratios in some particular cases (e.g., VGG19-MNIST data for the CWS method).

5.3. Comparison on FC layers

Another extension of the results in Section 5.1 consists in including in the evaluation two state-of-the-art quantization methods, precisely UQ and ECSQ[5] (cfr. Section 3.2 compression method based on matrix low-rank factorization (SLR, see Section 3.3). Further, to consolidate our analyses, henceforth we also include in our experiments the CIFAR100 dataset. Conforming to the results obtained in the previous section, the unified variants are tested as quantization strategies (denoted by prefixing a 'u' to the strategy name). The setting used up to now (compressing only FC layers and measuring accuracy/MSE and occupancy ratio) is maintained, but for a more complete comparison, a wider set of values for k, namely $\{2, 16, 32, 64, 128, 256\}$, is tried. The parameters λ (uECSQ) and δ (uUQ) have been tuned to give in output the number k of desired clusters, whereas to reduce the already massive set of experiments, we have set $d = 0$ for uUQ. In accordance with the sugges-

[5] The ECSQ technique does not accept the number k of distinct values as a hyperparameter. Therefore, we perform a preliminary tuning by trying $\lambda \in [10^{-13}, 10^{-2}]$, subsequently choosing the value yielding the desired k.

Table 3
Comparison between unified and non-unified variants when the configurations for non-unified methods are those reported in Table A.1. Same notations as in Table 1.

Net-Dataset	Type	Config	Perf	ψ
VGG19-MNIST (0.9954)	CWS	128-2–32	0.9954	0.1040
	uCWS	162	**0.9957**	0.2265
	uCWS	16	0.9954	0.1215
	PWS	1024-2–32	0.9955	0.1260
	uPWS	1058	0.9955	0.3121
	uPWS	2	0.9955	**0.0313**
VGG19-CIFAR10 (0.9344)	CWS	2–2-32	0.9360	0.0630
	uCWS	36	**0.9367**	0.1567
	uCWS	2	0.9362	**0.0313**
	PWS	2–2-32	0.9351	0.0630
	uPWS	36	0.9364	0.1576
	uPWS	2	0.9364	**0.0313**
DeepDTA-KIBA (0.1756)	CWS	32-32–2-2	0.1723	0.2280
	uCWS	68	**0.1601**	0.1843
	uCWS	8	0.1661	**0.0848**
	PWS	32–128-128–32	0.1761	0.4250
	uPWS	320	0.1631	0.2642
	uPWS	32	0.1718	0.1553
DeepDTA-DAVIS (0.3223)	CWS	2–2-2–2	0.2840	0.0630
	uCWS	8	0.2637	0.0900
	uCWS	4	0.2751	**0.0606**
	PWS	32-32–2-32	0.2430	0.2370
	uPWS	98	**0.2276**	0.2091
	uPWS	8	0.2783	0.0865

Fig. 1. Best accuracy (a) and best ψ at least attaining baseline accuracy (b) for the compared compression techniques applied to the FC layers of VGG19 on MNIST, CIFAR10 and CIFAR100 datasets. Values reported in the bars on the right denote the gain w.r.t. baseline.

tions of authors in [58], we set $c_r = c_s = 0.5$, while tuning $q \in \{8, 16, 32, 64, 128\}$.

To provide an insight on how pruning interacts with quantization, we also compare the methods Pr-X, preposing a pruning stage to quantization (cfr. Section 5.1), varying p in $\{30, 40, 50, 60, 70, 80, 90, 95, 96, 97, 98, 99\}$, and testing all possible combinations of p and k. All FC layers are compressed, and the overall compression ratio across them is computed. Figs. 1 and 2 summarize our main results, respectively for classification and regression. Further, the results for all combinations are shown in Table A.2 for quantization, whereas in Tables A.3,A.4 we report the extended results of best performance and best space ensuring baseline configurations for Pr-X methods (due to the large number of combinations of pruning and quantization hyperparameters, showing all combinations was not feasible in this case).

On classification data the top accuracy is attained by quantization methods, never worse than the baseline (Fig. 1(a)), with rare exceptions in which performance drastically worsens (e.g., uUQ with low k, Table A.2). However, for $k > 64$ (MNIST), $k > 32$ (CIFAR10) or $k > 2$ (CIFAR100), uUQ becomes competitive in accuracy, sensibly outdoing the occupancy ratio of the remaining methods (up to around $4\times$ lower). SRL exhibits good compression ratios, with a negligible drop in accuracy (cfr. Table A.5 for full results), whereas in terms of both ψ and accuracy Pr-uUQ (MNIST and CIFAR10) and Pr-uECSQ (CIFAR100) stand out. Even for regression it is difficult to characterize a winning method: the main difference is that SLR performance strongly decays, while compressing the lowest (see Fig. 2); uCWS and uEQCS tend to have the best MSE, with the latter being preferable for low values of k, even in terms of occupancy (Table A.2); uUQ and its combination with pruning often compress the most, but they still show a high MSE drop for too low values of k.

Although a clear rank of compressors does not emerge, we can attempt to list some remarks: 1) uUQ should be adopted only when enough distinct weights can be used; 2) uECSQ should be employed in the remaining cases, in which uPWS and uCWS (respectively on classification and regression) are valid alternatives; 3) in combination with pruning, uUQ compresses more than all other methods for classification, substantially confirming the results obtained with no pre-pruning; 4) in terms of best occupancy, uECSQ is competitive with uUQ, especially on CIFAR10, CIFAR100 and KIBA (Table A.4); 5) the tendency shown in Table A.2 is confirmed when pruning comes before sharing the weights, with the precious benefit of similarly performing while decreasing the occupancy by almost one order of magnitude.

5.4. Comparison on convolutional layers

This section is focused on the compression of convolutional layers only, with the aim to obtain useful information for the final experiment, where convolutional and FC layers will be compressed simultaneously. Only the performance (accuracy or MSE) is thereby evaluated here, to detect to most meaningful compression configurations on these layers. Before starting, it is worth pointing out that FC layers in CNNs typically contain most parameters, and low-rank factorization applied to convolution?lters mainly helps to speed up the time-taking convolution, while usually sensibly reducing the model accuracy [65]. Moreover, the authors of SLR discouraged its application to convolutional layers [58], and in light of these considerations, SLR has been excluded from the comparison carried out in this section.

Connection pruning. Although so far we have mainly applied pruning in synergy with quantization, here we dedicate a preliminary assessment to evaluate its effectiveness when applied only to convolutional tensors, to obtain an initial insight about its impact on the performance. The results are summarized in Table 4.

A main emerging feature is that, although pruning helps also in this case to improve the baseline accuracy (in brackets in the

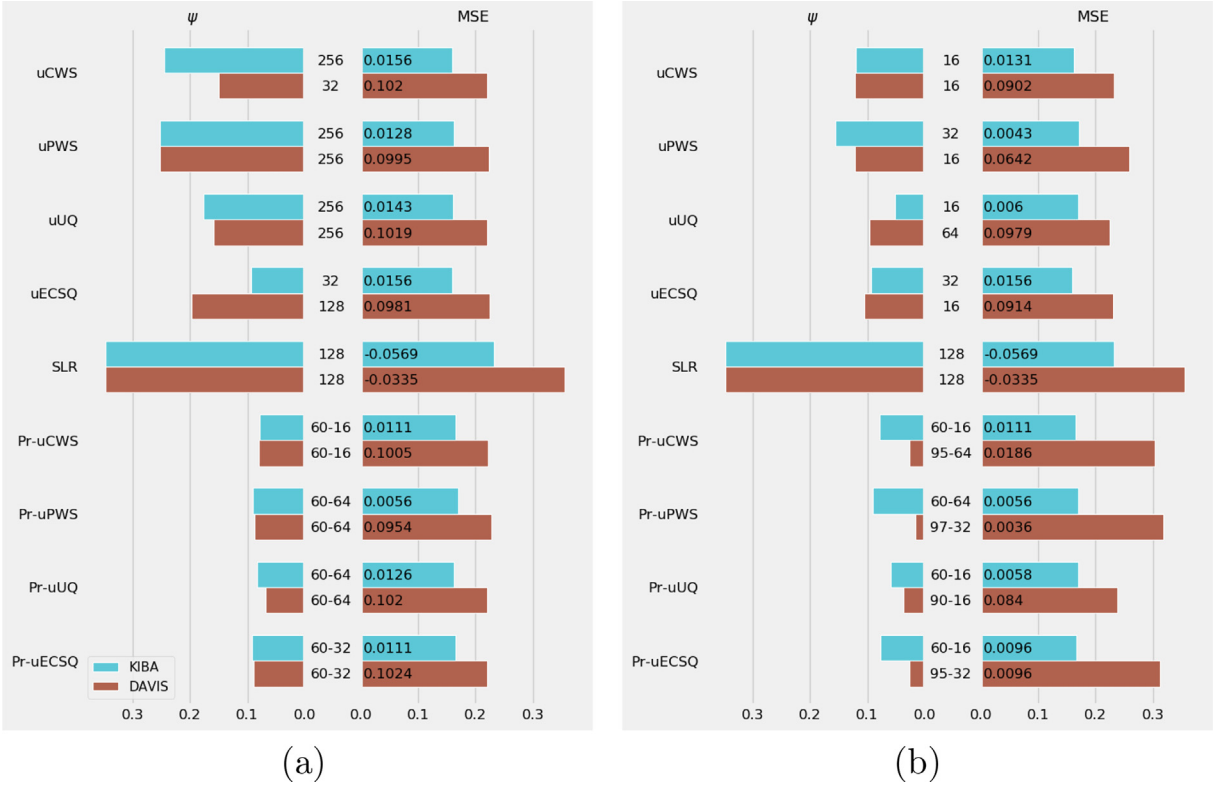


Fig. 2. Best MSE (a) and best ψ at least attaining baseline accuracy (b) for the compared compression techniques applied to the FC layers of DeepDTA on KIBA and DAVIS datasets. Same notations as in Fig. 1.

Table 4
Testing performance (accuracy for VGG19 and MSE for DeepDTA) when pruning convolutional layers. Column p contains the percentile level. Horizontal traits represent the limit level ensuring a performance gain.

	VGG19			DeepDTA	
p	**MNIST** (0.9954)	**CIFAR10** (0.9344)	**CIFAR100** (0.7126)	**KIBA** (0.1756)	**DAVIS** (0.3223)
10	0.9957	0.9355	0.7162	0.1561	0.2220
20	0.9957	0.9341	0.7176	0.1565	0.2233
30	0.9957	0.9337	0.7148	0.1566	0.2238
40	0.9957	0.9333	0.7123	0.1576	0.2218
50	0.9955	0.9289	0.7074	0.1571	0.2237
60	0.9956	0.9255	0.6994	0.1577	0.2224
70	0.9951	0.9179	0.6906	0.1600	0.2234
80	0.9944	0.9084	0.6773	0.2223	0.2433
90	0.9917	0.8802	0.6440	0.3139	0.3492
95	0.9907	0.7950	0.5870	0.3692	0.4136
96	0.9909	0.7608	0.5132	0.3796	0.4753
97	0.9903	0.6910	0.4122	0.4067	0.5180
98	0.9882	0.6154	0.2890	0.4576	0.5350
99	0.9852	0.5204	0.0852	0.5446	0.6548

table), the percentile level assuring an improvement is much lower compared to its counterpart on FC layers: up to $p = 60$ for MNIST, $p = 70$ for KIBA, only $p = 10$ and $p = 30$ for CIFAR10 and CIFAR100, and $p = 80$ for DAVIS. Moreover, the top performance is worse than that obtained when pruning FC layers (see Section 5.1). This suggests that the role of convolutional layers in a CNN is more critical and sensitive to a parameter reduction, which is reasonable considering that convolutional layers are responsible for input scan and elaboration, and that most parameters are contained in FC layers (at least in the considered models). Indeed, in most cases pruning on these layers cannot be increased much without having a sensible loss in performance. This deterioration would also be amplified by the application of quantization, still fostering a reduction of the pruning percentile to contain the performance decay. Taking into account also that the IM format would not benefit of connection pruning, in the experiment described in the next section (the one in which all network layers are compressed), only quantization will be applied to convolutional layers. *Quantization* The convolutional blocks have been compressed via unified quantization by considering $k \in \{32, 64, 128, 256\}$ (lower values of k performed too poorly). As in this experiment compressed models rarely beat the baseline, best performance and best occupancy results coincide and are reported in Fig. 3, while in Table A.6 we have shown the results of all configurations.

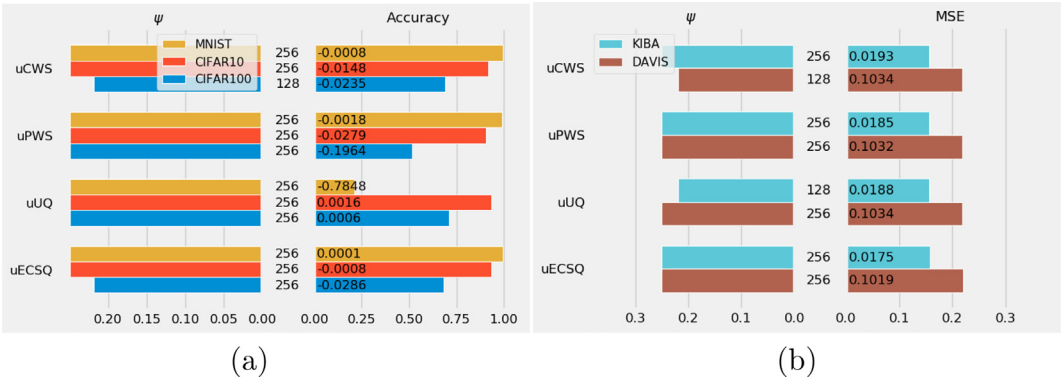
On classification data, the trend of uUQ with low values of k observed in Section 5.3 is confirmed and intensified, with even uPWS showing a similar behavior for $k = 32$ and $k = 64$. uCWS and uECSQ tend to perform better than the other two techniques, exhibiting higher robustness for small values of k. The lower accuracy induces to think that quantization should be much limited on these layers in classification settings. On the other hand, the performance on regression datasets (Fig. 3(b)) is better, more stable, and similar to those observed on FC layers. uUQ and uPWS are again competitive with the other two methods, with uUQ achieving the lowest MSE on KIBA, even when using the smallest tested k.

This experiment highlights that probably it is better to differently approach classification and regression problems when deciding the quantization level for convolutional layers: in the former case, we should not apply too radical quantizations, and $k \geqslant 256$ is preferable; in the latter, instead, it seems possible to adopt a more compressing configuration without getting unstable results. In the next section we leverage this cognition to attempt in designing the most appropriate and effective compression configuration for the combined experiment jointly considering FC and convolutional layers.

5.5. Comparison on the whole network

The final comparison among compression techniques involves compressing all layers in the CNN. As emphasized in previous sections, in this setting it is more convenient to not apply pruning to convolutional layers, since it helps preventing further accuracy decay and it does not affect the occupancy ratio. Furthermore, on FC layers we directly applied the combined methods Pr-X, due to their superior compression capabilities.

An interesting characteristic in this scenario is that the quantization methods here include all layers in the unified setup, hence requiring convolutional and FC layers to share the same



(a) (b)

Fig. 3. Best performance for the different quantization techniques applied only to convolutional layers of VGG19 (a) and DeepDTA (b). Same notations as in Fig. 1.

representatives; this allows a higher compression and offers thereby also a full insight of how the proposed strategies interact and perform when globally applied to the network. In accordance to the results obtained in Sections 5.1 and 5.4, the values $k \in \{32, 64, 128, 256\}$ have been retained. On the other side, the results obtained in Section 5.1 suggest to prune FC layers adopting specific values for p on each dataset: $p \in \{90, 92, 95, 97, 99\}$ for MNIST and CIFAR10 (here we extend this setting also to CIFAR100), $p \in \{50, 55, 60, 65, 70\}$ for KIBA, and $p \in \{70, 75, 80, 85, 90\}$ for DAVIS. We kept the same setting also for the SLR hyperparameter (q tuned in $\{8, 16, 32, 64\}$). To denote combined methods, we separated by a '/' the method applied to convolutional and that applied to FC layers: for instance, uCWS/SLR is the combined approach using uCWS and SLR to compress, respectively, convolutional and FC layers. When quantizing both layer types, the same k is used across layers (we remark this is a variant unified across the network). Fig. 4 depicts an extract of the best results obtained by each compression scheme (convolutional/FC), while Tables A.7–A.11 show the full results. For SLR we exhaustively reported only results for $q = 32, 64$, since for lower values of this parameter the overall performance sensibly dropped. Finally, to better understand these results, we remark that the occupancy ratios reported in the previous sections where limited to the compressed layers, not to the whole net; hence, since for the IM method the occupancy ratio cannot be lower than 0.25 (even higher when $k > 256$), the overall occupancy ratio is destined to increase with reference to that of FC layers. As a confirmation, the registered lowest occupancy not worsening baseline accuracy is 0.0576 (uPWS/Pr-uPWS, DAVIS, k = 128).

As a first important observation, the performance tends to deteriorate (as expected), mainly on classification (Fig. 4(a)); notwithstanding, we still have at least one configuration improving the baseline in all datasets: on MNIST, uECSQ/SLR ($k = 256, q = 32$) and occupying only 15.6% of the original network; on CIFAR10, uUQ/SLR ($k = 256, q = 64$) with 0.15 occupancy; on CIFAR100, uUQ/Pr-uUQ (k=256, $p = 90$) and occupancy 0.15; KIBA, uCWS/Pr-uCWS ($k = 32, p = 50$) and occupancy 0.115; on DAVIS, uPWS/Pr-uPWS ($k = 128, p = 90$) and occupancy 0.0576. These results are quite impressive if we consider that the model structure has not been modified. SLR confirms its main feature, that is a really effective performance on classification data, compensated by being the worst method on regression problems.

The flexibility of this approach aids the possibility to trade-off multiple criteria: for instance, in case we need to compress more on MNIST, the occupancy of the best model can be halved to the detriment of only 0.19% of its accuracy (uPWS, $p = 97, k = 32$). In classification settings, uUQ and uPWS still show unstable behavior, suggesting to not adopt them in this context. This performance confirms for uUQ the results obtained when quantizing only convolutional layers (Section 5.4), and it is likely to be amplified for both methods, when using low values of k, by the fact that the actual number of clusters can be smaller than k, due to a potential centroid overlapping during retraining (see Section 3.2). On the contrary, uUQ and uPWS are the most compressing methods in the regression settings (cfr. Tables 1), while improving the baseline performance. When performance should be preferred over compression, uCWS is able to achieve in 3 cases out of 4 the highest score, while still having an occupancy lower than 0.15 in all cases.

5.6. Discussion

We have carried out several experiments involving state-of-the-art CNN compression techniques with the purpose to shed light on their potential relationship with the type of problem (classification, regression), the layer to be compressed (convolutional, FC), the dataset at hand, and to assess their overall effectiveness in different settings. Here we recap the principal trends emerged so as to provide a reference to support researchers in the choice of a compression technique for an existing CNN, bearing in mind that this task might be at least partially specific to the setting here adopted (the pre-trained model and the dataset), and accordingly our results represent indications to guide such a choice.

In the comparison between unified and non-unified quantization techniques, the former basically compress more while being

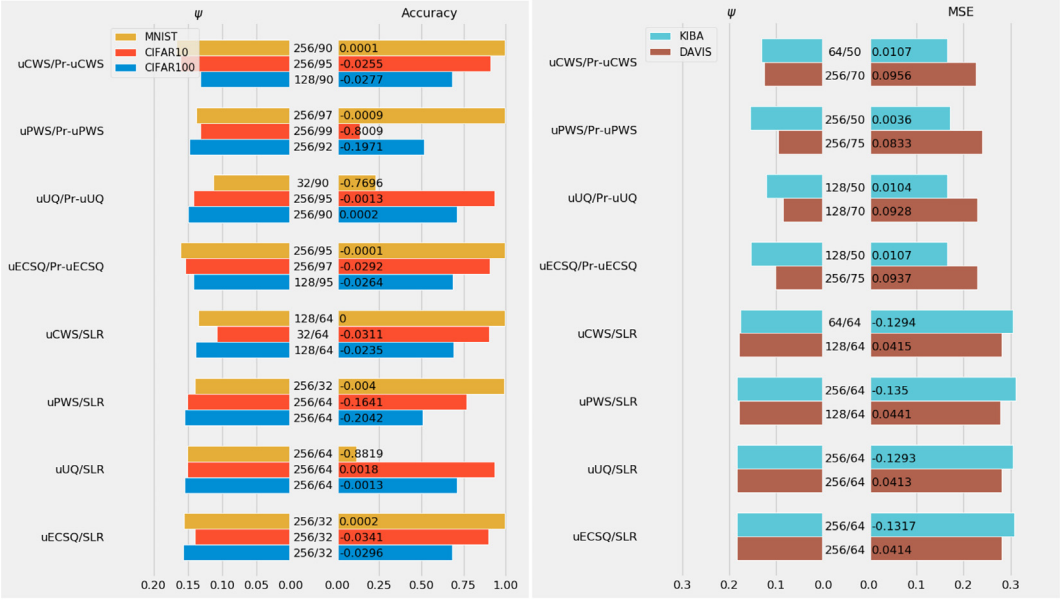


Fig. 4. Best performance when quantizing convolutional layers and applying SLR or pruning followed by quantization to FC layers of VGG19 (a) and DeepDTA (b). Same notations as in Fig. 1. The hyperparameter configuration is reported between the groups of bars.

at least as accurate as their non-unified variants, mainly on regression problems. Among them, the uniform quantization (uUQ) often achieved higher compression and better performance, but it has shown an unstable behavior with low values of k, so it is advisable to use it when the value of k is large enough (e.g., > 64). When this is not possible, uECSQ is preferable (similar performance but higher compression than uCWS and uPWS). It is always convenient to apply connection pruning along with quantization (Pr-X methods) on FC layers, when the pre-trained model has a topology that allows a relatively elevate pruning level ($p > 60$); indeed, it yields higher compression ratios and a negligible performance worsening. In classification problems, SLR is a valid alternative to Pr-X methods, and it is also faster (as it does not require retraining); however, its application is not recommended to regression problems, where we observed an important performance degeneration.

The compression of convolutional layers instead should be handled gingerly: we observed a high sensitivity to parameter reduction in this case. Quantization methods almost never improved the baseline on classification data, and accordingly it has not been possible to apply their combination with pruning. Among quantization methods, it is preferable to avoid uUQ and uPWS, which, as on FC layers, exhibit unstable behaviors. Instead, quantizing convolutional layers on regression data is less problematic; indeed, in this case roughly all techniques tend to compress a little less w.r.t. FC layers while performing alike.

The final experiment, focusing on the simultaneous compression of convolutional and FC layers, has finally provided some readable facts: compressed models still outmatch the uncompressed ones in most cases; some unstable behavior of uUQ and uPWS on classification data are still intensified, and accordingly uCWS and uECSQ are better choices in this case; compression schemes including uCWS quantization often attain the top performance on both regression and classification datasets, hinting a higher robustness of this quantization method to heterogeneous layers and problems; on regression problems, schemes exploiting uUQ and uPWS quantization tend to compress more, with uUQ, even in this setting, showing the best occupancy/performance trade-off.

6. Conclusions

This work proposed the first extensive empirical comparison, to the best of our knowledge, of several state-of-the-art-compression techniques for pre-trained CNNs, while keeping their original topology. Connection pruning, weight quantization, low-rank matrix factorization and a bunch of their variants and combinations have been studied. We have compressed first separately and then simultaneously both convolutional and FC layers, obtaining full insights about their impact on the model performance. Our results have confirmed the potential of compression techniques to trade-off between model performance and its space requirements, with the possibility to preserve at least the same accuracy of the baseline models while getting models more than $6\times$ smaller. Indeed, for each learning task and corresponding deep neural network we have found at least one compression strategy on the whole network (convolutional and fully-connected layers) preserving or improving the original performance: uECSQ/SLR ($k = 256, q = 32$) for MNIST, yielding a model around $6.41\times$ smaller; uUQ/SLR ($k = 256, q = 64$) for CIFAR10, with a space reduction of $6.66\times$; uUQ/Pr-uUQ ($k = 256, p = 90$) for CIFAR100, space reduction $6.66\times$; uCWS/PruCWS ($k = 32, p = 50$) for KIBA, space reduction around $8.7\times$; finally, uPWS/Pr-uPWS ($k = 128, p = 90$) for DAVIS, yielding a model even $17.36\times$ smaller. In addition, we have brought out some specific characteristics of individual compressors, and characterized their pros and cons with respect to the problem to be tackled. Although our analyses are limited to the models and datasets used here, the trends emerged can be of great support and help in the analysis and choice of configurations and compression methods more appropriate for who aims at compressing a state-of-the-art convolutional neural network in any given application domain.

CRediT authorship contribution statement

Giosué Cataldo Marinó: Software, Validation, Investigation, Writing – review & editing, Visualization. **Alessandro Petrini:** Software, Validation, Investigation, Writing – review & editing, Visualization. **Dario Malchiodi:** Conceptualization, Formal analysis, Methodology, Writing – original draft, Writing – review & editing, Supervision. **Marco Frasca:** Conceptualization, Formal analysis, Methodology, Writing – original draft, Writing – review & editing, Supervision, Funding acquisition, Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

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). Part of this work was done while D. Malchiodi was visiting scientist at Inria Sophia-Antipolis/I3S CNRS Université Côte d'Azur (France).

Appendix A. Full result tables

See Tables A.1–A.11.

Table A.1
Best occupancy ratio ensuring no decay in performance w.r.t. the uncompressed model. Same notations as in Table 1.

Net-Dataset	Type	Config	Perf	ψ
VGG19-MNIST (0.9954)	Pr	97	0.9953	0.0600
	CWS	128-2–32	0.9954	0.1040
	PWS	1024-2–32	0.9955	0.1260
	Pr-CWS a	96/2-128–2	0.9954	0.0380
	Pr-CWS b	96/128–32-32	**0.9956**	0.0390
	Pr-PWS a	96/32-128–32	**0.9956**	0.0260
	Pr-PWS b	97/32-32–2	0.9955	**0.0180**
	Pr	99	0.9357	0.0200
	CWS	2–2-32	0.9360	0.0630
	PWS	2–2-32	0.9351	0.0630

(continued on next page)

Table A.1 (*continued*)

Net-Dataset	Type	Config	Perf	ψ
	Pr-CWS a	60/2–2-32	**0.9366**	0.0880
	Pr-CWS b	99/32-32–2	0.9358	**0.0060**
	Pr-PWS a	60/2–2-32	0.9363	0.0880
VGG19-CIFAR10 (0.9344)	Pr-PWS b	99/32–2-32	0.9363	**0.0060**
	Pr	60	**0.1599**	0.8000
	CWS	32-32–2-2	0.1723	0.2280
	PWS	32–128-128–32	0.1761	0.4250
	Pr-CWS a	60/32–2-32–2	0.1739	**0.1270**
	Pr-CWS b	60/128–128-32–2	0.1712	0.2220
	Pr-PWS a	60/128–128-128–32	0.1769	0.2070
DeepDTA-KIBA (0.1756)	Pr-PWS b	50/32–128-128–32	0.1702	0.2430
	Pr	90	0.2425	0.2000
	CWS	2–2-2–2	0.2840	0.0630
	PWS	32-32–2-32	0.2567	0.2370
	Pr-CWS a	80/32–2-2–32	**0.2367**	0.0790
	Pr-CWS b	60/128–2-128–2	0.2906	0.1480
	Pr-PWS a	90/128–32-32–32	0.2671	**0.0600**
DeepDTA-DAVIS (0.3223)	Pr-PWS b	80/32–2-2–32	0.2943	0.0770

Table A.2
Extended results of quantization methods on FC layers. The performance of the uncompressed model is shown in brackets in the headline. The best results for each configuration (k, dataset) are shown in bold.

		MNIST (0.9954)		CIFAR10 (0.9344)		CIFAR100 (0.7126)		KIBA (0.1756)		DAVIS (0.3223)	
k	Method	Acc	ψ	Acc	ψ	Acc	ψ	MSE	ψ	MSE	ψ
2	uCWS	0.2266	**0.0313**	0.9355	**0.0313**	**0.7166**	0.0313	80.0142	0.0313	22.3118	0.0313
	uPWS	**0.9951**	**0.0313**	0.9363	**0.0313**	0.7121	0.0313	**0.7699**	0.0313	29.6357	0.0313
	uUQ	0.2213	**0.0313**	0.1981	**0.0313**	0.0204	0.0313	105.6028	0.0313	**22.3100**	0.0313
	uECSQ	0.9901	0.0461	**0.9368**	0.0472	0.7122	0.0313	80.0640	0.0313	22.3111	0.0313
16	uCWS	0.9954	0.1215	0.9366	0.1179	**0.7167**	0.1116	**0.1625**	0.1197	0.2321	0.1211
	uPWS	0.9953	0.1212	0.9368	0.1212	0.7103	0.1216	0.1847	0.1225	0.2581	0.1217
	uUQ	0.2159	**0.0314**	0.1991	**0.0316**	0.7156	**0.0390**	0.1696	**0.0523**	22.3108	**0.0413**
	uECSQ	**0.9957**	0.0949	**0.9369**	0.0801	0.7163	0.0799	0.1631	0.1031	**0.2309**	0.1059
32	uCWS	**0.9957**	0.1467	0.9365	0.1513	**0.7170**	0.1525	0.1606	0.1491	**0.2203**	0.1502
	uPWS	0.9955	0.1544	0.9365	0.1545	0.7109	0.1547	0.1713	0.1551	0.2432	0.1550
	uUQ	0.2239	**0.0322**	**0.9370**	**0.0355**	**0.7170**	**0.0563**	0.1639	**0.0841**	6.9812	**0.0652**
	uECSQ	0.9955	0.1338	0.9366	0.1241	0.7163	0.1491	**0.1600**	0.0933	0.2283	0.1183
64	uCWS	**0.9957**	0.1835	0.9364	0.1836	0.7162	0.1829	**0.1608**	0.1825	**0.2237**	0.1853
	uPWS	0.9955	0.1867	0.9365	0.1867	0.7126	0.1868	0.1674	0.1873	0.2258	0.1873
	uUQ	0.8908	**0.0397**	0.9362	**0.0498**	0.7161	**0.0794**	0.1639	**0.1161**	0.2244	**0.0959**
	uECSQ	0.9956	0.1841	**0.9366**	0.1359	**0.7166**	0.1715	0.1618	0.1237	0.2251	0.1636
128	uCWS	0.9956	0.2134	**0.9364**	0.2162	0.7164	0.2165	**0.1608**	0.2149	**0.2214**	0.2169
	uPWS	0.9954	0.2184	0.9363	0.2184	0.7110	0.2185	0.1659	0.2194	0.2323	0.2194
	uUQ	0.9955	**0.0559**	0.9363	**0.0736**	0.7167	**0.1085**	0.1620	**0.1498**	0.2304	**0.1272**
	uECSQ	**0.9958**	0.1953	**0.9364**	0.1787	**0.7171**	0.2055	0.1615	0.1894	0.2242	0.1967
256	uCWS	**0.9957**	0.2477	**0.9367**	0.2468	0.7168	0.2475	**0.1600**	0.2446	0.2272	0.2494
	uPWS	0.9955	0.2500	0.9363	0.2500	0.7112	0.2500	0.1628	0.2519	0.2228	0.2519
	uUQ	0.9953	**0.0971**	0.9364	**0.1154**	**0.7170**	**0.1247**	0.1613	**0.1772**	**0.2204**	**0.1593**
	uECSQ	**0.9957**	0.2283	**0.9367**	0.2395	0.7167	0.2383	0.1603	0.2187	0.2309	0.2069

Table A.3
Pr-X methods yielding the best performing model in terms of Accuracy when compressing FC layers. Same notations as in Table 1.

Net-Dataset	Method	p-k	Perf	ψ
	Pr-uCWS	60–16	**0.9955**	0.0777
VGG19-MNIST (0.9954)	Pr-uPWS	60–32	**0.9955**	0.0807
	Pr-uUQ	99–32	0.9954	**0.0057**
	Pr-uECSQ	95–16	0.9954	0.0276
	Pr-uCWS	95–32	0.9365	0.0338
VGG19-CIFAR10 (0.9344)	Pr-uPWS	60–16	**0.9366**	0.0617
	Pr-uUQ	99–16	0.9363	**0.0057**
	Pr-uECSQ	60–16	0.9365	0.0754
	Pr-uCWS	40–256	0.7172	0.1682

Table A.3 (*continued*)

Net-Dataset	Method	*p-k*	Perf	ψ
VGG19-CIFAR100 (0.7126)	Pr-uPWS	98–32	0.7156	0.0109
	Pr-uUQ	99–32	**0.7178**	0.0057
	Pr-uECSQ	99–32	**0.7178**	**0.0056**
	Pr-uCWS	60–16	0.1645	**0.0791**
DeepDTA-KIBA (0.1756)	Pr-uPWS	60–64	0.1700	0.0902
	Pr-uUQ	60–64	**0.1630**	0.0825
	Pr-uECSQ	60–32	0.1645	0.0918
	Pr-uCWS	60–16	0.2218	0.0795
DeepDTA-DAVIS (0.3223)	Pr-uPWS	60–64	0.2269	0.0875
	Pr-uUQ	60–64	0.2203	**0.0675**
	Pr-uECSQ	60–32	**0.2199**	0.0887

Table A.4
Pr-X methods yielding the smallest model performing not worse than baseline when compressing only FC layers. Same notations as in Table 1.

Net-Dataset	Method	*p-k*	Perf	ψ
VGG19-MNIST (0.9954)	Pr-uCWS	80–64	0.9954	0.0577
	Pr-uPWS	60–32	0.9954	0.0807
	Pr-uUQ	99–32	0.9954	**0.0057**
	Pr-uECSQ	95–16	0.9954	0.0276
VGG19-CIFAR10 (0.9344)	Pr-uCWS	99–32	0.9361	**0.0055**
	Pr-uPWS	99–32	0.9344	**0.0055**
	Pr-uUQ	99–16	**0.9363**	0.0056
	Pr-uECSQ	99–32	**0.9363**	**0.0055**
VGG19-CIFAR100 (0.7126)	Pr-uCWS	98–64	0.7156	0.0108
	Pr-uPWS	98–32	0.7156	0.0109
	Pr-uUQ	99–16	**0.7178**	0.0057
	Pr-uECSQ	99–16	0.7173	**0.0055**
DeepDTA-KIBA (0.1756)	Pr-uCWS	60–16	**0.1645**	0.0790
	Pr-uPWS	60–64	0.1700	0.0901
	Pr-uUQ	60–16	0.1698	**0.0596**
	Pr-uECSQ	60–16	0.1660	0.0766
DeepDTA-DAVIS (0.3223)	Pr-uCWS	95–64	0.3037	0.0268
	Pr-uPWS	97–32	0.3187	**0.0159**
	Pr-uUQ	90–16	**0.2383**	0.0365
	Pr-uECSQ	95–32	0.3127	0.0271

Table A.5
SLR combinations applied only FC layers. Same notations as in Table 1.

Net-Dataset	q	Perf	ψ
VGG19-MNIST (0.9954)	8	0.4997	0.0072
	16	0.8976	0.0130
	32	**0.9955**	0.0238
	64	0.9954	0.0455
	128	0.9954	0.0888
VGG19-CIFAR10 (0.9344)	8	0.3682	0.0072
	16	0.8371	0.0130
	32	**0.9343**	0.0238
	64	0.9343	0.0455
	128	0.9343	0.0888
VGG19-CIFAR100 (0.7126)	8	0.0510	0.0071
	16	0.2914	0.0141
	32	0.7091	0.0282
	64	0.7103	0.0564
	128	**0.7125**	0.1067
DeepDTA-KIBA (0.1756)	8	0.5568	0.0220
	16	0.4999	0.0437

(*continued on next page*)

Table A.5 (*continued*)

Net-Dataset	q	Perf	ψ
	32	0.4060	0.0871
	64	0.3055	0.1740
	128	**0.2325**	0.3476
DeepDTA-DAVIS (0.3223)	8	0.7614	0.0220
	16	0.6026	0.0437
	32	0.4453	0.0871
	64	0.4085	0.1740
	128	**0.3558**	0.3476

Table A.6
Performance of quantization techniques applied to convolutional layers. Accuracy and MSE are reported respectively for DeepDTA and VGG19. Top results for each dataset are shown in bold.

		VGG19			DeepDTA	
k	**Method**	**MNIST (0.9954)**	**CIFAR10 (0.9344)**	**CIFAR100 (0.7126)**	**KIBA (0.1756)**	**DAVIS (0.3223)**
	uCWS	0.9941	**0.9109**	**0.6741**	0.1570	0.2221
32	uPWS	0.9918	0.1129	0.1141	0.1634	**0.2209**
	uUQ	0.1434	0.1210	0.0100	**0.1557**	0.2234
	uECSQ	**0.9943**	0.8941	0.6053	0.1582	0.2301
	uCWS	**0.9943**	**0.9036**	**0.6850**	**0.1575**	0.2219
64	uPWS	0.9932	0.1234	0.1482	0.1602	0.2234
	uUQ	0.2059	0.1746	0.0100	0.1571	**0.2214**
	uECSQ	0.9933	0.9013	0.6697	0.1586	0.2326
	uCWS	0.9943	**0.9133**	0.6891	**0.1563**	**0.2189**
128	uPWS	0.9934	0.9020	0.3665	0.1579	0.2211
	uUQ	0.2076	0.1842	**0.7087**	0.1568	0.2216
	uECSQ	**0.9952**	0.9030	0.6791	0.1582	0.2315
	uCWS	0.9946	0.9196	0.6889	**0.1563**	0.2193
256	uPWS	0.9936	0.9065	0.5162	0.1571	0.2191
	uUQ	0.2106	0.9073	**0.7132**	0.1576	0.2189
	uECSQ	**0.9955**	**0.9336**	0.6840	0.1581	0.2204

Table A.7
Model accuracy after applying quantization to convolutional layers and pruning + quantization or only SLR to FC layers of VGG19 trained on the MNIST dataset. The *k* value used is the same for convolutional and FC layers. Compression ratio (ψ) in brackets. Baseline of the uncompressed model is reported in the top line. The configurations attaining the top accuracy and the top compression for each value of *k* are shown in bold.

		VGG19 – MNIST (0.9954)						
		Quantization +					SLR	
k	**Method**	**Pr: 90**	**Pr: 92**	**Pr: 95**	**Pr: 97**	**Pr: 99**	**q: 32**	**q: 64**
	uCWS	0.9936 (0.1150)	0.9937 (0.1144)	0.9936 (0.1103)	0.9916 (0.1047)	0.9937 (0.0992)	0.9941 (0.0919)	0.9941 (0.1024)
	uPWS	0.1538 (0.0988)	0.9935 (0.0981)	0.9934 (0.0940)	0.9935 (0.0884)	0.9932 (**0.0831**)	0.9293 (0.0919)	0.9286 (0.1024)
32	uUQ	0.2258 (0.1131)	0.1374 (0.1128)	0.1945 (0.1087)	0.1898 (0.1038)	0.1135 (0.0989)	0.1135 (0.0919)	0.1135 (0.1024)
	uECSQ	0.9931 (0.1167)	0.994 (0.1154)	0.9945 (0.1107)	0.9944 (0.1048)	0.9929 (0.0992)	**0.9949** (0.0923)	**0.9949** (0.1032)
	uCWS	0.9943 (0.1338)	0.9941 (0.1323)	0.9914 (0.1272)	0.9932 (0.1211)	0.9943 (0.1153)	0.9945 (0.1080)	0.9945 (0.1185)
	uPWS	0.9938 (0.1162)	0.9942 (0.1150)	0.9943 (0.1105)	0.9943 (0.1047)	0.9942 (**0.0992**)	0.9612 (0.1080)	0.9617 (0.1185)
64	uUQ	0.1325 (0.1292)	0.1334 (0.1289)	0.1361 (0.1254)	0.1330 (0.1203)	0.1221 (0.1152)	0.1135 (0.1080)	0.1135 (0.1185)
	uECSQ	0.9941 (0.1344)	0.9942 (0.1328)	0.9465 (0.1275)	0.9948 (0.1213)	0.9938 (0.1154)	**0.9951** (0.1240)	0.9951 (0.1346)
	uCWS	0.9944 (0.1509)	0.9941 (0.1492)	0.9952 (0.1438)	0.9931 (0.1377)	0.9941 (0.1316)	0.9953 (0.1240)	0.9954 (0.1346)
	uPWS	0.9937 (0.1336)	0.9935 (0.1322)	0.9939 (0.1271)	0.9796 (0.1211)	0.9592 (0.1153)	0.9882 (0.1240)	0.9883 (0.1346)
128	uUQ	0.2090 (0.1302)	0.1887 (0.1294)	0.1661 (0.1257)	0.1601 (0.1204)	0.2007 (**0.1152**)	0.1009 (0.1240)	0.1009 (0.1346)
	uECSQ	0.9946 (0.1520)	0.9949 (0.1501)	0.9925 (0.1442)	0.9926 (0.1378)	0.9943 (0.1316)	**0.9955** (0.1401)	0.9954 (0.1506)
	uCWS	0.9945 (0.1685)	0.9955 (0.1666)	0.9952 (0.1608)	0.9953 (0.1540)	0.9947 (0.1472)	0.9954 (0.1401)	0.9954 (0.1507)
	uPWS	0.9940	0.9939	0.9867	0.9945	0.9886	0.9914	0.9914

Table A.7 (*continued*)

VGG19 – MNIST (0.9954)								
		Quantization +					SLR	
k	**Method**	**Pr: 90**	**Pr: 92**	**Pr: 95**	**Pr: 97**	**Pr: 99**	**q: 32**	**q: 64**
		(0.1512)	(0.1494)	(0.1438)	(0.1375)	(0.1315)	(0.1401)	(0.1507)
256	uUQ	0.1247	0.1170	0.1993	0.1139	0.2077	0.1009	0.1009
		(0.1463)	(0.1459)	(0.1423)	(0.1368)	(**0.1313**)	(0.1401)	(0.1507)
	uECSQ	0.9949	0.9952	0.9953	0.9948	0.9938	**0.9956**	0.9956
		(0.1696)	(0.1674)	(0.1611)	(0.1543)	(0.1478)	(0.1562)	(0.1667)

Table A.8
Results for the CIFAR10 dataset. Same notations as in Table A.7.

VGG19 – CIFAR10 (0.9344)								
		Quantization +					SLR	
k	**Method**	**Pr: 90**	**Pr: 92**	**Pr: 95**	**Pr: 97**	**Pr: 99**	**q: 32**	**q: 64**
	uCWS	0.8810	0.8818	0.8655	0.8668	0.8822	**0.8962**	0.8964
		(0.1161)	(0.1150)	(0.1104)	(0.1047)	(0.0992)	(0.0919)	(0.1024)
	uPWS	0.1173	0.1170	0.1233	0.1221	0.1133	0.3100	0.3157
		(0.0993)	(0.0983)	(0.0941)	(0.0886)	(**0.0831**)	(0.0919)	(0.1024)
32	uUQ	0.1335	0.1110	0.1061	0.1362	0.1180	0.1000	0.1000
		(0.1131)	(0.1128)	(0.1087)	(0.1038)	(0.0989)	(0.0919)	(0.1024)
	uECSQ	0.8706	0.8731	0.8731	0.8668	0.8794	0.8912	0.8910
		(0.1172)	(0.1176)	(0.1139)	(0.1049)	(0.0992)	(0.0921)	(0.1025)
	uCWS	0.8898	0.8904	0.8892	0.8913	0.8923	**0.9033**	0.9031
		(0.1341)	(0.1328)	(0.1274)	(0.1213)	(0.1154)	(0.1080)	(0.1185)
	uPWS	0.1210	0.1216	0.1158	0.1156	0.1098	0.3755	0.3804
		(0.1165)	(0.1153)	(0.1105)	(0.1048)	(**0.0992**)	(0.1080)	(0.1185)
64	uUQ	0.1385	0.1601	0.1659	0.1673	0.1678	0.1000	0.1000
		(0.1292)	(0.1289)	(0.1255)	(0.1203)	(0.1152)	(0.1080)	(0.1185)
	uECSQ	0.8822	0.8774	0.8799	0.8799	0.8757	0.8921	0.8933
		(0.1348)	(0.1331)	(0.1307)	(0.1214)	(0.1154)	(0.1081)	(0.1188)
	uCWS	0.8919	0.8944	0.8936	0.9022	**0.9027**	0.8897	0.8901
		(0.1514)	(0.1497)	(0.1441)	(0.1376)	(0.1315)	(0.1240)	(0.1346)
	uPWS	0.1210	0.1164	0.1290	0.1172	0.1263	0.5905	0.5896
		(0.1339)	(0.1324)	(0.1272)	(0.1212)	(0.1154)	(0.1240)	(0.1346)
128	uUQ	0.1440	0.1409	0.1414	0.1494	0.1323	0.1098	0.1099
		(0.1297)	(0.1294)	(0.1258)	(0.1206)	(**0.1152**)	(0.1240)	(0.1346)
	uECSQ	0.8917	0.8889	0.8899	0.8955	0.8995	0.8941	0.8939
		(0.1523)	(0.1503)	(0.1475)	(0.1379)	(0.1316)	(0.1240)	(0.1346)
	uCWS	0.9083	0.9089	0.9089	0.9088	0.9080	0.8983	0.8989
		(0.1693)	(0.1672)	(0.1611)	(0.1544)	(0.1478)	(0.1401)	(0.1506)
	uPWS	0.1270	0.1239	0.1207	0.1244	0.1335	0.7703	0.7716
		(0.1515)	(0.1497)	(0.1440)	(0.1377)	(0.1315)	(0.1401)	(0.1506)
256	uUQ	0.9317	0.9305	0.9331	0.9328	0.9309	0.9359	**0.9362**
		(0.1461)	(0.1457)	(0.1420)	(0.1367)	(**0.1314**)	(0.1401)	(0.1506)
	uECSQ	0.9034	0.9040	0.9031	0.9052	0.9037	0.9003	0.9000
		(0.1699)	(0.1676)	(0.1643)	(0.1544)	(0.1478)	(0.1401)	(0.1506)

Table A.9
Results for the CIFAR100 dataset. Same notations as in Table A.7.

VGG19 – CIFAR100 (0.7126)								
		Quantization +					SLR	
k	**Method**	**Pr: 90**	**Pr: 92**	**Pr: 95**	**Pr: 97**	**Pr: 99**	**q: 32**	**q: 64**
	uCWS	0.6642	0.6605	0.6611	0.6685	0.6543	**0.6729**	0.6720
		(0.1001)	(0.0985)	(0.0938)	(0.0880)	(**0.0823**)	(0.0934)	(0.1073)
	uPWS	0.0201	0.0582	0.0453	0.0185	0.0136	0.0879	0.0983
		(0.0973)	(0.0967)	(0.0930)	(0.0876)	(**0.0823**)	(0.0934)	(0.1073)
32	uUQ	0.0100	0.0104	0.0099	0.0100	0.0100	0.0100	0.0100
		(0.0983)	(0.0977)	(0.0935)	(0.0878)	(0.0824)	(0.0934)	(0.1073)
	uECSQ	0.6432	0.6401	0.6398	0.6344	0.6312	0.6532	0.6547
		(0.1101)	(0.1082)	(0.1023)	(0.0961)	(0.0911)	(0.1031)	(0.1232)
	uCWS	0.6812	0.6789	0.6803	0.6765	0.5905	0.6808	**0.6843**
		(0.1168)	(0.1146)	(0.1101)	(0.1040)	(**0.0982**)	(0.1094)	(0.1232)
	uPWS	0.0441	0.0904	0.0800	0.0556	0.0135	0.1190	0.1341
		(0.1148)	(0.1135)	(0.1092)	(0.1036)	(**0.0982**)	(0.1094)	(0.1232)
64	uUQ	0.0107	0.0100	0.0100	0.0106	0.0100	0.0101	0.0095
		(0.1154)	(0.1143)	(0.1095)	(0.1040)	(0.0985)	(0.1094)	(0.1232)
	uECSQ	0.6742	0.6784	0.6776	0.6766	0.6712	0.6656	0.6662

(*continued on next page*)

Table A.9 (*continued*)

VGG19 – CIFAR100 (0.7126)								
		Quantization +					SLR	
k	**Method**	**Pr: 90**	**Pr: 92**	**Pr: 95**	**Pr: 97**	**Pr: 99**	**q: 32**	**q: 64**
		(0.1315)	(0.1301)	(0.1254)	(0.1195)	(0.1141)	(0.1253)	(0.1391)
	uCWS	0.6849	0.6841	0.6717	0.6728	0.5952	0.6874	0.6891
		(0.1316)	(0.1305)	(0.1255)	(0.1196)	(**0.1141**)	(0.1253)	(0.1391)
	uPWS	0.3434	0.3656	0.3504	0.3164	0.0864	0.3386	0.3589
		(0.1325)	(0.1308)	(0.1258)	(0.1198)	(0.1142)	(0.1253)	(0.1391)
128	uUQ	0.7080	0.7061	0.7049	0.7041	0.2913	0.7050	**0.7084**
		(0.1316)	(0.1313)	(0.1263)	(0.1204)	(0.1144)	(0.1253)	(0.1391)
	uECSQ	0.6840	0.6847	0.6862	0.6801	0.6788	0.6780	0.6801
		(0.1491)	(0.1474)	(0.1419)	(0.1359)	(0.1301)	(0.1412)	(0.1550)
	uCWS	0.6786	0.6814	0.6787	0.6797	0.6772	0.6866	0.6889
		(0.1501)	(0.1483)	(0.1423)	(0.1361)	(0.1303)	(0.1412)	(0.1550)
	uPWS	0.5137	0.5155	0.5078	0.5017	0.4624	0.5071	0.5084
		(0.1499)	(0.1480)	(0.1423)	(0.1361)	(**0.1302**)	(0.1412)	(0.1550)
256	uUQ	**0.7138**	0.7128	0.7122	0.7110	0.7062	0.7107	0.7113
		(0.1497)	(0.1482)	(0.1428)	(0.1366)	(0.1305)	(0.1412)	(0.1550)
	uECSQ	0.6850	0.6845	0.6842	0.6829	0.6847	0.6830	0.6826
		(0.1666)	(0.1646)	(0.1591)	(0.1533)	(0.1478)	(0.1571)	(0.1709)

Table A.10
Model MSE after applying quantization to convolutional layers and pruning + quantization or only SLR to FC layers of DeepDTA trained on the KIBA dataset. Same notations as in Table A.7.

DeepDTA – KIBA (0.1756)								
		Quantization +					SLR	
k	**Method**	**Pr: 50**	**Pr: 55**	**Pr: 60**	**Pr: 65**	**Pr: 70**	**q: 32**	**q: 64**
	uCWS	**0.1680**	0.1693	0.1717	0.1780	0.1954	0.4222	0.3117
		(0.1149)	(0.1085)	(0.1015)	(0.0953)	(0.0884)	(0.0956)	(0.1718)
	uPWS	0.1918	0.1978	0.1908	0.3108	0.3946	0.4315	0.3329
		(0.1021)	(0.0950)	(0.0883)	(0.0779)	(**0.0712**)	(0.0956)	(0.1718)
32	uUQ	0.1686	0.1690	0.1732	0.1987	0.3272	0.4222	0.3096
		(0.0869)	(0.0839)	(0.0806)	(0.0773)	(0.0733)	(0.0956)	(0.1718)
	uECSQ	0.1679	0.1684	0.1748	0.1777	0.1970	0.4174	0.3094
		(0.1272)	(0.1189)	(0.1094)	(0.1002)	(0.0913)	(0.0956)	(0.1718)
	uCWS	**0.1649**	0.1664	0.1704	0.1758	0.1938	0.4123	0.3050
		(0.1320)	(0.1240)	(0.1158)	(0.1088)	(0.1007)	(0.0994)	(0.1756)
	uPWS	0.1781	0.1789	0.1827	0.1956	0.3533	0.4255	0.3166
		(0.1198)	(0.1115)	(0.1032)	(0.0913)	(0.0833)	(0.0994)	(0.1756)
64	uUQ	0.1658	0.1674	0.1742	0.1906	0.1985	0.4183	0.3074
		(0.1040)	(0.0987)	(0.9418)	(0.0897)	(**0.0848**)	(0.0994)	(0.1756)
	uECSQ	0.1656	0.1665	0.1700	0.1785	0.4559	0.4179	0.3111
		(0.1423)	(0.1326)	(0.1234)	(0.1128)	(0.1034)	(0.1032)	(0.1795)
	uCWS	0.1657	0.1670	0.1702	0.1769	0.4429	0.4211	0.3089
		(0.1514)	(0.1419)	(0.1323)	(0.1229)	(0.1141)	(0.1032)	(0.1795)
	uPWS	0.1734	0.1741	0.1735	0.1876	0.2804	0.4224	0.3157
		(0.1377)	(0.1278)	(0.1144)	(0.1050)	(**0.0956**)	(0.1032)	(0.1795)
128	uUQ	0.1652	0.1670	0.1734	0.1866	0.1946	0.4189	0.3068
		(0.1218)	(0.1162)	(0.1097)	(0.1036)	(0.0974)	(0.1032)	(0.1795)
	uECSQ	**0.1649**	0.1678	0.1707	0.4311	0.4977	0.4134	0.3084
		(0.1546)	(0.1463)	(0.1353)	(0.1257)	(0.1148)	(0.1071)	(0.1833)
	uCWS	**0.1652**	0.1669	0.1705	0.4485	0.4111	0.4218	0.3111
		(0.1690)	(0.1585)	(0.1480)	(0.1371)	(0.1264)	(0.1071)	(0.1834)
	uPWS	0.1720	0.1737	0.1762	0.3684	0.4418	0.4176	0.3106
		(0.1555)	(0.1441)	(0.1294)	(0.1186)	(**0.1078**)	(0.1071)	(0.1834)
256	uUQ	0.1654	0.1662	0.1706	0.1766	0.1947	0.4165	0.3049
		(0.1409)	(0.1333)	(0.1259)	(0.1183)	(0.1103)	(0.1071)	(0.1834)
	uECSQ	0.1659	0.1673	0.1714	0.4221	0.5349	0.4198	0.3074
		(0.1651)	(0.1557)	(0.1453)	(0.1312)	(0.1202)	(0.1071)	(0.1834)

Table A.11
Results for the Davis dataset. Same notations as in Table A.7.

DeepDTA – DAVIS (0.3223)								
		Quantization +					SLR	
k	**Method**	**Pr: 70**	**Pr: 75**	**Pr: 80**	**Pr: 85**	**Pr: 90**	**q: 32**	**q: 64**
	uCWS	**0.2309**	0.2342	0.2408	0.2544	0.2530	0.3661	0.2994
		(0.0866)	(0.0789)	(0.0733)	(0.0666)	(0.0607)	(0.0956)	(0.1718)
	uPWS	0.4050	0.3990	0.4180	0.3626	0.5045	0.3671	0.2932

Table A.11 (*continued*)

DeepDTA – DAVIS (0.3223)								
		Quantization +					SLR	
k	**Method**	**Pr: 70**	**Pr: 75**	**Pr: 80**	**Pr: 85**	**Pr: 90**	**q: 32**	**q: 64**
		(0.0702)	(0.0642)	(0.0584)	(0.0528)	(**0.0486**)	(0.0956)	(0.1718)
32	uUQ	0.2331	0.2376	0.2397	0.2482	0.2801	0.3634	0.2853
		(0.0644)	(0.0633)	(0.0603)	(0.0575)	(0.0544)	(0.0956)	(0.1718)
	uECSQ	0.2310	0.2402	0.2441	0.2441	0.2552	0.3682	0.2915
		(0.0901)	(0.0771)	(0.0712)	(0.0634)	(0.0564)	(0.1032)	(0.1795)
	uCWS	0.2386	0.2346	0.2372	0.2524	0.2603	0.3662	0.2957
		(0.0995)	(0.0920)	(0.0844)	(0.0663)	(0.0590)	(0.0994)	(0.1756)
	uPWS	0.2539	0.2552	0.2610	0.2842	0.3271	0.3682	0.2809
		(0.0825)	(0.0747)	(0.0679)	(0.0610)	(**0.0549**)	(0.0994)	(0.1756)
64	uUQ	**0.2344**	0.2369	0.2410	0.2525	0.2584	0.3655	0.2957
		(0.0748)	(0.0719)	(0.0685)	(0.0645)	(0.0597)	(0.0994)	(0.1756)
	uECSQ	0.2364	0.2348	0.2356	0.2414	0.2467	0.3621	0.2831
		(0.0949)	(0.0866)	(0.0805)	(0.0714)	(0.0632)	(0.1071)	(0.1833)
	uCWS	0.2311	0.2301	0.2440	0.2412	0.2525	0.3612	0.2808
		(0.1114)	(0.1026)	(0.0791)	(0.0705)	(0.0598)	(0.1032)	(0.1795)
	uPWS	0.2535	0.2474	0.2538	0.2637	0.2755	0.3615	0.2783
		(0.0949)	(0.0857)	(0.0772)	(0.0690)	(**0.0576**)	(0.1032)	(0.1795)
128	uUQ	**0.2295**	0.2305	0.2342	0.2423	0.2636	0.3696	0.2946
		(0.0860)	(0.0818)	(0.0763)	(0.0716)	(0.0667)	(0.1032)	(0.1795)
	uECSQ	0.2343	0.2335	0.2355	0.2377	0.2570	0.3661	0.2945
		(0.1031)	(0.0934)	(0.0898)	(0.0794)	(0.0698)	(0.1071)	(0.1833)
	uCWS	**0.2267**	0.2273	0.2346	0.2411	0.2718	0.3677	0.2957
		(0.1252)	(0.1145)	(0.0862)	(0.0770)	(0.0644)	(0.1071)	(0.1834)
	uPWS	0.2397	0.2390	0.2440	0.2460	0.2581	0.3680	0.2808
		(0.1071)	(0.0965)	(0.0865)	(0.0770)	(**0.0642**)	(0.1071)	(0.1834)
256	uUQ	0.2305	0.2391	0.2393	0.2419	0.2505	0.3600	0.2811
		(0.0943)	(0.0882)	(0.0822)	(0.0760)	(0.0698)	(0.1071)	(0.1834)
	uECSQ	0.2291	0.2286	0.2329	0.2419	0.2663	0.3604	0.2809
		(0.1103)	(0.1020)	(0.0995)	(0.0876)	(0.0765)	(0.1071)	(0.1834)

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Inf. Process. Syst., 2012, pp. 1097–1105.

[2] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings, 2015.

[3] C. Raffel, N. Shazeer, A. Roberts, et al., Exploring the limits of transfer learning with a unified text-to-text transformer, CoRR abs/1910.10683. arXiv:1910.10683.

[4] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.

[5] S. Ruder, Neural transfer learning for natural language processing, NUI Galway, 2019, Ph.D. thesis.

[6] Z. Allen-Zhu, Y. Li, Y. Liang, Learning and generalization in overparameterized neural networks, going beyond two layers, in: H. Wallach, H. Larochelle, A. Beygelzimer, et al. (Eds.), Advances in Neural Information Processing Systems, vol. 32, Curran Associates Inc, 2019.

[7] J. Yang, W. Xiao, C. Jiang, et al., Ai-powered green cloud and data center, IEEE Access 7 (2019) 4195–4203, https://doi.org/10.1109/ACCESS.2018.2888976.

[8] E. Commission, C. Directorate-General for Communications Networks, Technology, The Assessment List for Trustworthy Artificial Intelligence (ALTAI) for self assessment, Publications Office, 2020. doi:doi/10.2759/791819.

[9] M. Zhang, F. Zhang, N.D. Lane, et al., Deep Learning in the Era of Edge Computing: Challenges and Opportunities, John Wiley & Sons, Ltd, 2020, Ch. 3, pp. 67–78. doi:https://doi.org/10.1002/9781119551713.ch3.

[10] P. Ferragina, G. Vinciguerra, The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds, PVLDB 13 (8) (2020) 1162–1175, https://doi.org/10.14778/3389133.3389135.

[11] M. Sandler, et al., Mobilenetv 2: Inverted residuals and linear bottlenecks, in: Proc. IEEE Conf. on Comput. Vision and Pattern Recognit., 2018, pp. 4510–4520.

[12] W. Dong, J. Wu, Z. Bai, et al., Mobilegcn applied to low-dimensional node feature learning, Pattern Recogn. 112 (2021), https://doi.org/10.1016/j.patcog.2020.107788.

[13] B. Neyshabur, Z. Li, S. Bhojanapalli, et al., The role of over-parametrization in generalization of neural networks, in: 7th International Conference on Learning Representations, ICLR 2019; Conference date: 06–05-2019 Through 09–05-2019, 2019.

[14] J. Ba, R. Caruana, Do deep nets really need to be deep?, Advances in Neural Inf Process. Syst, vol. 27, Curran Associates Inc, 2014.

[15] M.C. Mozer, P. Smolensky, Skeletonization: A technique for trimming the fat from a network via relevance assessment, in: D. Touretzky (Ed.), Advances in Neural Information Processing Systems, vol. 1, Morgan-Kaufmann, 1989.

[16] L. Deng et al., Model compression and hardware acceleration for neural networks: A comprehensive survey, Proc. IEEE 108 (4) (2020) 485–532.

[17] Y. Cheng, et al., A survey of model compression and acceleration for deep neural networks, arXiv preprint arXiv:1710.09282.

[18] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, Advances in neural information processing systems 2.

[19] M. Hagiwara, Removal of hidden units and weights for back propagation networks, in: Proc. of 1993 Int. Conf. on Neural Net. (IJCNN-93-Nagoya, Japan), vol. 1, 1993, pp. 351–354. doi:10.1109/IJCNN.1993.713929.

[20] A.S. Weigend, D.E. Rumelhart, B.A. Huberman, Generalization by weight-elimination with application to forecasting, in: Proc. of the 1990 Conf. on Advances in Neural Inf. Process. Syst., 1990, p. 875–882.

[21] D. Whitley, T. Starkweather, C. Bogart, Genetic algorithms and neural networks: optimizing connections and connectivity, Parallel Comput. 14 (3) (1990) 347–361, https://doi.org/10.1016/0167-8191(90)90086-O.

[22] J. Tu, Y. Zhan, F. Han, A neural network pruning method optimized with pso algorithm, in: 2010 Second Int. Conf. on Comput. Model. and Simul., vol. 3, 2010, pp. 257–259. doi:10.1109/ICCMS.2010.424.

[23] J. Su, N.J. Fraser, G. Gambardella, et al., Accuracy to throughput trade-offs for reduced precision neural networks on reconfigurable logic, in: N. Voros, M. Huebner, G. Keramidas, et al. (Eds.), Applied Reconfigurable Computing. Architectures, Tools, and Applications, Springer International Publishing, Cham, 2018, pp. 29–42.

[24] I. Hubara et al., Binarized neural networks, in: D. Lee, M. Sugiyama, U. Luxburg, et al. (Eds.), Advances in Neural Inf. Process. Syst., Curran Associates Inc, 2016.

[25] B. Jacob, et al., Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proc. of the IEEE Conf. on Comput. Vision and Pattern Recognition (CVPR), 2018, pp. 2704–2713.

[26] E. Park, S. Yoo, P. Vajda, Value-aware quantization for training and inference of neural networks, in: Proc. of the Eur. Conf. on Comput. Vision (ECCV), 2018, pp. 580–595.

[27] L. Hou, Q. Yao, J.T. Kwok, Loss-aware binarization of deep networks, in: 5th Int. Conf. on Learn. Representations, ICLR 2017, Toulon, France, April 24-26, 2017, OpenReview.net, 2017.

[28] A. Zhou, et al., Incremental network quantization: Towards lossless CNNs with low-precision weights, in: 5th Int. Conf. on Learn. Representations, ICLR 2017, Toulon, France, April 24-26, 2017, OpenReview.net, 2017.

[29] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding, in: Y. Bengio, Y. LeCun (Eds.), 4th International Conference on Learning Representations,

ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings, 2016.
[30] G.C. Marinó et al., Compression strategies and space-conscious representations for deep neural networks, 2020 25th Int. Conf. on Pattern Recognition (ICPR) 2021 (2020) 9835–9842, https://doi.org/10.1109/ICPR48806.2021.9412209.
[31] G.C. Marinò et al., Reproducing the sparse huffman address map compression for deep neural networks, in: Reproducible Research in Pattern Recognition, Springer International Publishing, Cham, 2021, pp. 161–166.
[32] Y. Choi, M. El-Khamy, J. Lee, Universal deep neural network compression, IEEE J. Sel. Topics Signal Process. 14 (4) (2020) 715–726.
[33] A. Gersho, R.M. Gray, Vector Quantization and Signal Compression, Kluwer Academic Publishers, USA, 1991.
[34] J. Xue, J. Li, Y. Gong, Restructuring of deep neural network acoustic models with singular value decomposition, in: Interspeech, interspeech Edition, 2013, pp. 2365–2369.
[35] T.N. Sainath, et al., Low-rank matrix factorization for deep neural network training with high-dimensional output targets, in: Proc. IEEE Int. Conf. on Acoust., Speech and Signal Proc., 2013, pp. 6655–6659.
[36] L. De Lathauwer, Decompositions of a higher-order tensor in block terms – part i: Lemmas for partitioned matrices, SIAM J. Matrix Anal. Appl. 30 (3) (2008) 1022–1032.
[37] R. Rigamonti, et al., Learning separable filters, in: 2013 IEEE Conf. on Comput. Vision and Pattern Recognition, 2013, pp. 2754–2761. doi:10.1109/CVPR.2013.355.
[38] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, CoRR abs/1405.3866. arXiv:1405.3866.
[39] X. Yu, et al., On compressing deep models by low rank and sparse decomposition, in: Proc. of the IEEE Conf. on Comput. Vision and Pattern Recognition (CVPR), 2017, pp. 7370–7379.
[40] R. Müller, S. Kornblith, G.E. Hinton, When does label smoothing help?, Advances in Neural Inf Proc. Syst., vol. 32, Curran Associates Inc, 2019.
[41] Y. Tian, D. Krishnan, P. Isola, Contrastive representation distillation, in: Int. Conf. on Learn. Representations, 2020.
[42] A. Mallya, S. Lazebnik, Packnet: Adding multiple tasks to a single network by iterative pruning, in: 2018 IEEE Conf. on Comput. Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, IEEE Computer Society, 2018, pp. 7765–7773. doi:10.1109/CVPR.2018.00810.
[43] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, in: Advances in Neural Inf. Proc. Syst., vol. 2, Morgan-Kaufmann, 1990.
[44] P. Molchanov, A. Mallya, S. Tyree, et al., Importance estimation for neural network pruning, in: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16–20, 2019, Computer Vision Foundation/ IEEE, 2019, pp. 11264–11272. doi:10.1109/CVPR.2019.01152.
[45] P. Molchanov, et al., Pruning convolutional neural networks for resource efficient transfer learning, CoRR abs/1611.06440. arXiv:1611.06440.
[46] H.-G. Han, J.-F. Qiao, A structure optimisation algorithm for feedforward neural network construction, Neurocomput. 99 (2013) 347–357, https://doi.org/10.1016/j.neucom.2012.07.023.
[47] H. Li, et al., Pruning filters for efficient convnets, arXiv preprint arXiv:1608.08710 arXiv:1608.08710.
[48] J.-H. Luo et al., Thinet: Pruning cnn filters for a thinner net, IEEE Trans. Pattern Anal. Mach. Intell. 41 (10) (2019) 2525–2538, https://doi.org/10.1109/TPAMI.2018.2858232.
[49] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: 2017 IEEE Int. Conf. on Comput. Vision (ICCV), 2017, pp. 1398–1406. doi:10.1109/ICCV.2017.155.
[50] X. He, Z. Zhou, L. Thiele, Multi-task zipping via layer-wise neuron sharing, Advances in Neural Inf. Proc. Syst., vol. 31, Curran Associates Inc, 2018.
[51] Y. Idelbayev, M.A. Carreira-Perpiñán, Lc: A flexible, extensible open-source toolkit for model compression, in: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, Association for Computing Machinery, New York, NY, USA, 2021.
[52] M. Á. Carreira-Perpiñán, Y. Idelbayev, Model compression as constrained optimization, with application to neural nets. part V: combining compressions, CoRR abs/2107.04380. arXiv:2107.04380.
[53] M. Courbariaux et al., Binaryconnect: Training deep neural networks with binary weights during propagations, in: Advances in Neural Inf. Proc. Syst. 28, Curran Associates Inc., 2015, pp. 3123–3131.
[54] L. Deng et al., Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework, Neural Networks 100 (2018) 49–58.
[55] H. Gish, J. Pierce, Asymptotically efficient quantizing, IEEE Trans. Inf. Theory 14 (5) (1968) 676–683.
[56] P.A. Chou, T. Lookabaugh, R.M. Gray, Entropy-constrained vector quantization, IEEE Trans. Acoust. Speech Signal Process. 37 (1) (1989) 31–42.
[57] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., Society for Industrial and Applied Mathematics, USA, 2003.
[58] S. Swaminathan, D. Garg, R. Kannan, et al., Sparse low rank factorization for deep neural network compression, Neurocomputing 398 (2020) 185–196, https://doi.org/10.1016/j.neucom.2020.02.035.
[59] Y. Lecun et al., Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[60] A. Krizhevsky, Learning multiple layers of features from tiny images, University of Toronto, 2009, Master's thesis,.
[61] A. Krizhevsky, Learning multiple layers of features from tiny images (Tech. rep.), University of Toronto, 2009.
[62] M.I. Davis et al., Comprehensive analysis of kinase inhibitor selectivity, Nat. Biotechnol. 29 (2011) 1046–1051.
[63] J. Tang et al., Making sense of large-scale kinase inhibitor bioactivity data sets: A comparative and integrative analysis, J. Chem. Inf. Model. 54 (3) (2014) 735–743, https://doi.org/10.1021/ci400709d.
[64] H. Öztürk et al., DeepDTA: deep drug–target binding affinity prediction, Bioinformatics 34 (17) (2018) i821–i829, https://doi.org/10.1093/bioinformatics/bty593.
[65] E. Denton, W. Zaremba, J. Bruna, et al., Exploiting linear structure within convolutional networks for efficient evaluation, in: Proceedings of the 27th International Conference on Neural Information Processing Systems – Volume 1, NIPS'14, MIT Press, Cambridge, MA, USA, 2014, p. 1269–1277.

Giosuè Cataldo Marinò received the B.Sc. degree in computer science from the Università degli Studi di Milano, Italy, where he is currently pursuing the master's degree in computer science. His research interests include machine learning and compression of neural network models.

Alessandro Petrini received a B.Sc. in Applied Mathematics, and a M.Sc and a Ph.D. in Computer Science from the Università degli Studi di Milano, Italy, in 2014, 2017 and 2021 respectively. His main research interest is focused in Parallel and Accelerated Computing, but during his career he contributed in several fields of scientific research such as Machine Learning, Deep Learning, Bioinformatics, Image/Video Analysis and Processing, and data compression. As a part of his continuous pursuit of the forefront of technical innovation, he is now a Senior Software Engineer contributing to Web 3.0 research and developing blockchain based applications.

Dario Malchiodi received the M.Sc. degree in computing and the Ph.D. degree in computational mathematics and operations research from the Università degli Studi di Milano, Italy, in 1997 and 2000, respectively. Since 2002, he has been an Assistant Professor with the Department of Computer Science, Universit? degli Studi di Milano, where he was appointed as an Associate Professor, in 2011. He teaches statistics and data analysis and algorithms for massive datasets. He is the author of about 100 scientific publications. He is also actively involved in the popularization of computing. His research interests include the treatment of uncertainty in machine learning, with a particular focus to data-driven induction of fuzzy sets, compression of machine learning models, mining of knowledge bases in semantic web, negative example selection in bioinformatics, and application of machine learning to the medical, veterinary, and forensics fields.

Marco Frasca received the Ph.D. degree in computer science from the Università degli Studi di Milano, Italy, in 2012. He was a Postdoctoral Researcher with the Department of Biosciences and the Department of Computer Science, Università degli Studi di Milano. Since 2017, he has been an Assistant Professor at the Department of Computer Science, of the same university, where he is a member of AnacletoLab, whose research activities regard the field of machine learning applied in biology and medicine. He has been an Invited Research Visitor at several universities, including the Terrence Donnelly Centre for Cellular and Biomolecular Research, University of Toronto, and the Institute of Molecular Biology, Johannes Gutenberg University of Mainz. He contributed to consolidate the application of Hopfield networks to classification and ranking problems with the development of single- and multi-task parametric Hopfield models. His research interest includes the design and analysis of new machine learning methods, with applications in bioinformatics, computational biology, and medicine.