Compact and Computationally Efficient Representation of Deep Neural Networks

Simon Wiedemann, Klaus-Robert Müller, *Member, IEEE*, and Wojciech Samek, *Member, IEEE*

***Abstract*— At the core of any inference procedure, deep neural networks are dot product operations, which are the component that requires the highest computational resources. For instance, deep neural networks, such as VGG-16, require up to 15-G operations in order to perform the dot products present in a single forward pass, which results in significant energy consumption and thus limits their use in resource-limited environments, e.g., on embedded devices or smartphones. One common approach to reduce the complexity of the inference is to prune and quantize the weight matrices of the neural network. Usually, this results in matrices whose entropy values are low, as measured relative to the empirical probability mass distribution of its elements. In order to efficiently exploit such matrices, one usually relies on, inter alia, sparse matrix representations. However, most of these common matrix storage formats make strong statistical assumptions about the distribution of the elements; therefore, cannot efficiently represent the entire set of matrices that exhibit low-entropy statistics (thus, the entire set of compressed neural network weight matrices). In this paper, we address this issue and present new efficient representations for matrices with low-entropy statistics. Alike sparse matrix data structures, these formats exploit the statistical properties of the data in order to reduce the size and execution complexity. Moreover, we show that the proposed data structures can not only be regarded as a generalization of sparse formats but are also more energy and time efficient under practically relevant assumptions. Finally, we test the storage requirements and execution performance of the proposed formats on compressed neural networks and compare them to dense and sparse representations. We experimentally show that we are able to attain up to ×42 compression ratios, ×5 speed ups, and ×90 energy savings when we *lossless* convert the state-of-the-art networks, such as AlexNet, VGG-16, ResNet152, and DenseNet, into the new data structures and benchmark their respective dot product.**

***Index Terms*— Computationally efficient deep learning, data structures, lossless coding, neural network compression, sparse matrices.**

I. Introduction

THE dot product operation between matrices constitutes one of the core operations in almost any field in science. Examples are the computation of approximate solutions of complex system behaviors in physics [1], iterative solvers in mathematics [2], and features in computer vision applications [3]. In addition, deep neural networks heavily rely on dot product operations in their inference [4], e.g., networks, such as VGG-16, require up to 16 dot product operations, which results in 15-G operations for a single forward pass. Hence, lowering the algorithmic complexity of these operations and thus increasing their efficiency is of major interest for many modern applications. Since the complexity depends on the data structure used for representing the elements of the matrices, a great amount of research has focused on designing data structures and respective algorithms that can perform efficient dot product operations [5]–[7].

Of particular interest are the so-called *sparse matrices*, a special type of matrices that have the property that many of their elements are zero-valued. In principle, one can design efficient representations of sparse matrices by leveraging the prior assumption that most of their element values are zero and, therefore, only store the nonzero entries of the matrix. Consequently, their storage requirements become of the order of the number of nonzero values. However, having an efficient representation with regard to storage requirement does not imply that the dot product algorithm associated with that data structure will also be efficient. Hence, a great part of the research was focused on the design of data structures that have as well low-complex dot product algorithms [7]–[10]. However, by assuming sparsity alone, we are implicitly imposing a spike-and-slab prior[1] over the probability mass distribution of the elements of the matrix. If the actual distribution of the elements greatly differs from this assumption, then the data structures become inefficient. Hence, sparsity can be a too constrained assumption for the representation of quantized neural networks.

In this paper, we alleviate the shortcomings of sparse representations by considering a more relaxed prior to the

[1]That is, a delta function at 0 and a uniform distribution elsewhere.

Manuscript received May 27, 2018; revised December 1, 2018, March 13, 2019 and April 3, 2019; accepted April 5, 2019. Date of publication May 29, 2019; date of current version February 28, 2020. This work was supported in part by the Fraunhofer Society through the MPI-FhG collaboration project "Theory and Practice for Reduced Learning Machines," in part by the German Ministry for Education through the Berlin Big Data Center under Grant 01IS14013A, in part by the Berlin Center for Machine Learning under Grant 01IS18037I, in part by DFG (EXC 2046/1) under Grant 390685689, and in part by the Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government under Grant 2017-0-00451. *(Corresponding authors: Klaus-Robert Müller; Wojciech Samek.)*

S. Wiedemann and W. Samek are with the Fraunhofer Heinrich Hertz Institute, 10587 Berlin, Germany (e-mail: wojciech.samek@hhi.fraunhofer.de).

K.-R. Müller is with Technische Universität Berlin, 10587 Berlin, Germany, also with the Max Planck Institute for Informatics, 66123 Saarbrücken, Germany, and also with the Department of Brain and Cognitive Engineering, Korea University, Seoul 136-713, South Korea (e-mail: klaus-robert.mueller@tu-berlin.de).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2910073

distribution of the matrix elements. More precisely, we assume that the empirical probability mass distribution of the matrix elements has a low-entropy value, as defined by Shannon [11]. Mathematically, sparsity can be considered a subclass of the general family of low entropic distributions. In fact, sparsity measures the min-entropy of the element distribution, which is related to Shannon's entropy measure through Renyi's generalized entropy definition [12]. With this goal in mind, we ask the question:

> *"Can we devise efficient data structures under the implicit assumption that the entropy of the distribution of the matrix elements is low?"*

We want to stress once more that by efficiency we regard two related but distinct aspects: 1) efficiency with regard to storage requirements and 2) efficiency with regard to algorithmic complexity of the dot product associated with the representation. For the later, we focus on the number of elementary operations required in the algorithm, since they are related to the energy and time complexity of the algorithm. It is well known that the minimal bit length of a data representation is bounded by the entropy of its distribution [11]. Hence, matrices with low entropic distributions automatically imply that we can design data structures that do not require high storage resources. In addition, as we will discuss in the following, low entropic distributions also attain gains in efficiency if these data structures implicitly encode the distributive law of multiplications. By doing so, a great part of the algorithmic complexity of the dot product is reduced to the order of the number of shared weights per row in a matrix. This number is related to the entropy, such that it is small as long as the entropy of the matrix is low. Therefore, these data structures not only attain higher compression gains but also require less total number of operations when performing the dot product.

Our contributions can be summarized as follows.

1) We propose new highly efficient data structures that exploit on the prior that the matrix has a low number of shared weights per row (i.e., low entropy).
2) We provide a detailed analysis of the storage requirements and algorithmic complexity of performing the dot product associated with these data structures.
3) We establish a relation between the known sparse and the proposed data structures. Namely, sparse matrices belong to the same family of low entropic distributions; however, they can be considered a more constrained subclass of them.
4) We show through experiments that indeed, these data structures attain gains in efficiency on simulated as well as real-world data. In particular, we show that up to $\times 42$ compression ratios, $\times 5$ speed ups, and $\times 90$ energy savings can be achieved when we benchmark the compressed weight matrices of the state-of-the-art neural networks relative to the matrix-vector multiplication.

In Section II, we introduce the problem of efficient representation of neural networks and briefly review related literature. In Section III, the proposed data structures are given. We demonstrate through a simple example that these data structures are able to: 1) achieve higher compression ratios than their respective dense and sparse counterparts and 2) reduce the algorithmic complexity of performing the dot product. Section IV analyses the storage and energy complexity of these novel data structures. Experimental evaluation is performed in Section V using simulations as well as the state-of-the-art neural networks, such as AlexNet, VGG-16, ResNet152, and DenseNet. Section VI concludes this paper with a discussion.

II. Efficient Inference in Neural Networks

Deep neural networks [13], [14] became the state of the art in many fields of machine learning, such as in computer vision, speech recognition, and natural language processing [15]–[18], and have also been progressively used in the sciences, e.g., physics [19], neuroscience [20], and chemistry [21], [22]. In their most basic form, they constitute a chain of affine transformations concatenated with a nonlinear function which is applied elementwise to the output. Hence, the goal is to learn the values of those transformation or weight matrices (i.e., parameters) such that the neural network performs its task particularly well. The procedure of calculating the output prediction of the network for a particular input is called *inference*. The computational cost of performing inference is dominated by computing the affine transformations (thus, the dot products between matrices). Since today's neural networks perform many dot product operations between large matrices, this greatly complicates their deployment onto resource-constrained devices.

However, it has been extensively shown that most neural networks are overparameterized, i.e., there are many more parameters than actually needed for the tasks of interest [23]–[26]. This implies that these networks are highly inefficient with regard to the resources they require when performing inference. This fact motivated an entire research field of model compression [27]. One of the suggested approaches is to: 1) compress the weight elements of the neural network without (considerably) affecting their prediction accuracy and 2) convert the resulting weights into a representation that achieves high compression ratios and is able to execute the dot product operation efficiently. While there has been a plethora of work focusing on the first step [25], [26], [28]–[38], previous literature has not focused as much on the second part. As a consequence, most of the research has focused on developing techniques that either sparsify the network weights [26], [28]–[30] or reduce the cardinality of the weight elements [31]–[33], since then sparse matrix representations or dense matrices with compressed numerical representations can be employed in order to efficiently perform inference.

However, this greatly reduces the possible efficiency gains that can be achieved. In fact, highest reported compression gains are attained with techniques that either implicitly [25], [37] or explicitly [34]–[36], [38], [39] attempt to reduce the entropy of the weight matrices of the network. To recall, throughout this paper, we consider the entropy of the empirical probability mass distribution of the weight elements.[2] With sparse or dense representations,

[2]We identify the set of unique elements Ω in the matrix and for each element $\omega_k \in \Omega$ count it's relative frequency of appearance $p_k = \#(\omega_k)/N$. Finally, we calculate Shannon's entropy $H = -\sum_k p_k \log_2 p_k$.

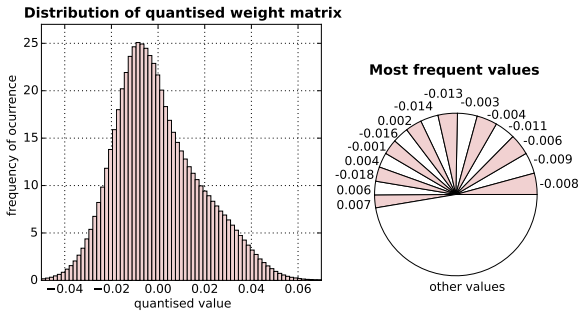

Fig. 1. Distribution of the weight matrix of the last layer of VGG-16 after uniform quantization over the range of values, with 2^7 quantization points, which resulted in no loss of accuracy. Left: distribution of quantized weights. Right: frequency of appearance of the 15 most frequent values.

i.e., formats which are not specifically designed for representing low-entropy weight matrices, the theoretically possible efficiency gains may not be achieved.

For instance, Fig. 1 shows the distribution of the weight elements of the last classification layer of VGG-16 [40] (1000×4096 dimensional matrix), after having applied uniform quantization on the weight elements. We stress that the prediction accuracy and generalization of the network were not affected by this operation. As we can see, the distribution of the compressed layer does not satisfy the sparsity assumption, i.e., there is not one particular element (such as 0) that appears specially frequent in the matrix. The most frequent value is -0.008 and its frequency of appearance does not dominate over the others (about 4.2%). Thus, although this matrix has low entropy, it is not sparse enough to be efficiently represented by sparse matrix formats. Alternatively, one could trivially reduce the numerical precision [31], [32], [41], [42] needed for representing each weight element value down to 7 bits in this case. This greatly reduces the storage requirements without affecting the accuracy. However, if the activation values remained unquantized, then the associated dot product algorithm would require multiple, mostly expensive decoding operations in order to convert back each weight element to its original 32-bit floating point value, increasing as such by at least $\times 2$ the workload. If, on the other hand, the activation values were quantized and reduced to a 7-bit representation, then we could apply similar optimization techniques as in [43] and [44] in order to reduce both the memory and algorithmic complexity for performing inference. However, the accuracy of most networks we considered in this paper is harmed by 1%–3% (see the Supplementary Material I-B) in this case. Since the main focus of this paper is on lossless encoding of neural networks, we do not consider these types of representations here. However, we acknowledge that nonlinear quantization methods as well as low-precision representations, especially in combination with retraining [45]–[49] may constitute a good alternative to dedicated matrix representations in practice.

In this paper, we present new lossless matrix representations that become more efficient as the entropy of the weight matrices is reduced. In particular, their complexity depends partially on the number of shared weights present in the matrix, which is reduced as the entropy of the matrix is reduced. Indeed, we note that for the matrix in Fig. 1, most of the entries are dominated by only 15 distinct values, which is 1.5% of the number of columns of the matrix. In Section III, we will describe with a simple example how these new representations leverage on this property in order to achieve both high compression ratios and efficient dot products.

III. Data Structures for Matrices With Low-Entropy Statistics

In this section, we introduce the proposed data structures and show that they implicitly encode the distributive law. Consider the following matrix:

$$M = \begin{pmatrix} 0 & 3 & 0 & 2 & 4 & 0 & 0 & 2 & 3 & 4 & 0 & 4 \\ 4 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 4 & 4 & 0 & 4 \\ 4 & 0 & 3 & 4 & 0 & 0 & 0 & 4 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 & 4 & 0 & 3 & 4 & 4 & 0 & 0 \\ 0 & 4 & 4 & 0 & 0 & 4 & 0 & 4 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Now assume that we want to: 1) store this matrix with the minimum amount of bits and 2) perform the dot product with a vector $a \in \mathbb{R}^{12}$ with the minimum complexity.

A. Minimum Storage

We firstly comment on the storage requirement of dense and sparse formats and then introduce two new formats which more effectively store matrix M.

1) Dense Format: Arguably, the simplest way to store the matrix M is in its so-called dense representation. That is, we store its elements in a 5×12 long array (in addition to its dimensions $m = 5$ and $n = 12$).

2) Sparse Format: However, note that more than 50% of the entries are 0. Hence, we may be able to attain a more compressed representation of this matrix if we store it in one of the well-known sparse data structures, for instance, in the *compressed sparse row* (CSR) format. This particular format stores the values of the matrix in the following way.

1) Scans the nonzero elements in row-major order (that is, from left to right and up to down) and stores them in an array (which we denote as W).
2) Simultaneously, it stores the respective column indices in another array (which we call colI).
3) Finally, it stores pointers that signal when a new row starts (we denote this array as rowPtr).

Hence, our matrix M would take the form

$$\begin{aligned} W &: [3, 2, 4, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, \\ &\quad 4, 4, 2, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4] \\ colI &: [1, 3, 4, 7, 8, 9, 11, 0, 1, 5, 8, 9, 11, 0, \\ &\quad 2, 3, 7, 9, 3, 4, 5, 7, 8, 9, 1, 2, 5, 7] \\ rowPtr &: [0, 7, 13, 18, 24, 28]. \end{aligned}$$

If we assume the same bit size per element for all arrays, then the CSR data structure does not attain higher compression gains in spite of not saving the zero-valued elements (62 entries versus 60 that are being required by the dense data structure).

We can improve this by exploiting the low-entropy property of matrix M. In the following, we propose two new formats which realize this.

3) Compressed Entropy Row Format: First, note that many elements in M share the same value. In fact, only the four values $\Omega = \{0, 4, 3, 2\}$ appear in the entire matrix. Hence, it appears reasonable to assume that data structures that repeatedly store these values (such as the dense or CSR structures) induce high redundancies in their representation. Therefore, we propose a data structure where we only store those values once. Second, note that different elements appear more frequently than others, and their relative order does not change throughout the rows of the matrix. Concretely, we have a set of unique elements $\Omega = \{0, 4, 3, 2\}$ which appear $P_{\#} = \{32, 21, 4, 3\}$ times, respectively, in the matrix, and we obtain the same relative order of highest to lowest frequent value throughout the rows of the matrix. Hence, we can design an efficient data structure which leverages on both properties in the following way.

1) Store unique elements present in the matrix in an array in frequency-major order (that is, from most to least frequent). We name this array Ω.
2) Store, respectively, the column indices in row-major order, excluding the first element (thus excluding the most frequent element). We denote it as colI.
3) Store pointers that signal when the positions of the next new element in Ω start. We name it ΩPtr. If a particular pointer in ΩPtr is the same as the previous one, this means that the current element is not present in the matrix and we jump to the next element.
4) Store pointers that signal when a new row starts. We name it rowPtr. Here, rowPtr points to entries in ΩPtr.

Hence, this new data structure represents matrix M as

$$\begin{aligned}
&\Omega : [0, 4, 3, 2]\\
&colI : [4, 9, 11, 1, 8, 3, 7, 0, 1, 5, 8, 9, 11, 0,\\
&\qquad 3, 7, 2, 9, 3, 4, 5, 8, 9, 7, 1, 2, 5, 7]\\
&\Omega Ptr : [0, 3, 5, 7, 13, 16, 17, 18, 23, 24, 28]\\
&rowPtr : [0, 3, 4, 7, 9, 10].
\end{aligned}$$

Note that we can uniquely reconstruct M from this data structure. We refer to this data structure as the *compressed entropy row* (CER) data structure. One can verify that indeed, the CER data structure only requires 49 entries (instead of 60 or 62) attaining as such a compressed representation of the matrix M.

To summarize, the CER representation is able to attain higher compression gains because it leverages on the following two properties: 1) many matrix elements share the same value and 2) the empirical probability mass distribution of the shared weight elements does not change significantly across rows.

4) Compressed Shared Elements Row Format: In some cases, it may be well that the probability distribution across rows in a matrix is not similar to each other. Hence, the second assumption in the CER data structure would not apply and we would only be left with the first one. That is, we only know that not many distinct elements appear per row in the matrix or, in other words, that many elements share the same value. The *compressed shared elements row* (CSER) data structure is a slight extension to the previous CER representation. Here, we add an element pointer array, which signals which element in Ω the colI indices refer to. We called it ΩI. Thus, ΩI points to entries in Ω, ΩPtr to entries in colI, and rowPtr to entries in ΩPtr. Hence, the above matrix would then be represented as follows:

$$\begin{aligned}
&\Omega : [0, 2, 3, 4]\\
&colI : [4, 9, 11, 1, 8, 3, 7, 0, 1, 5, 8, 9, 11, 0,\\
&\qquad 3, 7, 2, 9, 3, 4, 5, 8, 9, 7, 1, 2, 5, 7]\\
&\Omega I : [3, 2, 1, 3, 3, 2, 1, 3, 2, 3]\\
&\Omega Ptr : [0, 3, 5, 7, 13, 16, 17, 18, 23, 24, 28]\\
&rowPtr : [0, 3, 4, 7, 9, 10].
\end{aligned}$$

Thus, for storing matrix M, we require 59 entries, which is still a gain but not a significant one. Note that, now the ordering of the elements in Ω is not important anymore, as long as the ΩI array is accordingly adjusted. Similarly, the ordering of ΩI at each row can also be arbitrary, as long as the ΩPtr and colI arrays are accordingly adjusted.

The relationship between CSER, CER, and CSR data structures is described in Section IV.

B. Dot Product Complexity

We just saw that we can attain gains with regard to compression if we represent the matrix in the CER and CSER data structures. However, we can also devise corresponding dot product algorithms that are more efficient than their dense and sparse counterparts. As an example, consider the scalar product between the second row of matrix M with an arbitrary input vector $a = [a_1\ a_2\ \ldots\ a_{12}]^\top$. In principle, the difference in the algorithmic complexity arises because each data structure implicitly encodes a different expression of the scalar product, namely

$$\begin{aligned}
dense:\ & 4a_1 + 4a_2 + 0a_3 + 0a_4 + 0a_5 + 4a_6\\
&+ 0a_7 + 0a_8 + 4a_9 + 4a_{10} + 0a_{11} + 4a_{12}\\
CSR:\ & 4a_1 + 4a_2 + 4a_6 + 4a_9 + 4a_{10} + 4a_{12}\\
CER/CSER:\ & 4(a_1 + a_2 + a_6 + a_9 + a_{10} + a_{12}).
\end{aligned}$$

For instance, the dot product algorithm associated with the dense format would calculate the above scalar product by: 1) loading M and a and 2) calculating $4a_0+4a_1+0a_2+0a_3+0a_4+4a_5+0a_6+0a_7+4a_8+4a_9+0a_{10}+4a_{11}$. This requires 24 load (12 for the matrix elements and 12 for the input vector elements), 12 multiply, 11 add, and 1 write operations (for writing the result into memory). We purposely omitted the accumulate operation, which stores the intermediate values of the multiply-sum operations, since their cost can effectively be associated with the sum operation. Moreover, we only considered read/write operations from and into memory. Hence, this makes 48 operations in total.

In contrast, the dot product algorithm associated with the CSR representation would only multiply-add the nonzero entries. It does so by performing the following steps.

1) Load the subset of rowPtr respective to row 2. Thus, $rowPtr \rightarrow [7, 13]$.
2) Then, load the respective subset of nonzero elements and column indices. Thus, $W \rightarrow [4, 4, 4, 4, 4, 4]$ and $colI \rightarrow [0, 1, 5, 8, 9, 11]$.
3) Finally, load the subset of elements of a respective to the loaded subset of column indices and subsequently multiply-add them to the loaded subset of W. Thus, $a \rightarrow [a_0, a_1, a_5, a_8, a_{10}, a_{11}]$ and calculate $4a_0 + 4a_1 + 4a_5 + 4a_8 + 4a_9 + 4a_{11}$.

By executing this algorithm, we would require 20 load operations (2 from rowPtr and 6 from W, colI, and the input vector, respectively), six multiplications, five additions, and one write. In total, this dot product algorithm requires 32 operations.

However, we can still see that the above dot product algorithm is inefficient in this case since we constantly multiply by the same element 4. Instead, the dot product algorithm associated with, e.g., the CER data structure would perform the following steps.

1) Load the subset of rowPtr respective to row 2. Thus, rowPtr $\rightarrow [3, 4]$.
2) Load the corresponding subset in ΩPtr. Thus, ΩPtr $\rightarrow [7, 13]$.
3) For each pair of elements in ΩPtr, load the respective subset in colI and the element in Ω. Thus, $\Omega \rightarrow [4]$ and $colI \rightarrow [0, 1, 5, 8, 9, 11]$.
4) For each loaded subset of colI, perform the sum of the elements of a respective to the loaded colI. Thus, $a \rightarrow [a_0, a_1, a_5, a_8, a_{10}, a_{11}]$ and do $a_0 + a_1 + a_5 + a_8 + a_9 + a_{11} = z$.
5) Subsequently, multiply the sum with the respective element in Ω. Thus, compute $4z$.

A similar algorithm can be devised for the CSER data structure. One can find both pseudocodes in the Supplementary Material I-C. The operations required by this algorithm are 17 load operations (2 from rowPtr, 2 from ΩPtr, 1 from Ω, 6 from colI, and 6 from a), one multiplication, five additions, and one write. In total, these are 24 operations.

Hence, we have observed that for the matrix M, the CER (and CSER) data structure does not only achieve higher compression rates but also attains gains in efficiency with respect to the dot product operation.

In Section IV, we give a detailed analysis of the storage requirements needed by the data structures and also the efficiency of the dot product algorithm associated with them. This will help us identify when one type of data structure will attain higher gains than the others.

IV. Analysis of the Storage and Energy Complexity of Data Structures

Without loss of generality, in the following, we assume that we aim to encode a particular matrix $M \in \Omega^{n \times m = N}$, where its elements $M_{ij} = \omega_k \in \Omega$ take values from a finite set of elements $\Omega = \{\omega_0, \omega_1, \ldots, \omega_{K-1}\}$. Moreover, we assign to each element ω_k a probability mass value $p_k = \#(\omega_k)/N$, where $\#(\omega_k)$ counts the number of times the element ω_k appears in the matrix M. We denote the respective set of probability mass values $P_\Omega = \{p_0, p_1, \ldots, p_{K-1}\}$. In addition, we assume that each element in Ω appears at least once in the matrix (thus, $p_k > 0$ for all $k = 0, \ldots, K-1$) and that $\omega_0 = 0$ is the most frequent value in the matrix. Finally, we order the elements in Ω and P_Ω in probability-major order, that is, $p_0 \geq p_1 \geq \cdots \geq p_{K-1}$.

A. Measuring the Energy Efficiency of the Dot Product

This paper proposes representations that are efficient with regard to storage requirements as well as their dot product algorithmic complexity. For the latter, we focus on the energy requirements, since we consider it as the most relevant measures for neural network compression. However, exactly measuring the energy of an algorithm is unreliable since it depends on the software implementation and on the hardware the program is running on. Therefore, we will model the energy costs in a way that can easily be adapted across different software implementations as well as hardware architectures.

In the following, we model a dot product algorithm by a computational graph, whose nodes can be labeled with one of the four elementary operations, namely: 1) a *mul* or multiply operation which takes two numbers as input and outputs their multiplied value; 2) a *sum* or summation operation which takes two values as input and outputs their sum; 3) a *read* operation which reads a particular number from memory; and 4) a *write* operation which writes a value into memory. Note that, we do not consider read/write operations from/into low-level memory (like caches and registers) that stores temporary runtime values, e.g., outputs from summation and/or multiplications, since their cost can be associated with those operations. Now, each of these nodes can be associated with an energy cost. Then, the total energy required for a particular dot product algorithm simply equals the total cost of the nodes in the graph.

However, the energy cost of each node depends on the hardware architecture and on the bit size of the values involved in the operation. Hence, in order to make our model flexible with regard to different hardware architectures, we introduce four cost functions σ, μ, γ, and $\delta : \mathbb{N} \rightarrow \mathbb{R}$, which take as input a bit size and output the energy cost of performing the operation associated with them[3]; σ is associated with the *sum* operation, μ with the *mul*, γ with the *read*, and δ with the *write* operation.

A simple dot product algorithm for two 2-D input vectors requires four *read* operations, two *mul*, one *sum*, and one *write*. Assuming that the bit size of all numbers is $b \in \mathbb{N}$, we can state that the energy cost of this dot product algorithm would be $E = 1\sigma(b) + 2\mu(b) + 4\gamma(b) + 1\delta(b)$. Note that similar energy models have been previously proposed [50], [51]. In Section V, we validate the model by comparing it to real energy results measured by previous authors.

Considering this energy model, we can now provide a detailed analysis of the complexity of the CER and CSER data structures. However, we start with a brief analysis of the storage and energy requirements of the dense and sparse data structure in order to facilitate the comparison between them.

[3]The *sum* and *mul* operations take two numbers as input and they may have different bit sizes. Hence, in this case, we take the maximum of those as a reference for the bit sizes involved in the operation.

B. Efficiency Analysis of the Dense and CSR Formats

The dense data structure stores the matrix in an N-long array (where $N = m \times n$) using a constant bit size b_Ω for each element. Therefore, its effective per element storage requirement is

$$S_{\text{dense}} = b_\Omega \tag{1}$$

bits. The associated standard scalar product algorithm then has the following per element energy costs:

$$E_{\text{dense}} = \sigma(b_o) + \mu(b_o) + \gamma(b_a) + \gamma(b_\Omega) + \frac{1}{n}\delta(b_o) \tag{2}$$

where b_a denotes the bit size of the elements of the input vector $a \in \mathbb{R}^n$ and b_o is the bit size of the elements of the output vector. The cost (2) is derived by considering: 1) loading the elements of the input vector $[\gamma(b_a)]$; 2) loading the elements of the matrix $[\gamma(b_\Omega)]$; 3) multiplying them $[\mu(b_o)]$; 4) summing the multiplications $[\sigma(b_o)]$; and 5) writing the result $[\delta(b_o)/n]$. We can see that both the storage and the dot product efficiency have a constant cost attached to them, despite the distribution of the elements of the matrix.

In contrast, the CSR data structure requires only

$$S_{\text{CSR}} = (1 - p_0)(b_\Omega + b_I) + \frac{1}{n}b_I \tag{3}$$

effective bits per element in order to represent the matrix, where b_I denotes the bit size of the column indices. This comes from the fact that we need in total $N(1-p_0)b_\Omega$ bits for representing the nonzero elements of the matrix, $N(1-p_0)b_I$ bits for their respective column indices, and mb_I bits for the row pointers. Moreover, it requires

$$E_{\text{CSR}} = (1 - p_0)(\sigma(b_o) + \mu(b_o) + \gamma(b_a) + \gamma(b_\Omega) + \gamma(b_I)) + \frac{1}{n}\gamma(b_I) + \frac{1}{n}\delta(b_o) \tag{4}$$

units of energy per matrix element in order to perform the dot product. The expression (4) was derived from: 1) loading the nonzero element values $[(1 - p_0)\gamma(b_\Omega)]$, their respective indices $[(1-p_0)\gamma(b_I)+\gamma(b_I)/n]$, and the respective elements of the input vector $[\gamma(b_a)]$; 2) multiplying and summing those elements $[\sigma(b_o) + \mu(b_o)]$; and 3) writing the result into memory $[\delta(b_o)/n]$.

Different from the dense format, the efficiency of the CSR data structure increases as $p_0 \to 1$, and thus, the number of zero elements increases. Moreover, if the matrix size is large enough, the storage requirement and the cost of performing a dot product becomes effectively 0 as $p_0 \to 1$.

For the ease of the analysis, we introduce the big $\mathcal{O}$ notation for capturing terms that depend on the shape of the matrix. In addition, we denote the following set of operations:

$$c_a = \sigma(b_a) + \gamma(b_a) + \gamma(b_I) \tag{5}$$

$$c_\Omega = \gamma(b_I) + \gamma(b_\Omega) + \mu(b_o) + \sigma(b_o) - \sigma(b_a). \tag{6}$$

c_a can be interpreted as the total effective cost of involving an element of the input vector in the dot product operation. Analogously, can c_Ω be interpreted with regard to the elements of the matrix. Hence, we can rewrite (2) and (4) as follows:

$$E_{\text{dense}} = c_a + c_\Omega - 2\gamma(b_I) + \mathcal{O}(1/n) \tag{7}$$

$$E_{\text{CSR}} = (1 - p_0)(c_a + c_\Omega) + \mathcal{O}(1/n) \tag{8}$$

C. Efficiency Analysis of the CER and CSER Formats

Following a similar reasoning as mentioned above, we can state Theorem 1.

Theorem 1: Let $M \in \mathbb{R}^{m\times n}$ be a matrix. Let further $p_0 \in (0, 1)$ be the empirical probability mass distribution of the zero element, and let $b_I \in \mathbb{N}$ be the bit size of the numerical representation of a column or row index in the matrix. Then, the CER representation of M requires

$$S_{\text{CER}} = (1 - p_0)b_I + \frac{\bar{k} + \tilde{k}}{n}b_I + \mathcal{O}(1/n) + \mathcal{O}(1/N) \tag{9}$$

effective bits per matrix element, where $\bar{k}$ denotes the average number of shared elements that appear per row (excluding the most frequent value), $\tilde{k}$ is the average number of padded indices per row, and $N = m \times n$ is the total number of elements of the matrix. Moreover, the effective cost associated with the dot product with an input vector $a \in \mathbb{R}^n$ is

$$E_{\text{CER}} = (1 - p_0)c_a + \frac{\bar{k}}{n}c_\Omega + \frac{\tilde{k}}{n}\gamma(b_I) + \mathcal{O}(1/n) \tag{10}$$

per matrix element, where c_a and c_Ω are as in (5) and (6). Analogously, we can state Theorem 2.

Theorem 2: Let M, p_0, b_I, $\bar{k}$, c_a, and c_Ω be as in Theorem 1. Then, the CSER representation of M requires

$$S_{\text{CSER}} = (1 - p_0)b_I + \frac{2\bar{k}}{n}b_I + \mathcal{O}(1/n) + \mathcal{O}(1/N) \tag{11}$$

effective bits per matrix element, and the per element cost associated with the dot product with an input vector $a \in \mathbb{R}^n$ is

$$E_{\text{CSER}} = (1 - p_0)c_a + \frac{\bar{k}}{n}c_\Omega + \frac{\bar{k}}{n}\gamma(b_I) + \mathcal{O}(1/n). \tag{12}$$

The proofs of Theorems 1 and 2 are in the Supplementary Material II. These theorems state that the efficiency of the data structures depends on the $(\bar{k}, p_0)$ (average number of distinct elements per row—sparsity) values of the empirical distribution of the elements of the matrix. That is, these data structures are increasingly efficient for distributions that have high p_0 and low $\bar{k}$ values. However, since the entropy measures the effective average number of distinct values that a random variable outputs,[4] both values are intrinsically related to it. In fact, from Renyi's generalized entropy definition [12], we know that $p_0 \geq 2^{-H}$. Moreover, the following properties are satisfied.

1) $\bar{k} \to \min\{K - 1, n\}$, as $H \to \log_2 K$ or $n \to \infty$, and
2) $\bar{k} \to 0$, as $H \to 0$ or $n \to 1$.

Consequently, we can state Corollary 1.

Corollary 1: For a fixed set size of unique element $|\Omega| = K$ and constant index bit size b_I, the storage requirements S as well as the cost of the dot product operation E of the CER and CSER representations satisfy

$$S, E \leq \mathcal{O}(1 - 2^{-H}) + \mathcal{O}(K/n) + \mathcal{O}(1/N) = \mathcal{O}(1 - 2^{-H}) + \mathcal{O}(1/n)$$

[4]From Shannon's source coding theorem [11], we know that the entropy H of a random variable gives the effective average number of bits that it outputs. Therefore, we may interpret 2^H as the effective average number of distinct elements that a particular random variable outputs.

where p_0, b_I, n, and N are as in Theorems 1 and 2, and H denotes the entropy of the matrix element distribution.

Thus, the efficiency of the CER and CSER data structures increases as the column size increases, or as the entropy decreases. Interestingly, when $n \to \infty$, both representations will converge to the same values, and thus, will become equivalent. In addition, there will always exist a column size n, where both formats are more efficient than the original dense and sparse representations (see Fig. 4 where this trend is demonstrated experimentally).

D. Connection Between CSR, CER, and CSER

The CSR format is considered to be one of the most general sparse matrix representations, since it makes no further assumptions regarding the empirical distribution of the matrix elements. Consequently, it implicitly assumes a spike-and-slab distribution on them. However, spike-and-slab distributions are a particular class of low-entropic (for sufficiently high sparsity levels p_0) distributions. In fact, spike-and-slab distributions have the highest entropy values compared to all other distributions that have the same sparsity level. In contrast, as a consequence of Corollary 1, the CER and CSER data structures relax this prior and can, therefore, efficiently represent the entire set of low-entropic distributions. Hence, the CSR data structure can be interpreted as a more specialized version of the CER and CSER representations.

This may be more evident via the following example: consider the first row of the matrix example from Section III:

$$(0 \quad 3 \quad 0 \quad 2 \quad 4 \quad 0 \quad 0 \quad 2 \quad 3 \quad 4 \quad 0 \quad 4).$$

The CSER data structure would represent the above row in the following manner:

$$\begin{aligned} &\Omega : [0, 4, 3, 2] \\ &colI : [4, 9, 11, 1, 8, 3, 7] \\ &\Omega I : [1, 2, 3] \\ &\Omega Ptr : [0, 3, 5, 7] \\ &rowPtr : [0, 3]. \end{aligned}$$

In comparison, the CER representation assumes that the ordering of the elements in ΩI is similar for all rows; therefore, it directly omits this array and implicitly encodes this information in the Ω array. Therefore, the CER representation can be interpreted as a more explicit/specialized version of the CSER. The representation would then be

$$\begin{aligned} &\Omega : [0, 4, 3, 2] \\ &colI : [4, 9, 11, 1, 8, 3, 7] \\ &\Omega Ptr : [0, 3, 5, 7] \\ &rowPtr : [0, 3]. \end{aligned}$$

Similarly, the CSR representation omits the ΩPtr array since it assumes a uniform distribution over the nonzero elements (thus, over the Ω array), and in such case, all the entries in ΩPtr would redundantly be equal to 1. Therefore, the respective representation would be

$$\begin{aligned} &\Omega : [3, 2, 4, 2, 3, 4, 4] \\ &colI : [1, 3, 4, 7, 8, 9, 11] \\ &rowPtr : [0, 7]. \end{aligned}$$

Consequently, the CER and CSER representations will have superior performance for all those distributions that are not similar to the spike-and-slab distributions. Fig. 2 shows a sketch of the regions on the entropy-sparsity plane, where we expect the different data structures to be more efficient. The sketch shows that the efficiency of sparse data structures is high on the subset of distributions that are close to the right border line of the (H, p_0)-plane and that are close to the family of spike-and-slab distribution. In contrast, dense representations are increasingly efficient for high-entropic distributions, hence, in the top-left region. The CER and CSER data structures would then cover the rest of them. Fig. 3 confirms this trend experimentally.

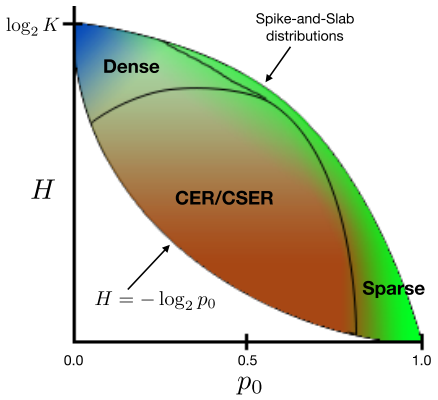

Fig. 2. Sketch of efficiency regions of the different data structures on the entropy-sparsity-plane (H denotes the entropy and p_0 is the sparsity). A point in the plane corresponds to a distribution of the matrix elements with respective entropy-sparsity value. Red, blue, and green regions indicate regions where CER/CSER, dense, and sparse data structures are most efficient. The bottom line corresponds to a min-entropy distribution, and the line at the most right represents the family of spike-and-slab distributions.

V. Experiments

We applied the dense, CSR, CER, and CSER representations on simulated matrices as well as on quantized neural network weight matrices and benchmarked their efficiency with regard to the following four criteria.

1) *Storage Requirements:* We calculated the storage requirements according to (1), (3), (9), and (11).
2) *Number of Operations:* We implemented the dot product algorithms associated with the four above data structures (implementation details and pseudocodes can be found in the Supplementary Materials I-A and I-C) and counted the number of elementary operations they require to perform a matrix-vector multiplication.
3) *Time Complexity:* We timed each respective elementary operation and calculated the total time from the sum of those values.

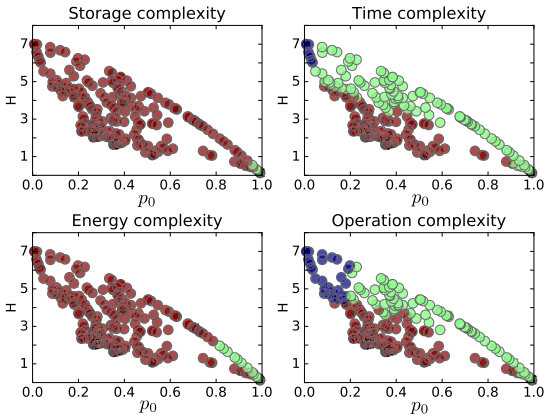

Fig. 3. Plots show the most efficient data structure at different points in the $H - p_0$ plane. The colors indicate the most efficient data structure at that point in the plane. We compare the dense data structure (blue), the CSR format (green), and the proposed CER/CSER data structures (red). We employed a 100×100 matrix and calculated the average complexity over ten matrix samples at each point. The size of the set of the elements was 2^7.

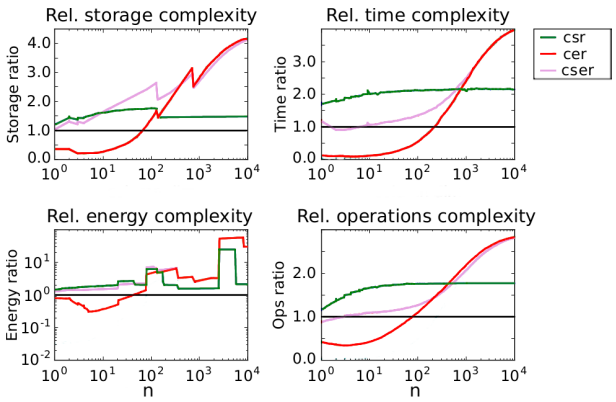

Fig. 4. Efficiency ratios compared to the dense data structure of the different data representations. n denotes the column size. We chose a matrix with $H = 4$, $p_0 = 0.55$, and a fixed row size of 100. The results show the averaged values over 20 matrix samples. The size of the set of the elements was 2^7. The proposed data structures tend to be more efficient, as the column dimension of the matrix increases, and converge to the same value for $n \to \infty$.

TABLE I

ENERGY VALUES (IN pJ) OF DIFFERENT ELEMENTARY OPERATIONS FOR A 45-nm CMOS PROCESS [52]. WE SET THE 8-bit FLOATING POINT OPERATIONS TO BE HALF THE COST OF A 16-bit OPERATION, WHEREAS WE LINEARLY INTERPOLATED THE VALUES IN THE CASE OF THE READ AND WRITE OPERATIONS

Op	8 bits	16 bits	32 bits
float add	0.2	0.4	0.9
float mul	0.6	1.1	3.7
R/W (<8KB)	1.25	2.5	5.0
R/W (<32KB)	2.5	5.0	10.0
R/W (<1MB)	12.5	25.0	50.0
R/W (>1MB)	250.0	5000.0	1000.0

4) *Energy Complexity:* We estimated the respective energy cost by weighting each operation according to Table I. The total energy results consequently from the sum of those values. As for the case of the IO operations (read/write operations), their energy cost depends on the size of the memory the values reside on. Therefore, we calculated the total size of the array where a particular number is entailed and chose the respective maximum energy value. For instance, if a particular column index is stored using a 16-bit representation and the total size of the column index array is 30 kB, then the respective read/write energy cost would be 5.0 pJ.

In addition, we used single-precision floating point representations for the matrix elements and unsigned integer representations for the index and pointer arrays. For the later, we compressed the index-element-values to their minimum required bit sizes, where we restricted them to be either 8, 16, or 32 bits.

Note that we do not consider the complexity of converting the dense representation into the different formats in our experiments. This is justified in the context of neural network compression since we can apply this step *a priori* to the inference procedure. That is, in most real-world scenarios, one first converts the weight matrices, possibly with the help of a capable computer, and then deploys the converted neural network into a resource-constrained device. We are mostly interested in the resource consumption that will take place on the device. Nevertheless, as an additional side note, we would like to mention that the algorithmic complexity of conversion into the CSR, CER, and CSER representations is of $\mathcal{O}(N)$, that is, of the order of the number of elements in the matrix.

A. Experiments on Simulated Matrices

As the first experiments, we aimed to confirm the theoretical trends described in Section IV.

1) Efficiency on Different Regions of the Entropy-Sparsity Plane: First, we argued that each distribution has a particular entropy-sparsity value, and that the superiority of the different data structures is manifested in different regions on that plane. Concretely, we expected the dense representation to be increasingly more efficient in the top-left corner, the CSR on the bottom-right (and along the right border), and the CER and CSER on the rest.

Fig. 3 shows the result of performing one such experiment. In particular, we randomly selected a point distribution on the (H, p_0)-plane and sampled ten different matrices from that distribution. Subsequently, we converted each matrix into the respective dense, CSR, CER, and CSER representations and benchmarked the performance with regard to the four different measures described earlier. We then averaged the results over these ten different matrices. Finally, we compared the performances with each other and, respectively, color-coded the max result. That is, blue corresponds to points where the dense representation was the most efficient, green to the CSR, and red to either the CER or CSER. As one can see, the result closely matches the expected behavior.

2) Efficiency as a Function of the Column Size: As the second experiment, we study the asymptotic behavior of the data structures, as we increase the column size of the matrices. From Corollary 1, we expect that the CER and CSER data structures increase their efficiency as the number of columns in the matrix grows (thus, as $n \to \infty$), until they converge to the same point, outperforming the dense and sparse data structures. Fig. 4 confirms this trend experimentally with

regard to all four benchmarks. Here, we chose a particular point distribution on the (H, p_0)-plane and fixed the number of rows. Concretely, we chose $H = 4.0$, $p_0 = 0.55$, and $m = 100$ (the latter is the row dimension) and measured the average complexity of the data structures as we increased the number of columns $n \to \infty$.

As a side note, the sharp changes in the plots are due to the sharp discontinuities in the values of Table I. For instance, the sharp drops in storage ratios come from the change of the index bit sizes, e.g., from $8 \to 16$ bits.

B. Compressed Neural Networks Without Retraining

As the second set of experiments, we tested the efficiency of the proposed data structures on compressed deep neural networks. In particular, we benchmarked their weight matrices relative to the matrix-vector operation, after them being compressed using two different types of quantization techniques: one where retraining of the network is required (Section V-C) and one where it is not (Section V-B). We treat them separately, since the statistics of the resulting compressed weight matrices are conditioned by the quantization applied on them. Further results can be found in the Supplementary Material I-B.

We start by first analyzing the latter case. This scenario is of particular interest since it applies to cases where one does not have access to the training data (e.g., federated learning scenario) or it is prohibited to retrain the model (e.g., limited access to computational resources). Moreover, common matrix representations, such as the dense or CSR, may fail to efficiently exploit the statistics present in these compressed weight matrices (see Fig. 1 and discussion in Section II).

In our experiments, we first quantized the elements of the weight matrices of the networks in a lossy manner, while ensuring that we negligible impact their prediction accuracy. Similar to [34] and [35], we applied a uniform quantizer over the range of weight values at each layer and subsequently rounded the values to their nearest quantization point. That is, for each weight matrix W, we calculated the range of values $[w_{\min}, w_{\max}]$ (with $w_{\min}$ being the lowest weight element value and $w_{\max}$ analogously) and inserted $K = 2^b$ equidistant points inside that range, whose values were stored in the array Ω. Then, we quantized each weight element in W to its closest neighbor relative to Ω and measured the validation accuracy of the quantized network. In our experiments, we did not see any significant impact on the accuracy for all $b \geq 7$ (see Table II). We chose the uniform quantizer because of its simplicity and high performance relative to other more sophisticated quantizers such as entropy-constrained k-mean algorithms [34], [35]. Finally, we lossless converted the quantized weight matrices into the different data structures and tested their efficiency with regard to the four above mentioned benchmark criteria.

1) Storage Requirements: Table II shows the gains in storage requirements of different state-of-the-art neural networks. Gains can be attained when storing the networks in CER or CSER formats. In particular, we achieve more than $\times 2.5$ savings on the DenseNet architecture [53], whereas in contrast, the CSR data structure attains negligible gains. This is mainly attributed to the fact that the dense and sparse representations store very inefficiently the weight element values of these networks. This is also reflected in Fig. 5, where one can see that most of the storage requirements for the dense and CSR representations are spent in storing the elements of the weight matrices Ω. In contrast, most of the storage cost for the CER and CSER data structures comes from storing the column indices colI, which is much lower than the actual weight values.

TABLE II

STORAGE GAINS OF MODELS AFTER THEIR WEIGHT MATRICES HAVE BEEN COMPRESSED DOWN TO 7 bits AND CONVERTED INTO THE DIFFERENT DATA STRUCTURES. THE GAINS ARE RELATIVE TO THE ORIGINAL DENSE REPRESENTATION OF THE COMPRESSED WEIGHT MATRICES, AND THEY SHOW THE OVER THE LAYERS' AGGREGATED RESULTS. THE ACCURACY IS MEASURED ON THE VALIDATION SET (IN PARENTHESIS IS THE ACCURACY OF THE ORIGINAL MODEL) OF THE IMAGENET CLASSIFICATION TASK

Storage	Accuracy [%]	original [MB]	CSR	CER	CSER
VGG16	71.63 (71.59)	553.43	x0.71	x2.11	x2.11
ResNet152	78.83 (78.31)	240.77	x0.76	x2.08	x2.10
DenseNet	77.02 (77.12)	114.72	x1.04	x2.74	x2.79

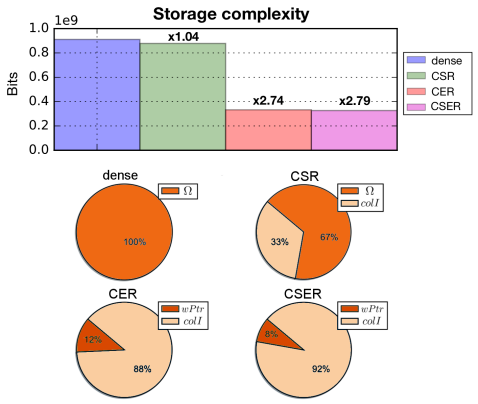

Fig. 5. Storage requirements of a compressed DenseNet after converting its weight matrices into the different data structures. The weights of the network layers were compressed down to 7 bits (resulting accuracy is 77.09%). The plots show the over the layers' averaged result. Top: compression ratio relative to the dense format. Bottom: contribution of different parts of the data structures to the storage requirements.

2) Number of Operations: Table III shows the savings attained with regard to the number of elementary operations needed to perform a matrix-vector multiplication. As one can see, we can save up to 40% of the number of operations if we use the CER/CSER data structures on the DenseNet architecture. This is mainly due to the fact that the dot product algorithm of the CER/CSER formats implicitly encodes the distributive law of multiplications and consequently they require much less number of them. This is also reflected in Fig. 6, where one can see that the CER/CSER dot product algorithms are mainly performing input load (In_{load}), column index load ($\text{colI}_{\text{load}}$), and addition (*add*) operations. Here, *others* refer to any other operation involved in the dot product,

TABLE III

GAINS ATTAINED WITH REGARD TO THE NUMBER OF OPERATIONS, TIME AND ENERGY COST NEEDED FOR PERFORMING A MATRIX-VECTOR MULTIPLICATION WITH THE COMPRESSED WEIGHT MATRICES OF DIFFERENT NEURAL NETWORKS. THE EXPERIMENT SETTING AND TABLE STRUCTURE IS THE SAME AS IN TABLE II

#ops [G] time [s] energy [J]	original	CSR	CER	CSER
VGG16	15.08 3.37 2.70	x0.88 x0.85 x0.76	x1.40 x1.27 x2.37	x1.39 x1.29 x2.38
ResNet152	10.08 2.00 1.92	x0.93 x0.93 x1.25	x1.42 x1.30 x3.73	x1.41 x1.31 x3.74
DenseNet	7.14 1.53 0.51	x1.11 x1.10 x1.95	x1.66 x1.43 x6.40	x1.65 x1.45 x6.57

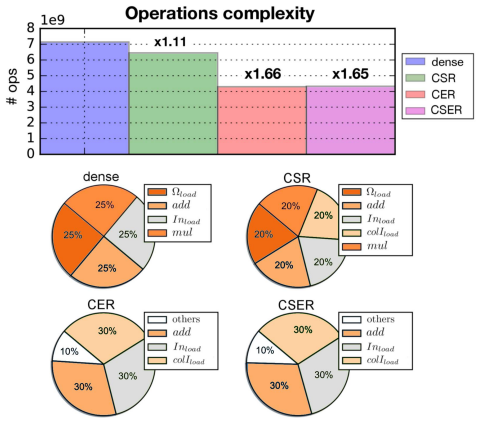

Fig. 6. Number of operations required to perform a dot product in the different formats for the experimental setup described in Fig. 5 (DenseNet). The CER/CSER formats require less operations than the other formats, because they do not need to perform as many multiplications and they do not need to load as many matrix weight elements.

such as multiplications, weight loading, writing, and so on. In contrast, the dense and CSR dot product algorithms require an additional equal number of weight element load (Ω_{load}) and multiplication (*mul*) operations.

3) Time Cost: In addition, Table III also shows that we attain speedups when performing the dot product in the new representation. Interestingly, Fig. 7 shows that most of the time is being consumed on IO's operations (that is, on *load* operations). Consequently, the CER and CSER data structures attain speedups since they do not have to load as many weight elements. In addition, 20% and 16% of the time is spent in performing multiplications, respectively, in the dense and sparse representations. In contrast, this time cost is negligible for the CER and CSER representations.

4) Energy Cost: Similarly, we see that most of the energy consumption is due to IOs' operations (see Fig. 8). Here, the cost of loading an element may be up to three orders of magnitude higher than any other operations (see Table I); therefore, we obtain up to $\times 6$ energy savings when using the CER/CSER representations (see Table III).

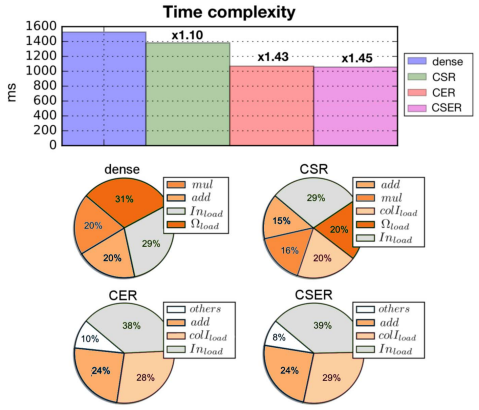

Fig. 7. Time cost of a dot product in the different formats for the experimental setup described in Fig. 5 (DenseNet). The CER/CSER formats save time, because they do not require to perform as many multiplications and they do not spend as much time loading the matrix weight elements.

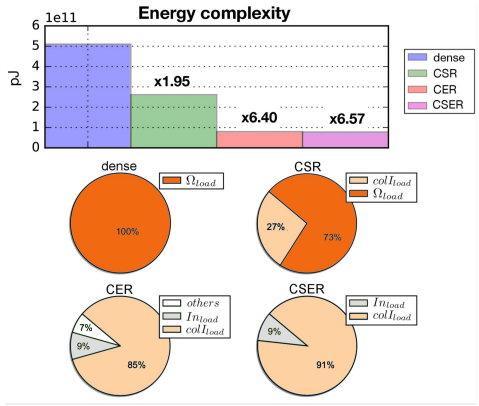

Fig. 8. Energy cost of a dot product in the different formats for the experimental setup described in Fig. 5 (DenseNet). Performing loading operations consumes up to three orders more energy than sum and mul operations (see Table I). Since the CER/CSER formats need substantially less matrix weight element loading operations, they attain great energy saving compared to the dense and CSR formats.

TABLE IV

STATISTICS OF DIFFERENT NEURAL NETWORK WEIGHT MATRICES TAKEN OVER THE ENTIRE NETWORK. p_0 DENOTES THE SPARSITY LEVEL OF THE NETWORK, H THE ENTROPY, $\bar{k}$ THE NUMBER OF SHARED ELEMENTS PER ROW, AND n THE COLUMN DIMENSION. ALL NEURAL NETWORKS HAVE RELATIVELY LOW ENTROPY, I.E., A LOW NUMBER OF SHARED ELEMENTS COMPARED TO THE COLUMN DIMENSIONALITY

	p_0	H	$\bar{k}$	n	$\bar{k}/n$
VGG16	0.07	4.8	55.80	10311.86	0.01
ResNet152	0.12	4.12	32.35	782.67	0.03
DenseNet	0.36	3.73	43.95	1326.93	0.03
AlexNet [25]	0.89	0.89	18.33	5767.85	0.01

Finally, Table IV and Fig. 9 further justify the observed gains. Namely, Table IV shows that the effective number of shared elements per row of the network is small relative to the networks' effective column dimension. To clarify, we calculated the effective number of shared elements by: 1) for all rows, calculating the number of shared weights; 2) aggregating

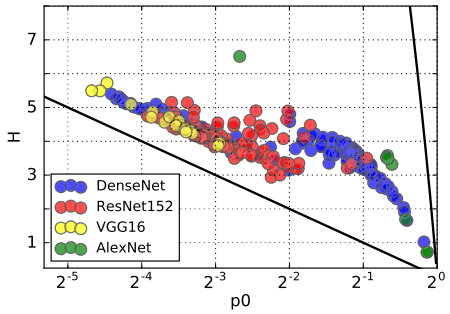

Fig. 9. Empirical distributions of the weight matrices of different neural network architectures after compression displayed on the entropy-sparsity plane. As we see, most of the layers lay in the region where the CER/CSER data structures outperform the dense and sparse representations. The top and bottom black lines constrain the set of possible distributions.

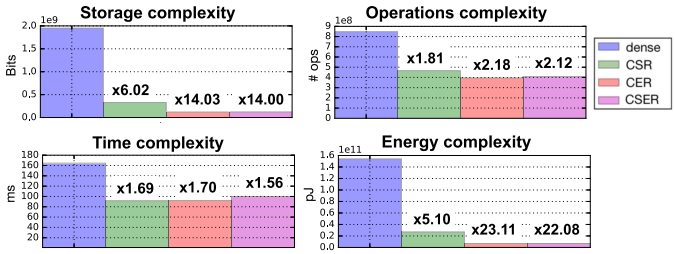

Fig. 10. Efficiency comparison of a compressed AlexNet after converting its weight matrices into the different data structures and benchmarking their matrix-vector dot product operation. The network was compressed using the deep compression technique. The plots show the over the layers aggregated results compared to the original dense data structure.

the numbers; and 3) dividing the result by the total number of rows that appear in the network. Similarly, the effective number of columns indicates the average number of columns in the network, and the effective sparsity level and effective entropy values indicate the over the total number of weights averaged result. Fig. 9 shows the distributions of the different layers of the networks on the entropy-sparsity plane, where we see that most of them lay in the regions where we expect the CER/CSER formats to be more efficient.

C. Compressed Neural Networks With Retraining

In this section, we benchmark the CER/CSER matrix representation on networks whose weight matrices have been compressed using quantization techniques where retraining was required in the process. This case is also of particular interest since highest compression gains can only be achieved if one applies such quantizations techniques on to the network [25], [26], [36]–[38].

For instance, deep compression [25] is a technique for compressing neural networks that is able to attain high compression rates without incurring significant loss of accuracy. It is able to do so by applying a three-staged pipeline: 1) prune unimportant connections by employing algorithm [30]; 2) cluster the nonpruned weight values and refine the cluster centers to the loss surface; and 3) employ an entropy coder for storing the final weights. Note that the first two stages aim to implicitly minimize the entropy of the weight matrices without incurring significant loss of accuracy, whereas the third stage lossless converts the weight matrices into low-bit representation. However, the proposed representation is based on the CSR format, and consequently, the complexity of the respective dot product algorithm remains on the same order. Concretely, the total number of operations that need to be performed is greater or equal to the original CSR format. In fact, one requires specialized hardware in order to efficiently exploit this final neural network representation during inference [54]. Therefore, many authors benchmark the inference efficiency of highly compressed deep neural networks with regard to the standard CSR representation when tested on standard hardware such as CPUs and/or GPUs [25], [37], [50]. However, this comes at the cost of adding redundancies, since then one does not exploit step 2 of the compression pipeline. In contrast, the CER/CSER representations become increasingly efficient as the entropy of the network is reduced, even if the sparsity level is maintained (see Figs. 2 and 3). Hence, it is of high interest to benchmark their efficiency on highly compressed networks and compare them to their sparse (and dense) counterparts.

As the first experiment, we chose the AlexNet model [55], which was compressed using the deep compression technique.[5] The overall entropy of the network was reduced down to 0.89 without incurring any loss of accuracy. Fig. 10 shows the gains in efficiency when the network layers are converted into the different data structures. We see that the proposed data structures are able to surpass the dense and sparse data structures for all four benchmark criteria. Therefore, CER/CSER data structures are much less redundant and efficient representations of highly compressed neural network models. Interestingly, the CER/CSER data structures attain up to $\times 14$ storage and $\times 20$ energy savings, which is considerably higher than the sparse counterpart. Nevertheless, we do not attain significant time gains. This is due to the fact that, in our implementations, the time cost of loading the input elements was significantly higher than any other component in the algorithm (see Fig. 3 in the Supplementary Material). This also explains why the CSR format shows similar speedups than the CER and CSER. However, this effect can be mitigated if one applies further optimizations on the input vector, such as data reuse techniques and/or better storage management of its values during the dot product procedure. With that, we also expect significant gains in time performance relative to the CSR format.

Finally, we trained and compressed additional architectures while following a similar compression pipeline, as described in [25]. Concretely, we: 1) pretrained the architectures until we reached the state-of-the-art accuracies; 2) sparsified the architectures using the technique proposed in [26]; 3) applied a uniform quantizer to the nonzero values in order to reduce their effective bit size; and 4) converted the weight matrices into the different representations and benchmarked their efficiency relative to their matrix-vector product operation. In step 2), we chose [26] since it is the current state-of-the-art

[5]https://github.com/songhan/Deep-Compression-AlexNet

TABLE V

STORAGE GAINS OF DIFFERENT MODELS AFTER THEY HAVE BEEN COMPRESSED AS DESCRIBED IN SECTION V-C. THE VGG MODEL WAS TRAINED ON CIFAR-10 AS BENCHMARKED IN [26] AND [37]. THE LENET ARCHITECTURES WERE TRAINED ON MNIST AS BENCHMARKED IN [26] AND [37]. THE ACCURACY COLUMN (ACC) SHOWS THE ACCURACIES OF THE COMPRESSED (PARENTHESIS) MODELS. FINALLY, THE SPARSITY COLUMN (SP) DISPLAYS THE RATIO BETWEEN THE NONZERO WEIGHT VALUES AND THE TOTAL NUMBER OF WEIGHT ELEMENTS

Storage	Acc [%]	sp [%]	orgnl [MB]	CSR	CER	CSER
VGG-CIFAR10	90.13 (91.54)	4.28	59.91	x17.00	**x41.95**	x41.59
LeNet-300-100	97.16 (98.32)	9.05	1.06	x8.00	**x19.52**	x18.98
LeNet5	98.27 (99.44)	1.90	1.722	x35.08	**x73.16**	x72.62

TABLE VI

GAINS ATTAINED (AGGREGATED OVER LAYERS) WITH REGARD TO THE NUMBER OF OPERATIONS, AND TIME AND ENERGY COST NEEDED WHEN BENCHMARKING THE MATRIX-VECTOR MULTIPLICATION OF THE WEIGHT MATRICES OF THE NETWORKS DESCRIBED IN TABLE V. THE PERFORMANCE GAINS ARE RELATIVE TO THE ORIGINAL DENSE REPRESENTATION OF THE COMPRESSED WEIGHT MATRICES

#ops [M] time [ms] energy [mJ]	original	CSR	CER	CSER
VGG-CIFAR10	878.38 208.00 139.64	x3.71 x3.63 x35.41	x5.53 x5.09 x89.81	x5.43 x5.10 **x90.34**
LeNet-300-100	1.065 0.25 0.02	x9.54 x9.76 x14.23	x12.73 x11.61 **x54.46**	x12.33 x11.10 x54.10
LeNet5	7.59 1.94 0.48	x3.61 x3.52 x60.90	x4.15 x3.54 x87.49	x4.00 x3.63 **x96.58**

sparsification technique. In our experiments, we chose to benchmark the same architectures as reported in [26] and [37], that is, an adapted version of the VGG network[6] for the CIFAR-10 image classification task and the fully connected and convolutional LeNet architectures for the MNIST classification task. The respective accuracies and compression gains can be seen in Table V and the gains relative to the dot product complexity in Table VI. As we can see, we attain significantly higher gains in all four benchmarks when we convert their weight matrices into the CER/CSER representations. In particular, we are able to attain up to $\times 42$ compression gains, $\times 5$ speedups, and $\times 90$ energy gains on the VGG model.

As a last side note, we want to mention again that compressing further the CSR representation by, for instance, replacing the nonzero values by their respective quantization indices (as proposed in [25]) does not necessarily result in higher gains with regard to the dot product since it requires an additional decoding step per nonzero element in the process. For instance, we got only $\times 2.89$ speedups on our compressed CIFAR10-VGG model, which is less than the speedups attained by the original CSR format ($\times 3.63$). Furthermore, we attained $\times 33.62$, $\times 3.10$, and $\times 62.32$ gains in storage, the number of operations, and energy, respectively, which is also lower than the gains attained by the CER/CSER representations (Tables V and VI).

[6]http://torch.ch/blog/2015/07/30/cifar.html.

VI. CONCLUSION

We presented two new matrix representations, CER and CSER, that are able to attain high compression ratios and energy savings if the distribution of the matrix elements has low entropy. We showed on an extensive set of experiments that the CER/CSER data structures are more compact and computationally efficient representations of compressed state-of-the-art neural networks than dense and sparse formats. In particular, we attained up to $\times 42$ compression ratios and $\times 90$ energy savings by representing the weight matrices of a highly compressed VGG model in their CER/CSER forms and benchmarked against the matrix-vector product operation.

By demonstrating the advantages of entropy-optimized data formats for representing neural networks, this paper opens up new directions for future research, e.g., the exploration of entropy-constrained regularization and quantization techniques for compressing deep neural networks [39]. The combination of entropy-constrained regularization and quantization and entropy-optimized data formats may push the limits of neural network compression even further and also be beneficial for applications such as federated or distributed learning [56], [57].

Future work will also study lossy compression schemes, especially in combination with their analysis with explanation methods [58], [59].

REFERENCES

[1] R. H. Landau, J. Paez, and C. C. Bordeianu, *A Survey of Computational Physics: Introductory Computational Science*. Princeton, NJ, USA: Princeton Univ. Press, 2008.
[2] D. M. Young and R. T. Gregory, *A Survey of Numerical Mathematics*. New York, NY, USA: Dover, 1988.
[3] S. Krig, *Computer Vision Metrics: Survey, Taxonomy, and Analysis*, 1st ed. Berkely, CA, USA: Apress, 2014.
[4] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade* (Lecture Notes in Computer Science), vol. 7700, 2nd ed. Berlin, Germany: Springer, 2012, pp. 9–48.
[5] S. Afroz, M. Tahaseen, F. Ahmed, K. S. Farshee, and M. N. Huda, "Survey on matrix multiplication algorithms," in *Proc. 5th Int. Conf. Inform., Electron. Vis. (ICIEV)*, 2016, pp. 151–155.
[6] M. Bläser, "Fast matrix multiplication," *Theory Comput., Graduate Surv.*, vol. 5, pp. 1–60, Dec. 2013.
[7] I. S. Duff, "A survey of sparse matrix research," *Proc. IEEE*, vol. 65, no. 4, pp. 500–535, Apr. 1977.
[8] V. Karakasis, T. Gkountouvas, K. Kourtis, G. Goumas, and N. Koziris, "An extended compression format for the optimization of sparse matrix-vector multiplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 10, pp. 1930–1940, Oct. 2013.
[9] J. King, T. Gilray, R. M. Kirby, and M. Might, "Dynamic-CSR: A format for dynamic sparse-matrix updates," in *Proc. 31st Int. Conf. High Perform. Comput.* (Lecture Notes in Computer Science), vol. 9697. Cham, Switzerland: Springer-Verlag, 2016, pp. 61–80.
[10] R. Yuster and U. Zwick, "Fast sparse matrix multiplication," in *Algorithms—ESA*. Berlin, Germany: Springer, 2004, pp. 604–615.
[11] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 5, no. 1, pp. 3–55, 2001.

[12] A. Rényi, "On measures of entropy and information," in *Proc. 4th Berkeley Symp. Math. Statist. Probab., Contrib. Theory Statist.*, vol. 1, 1961, pp. 547–561.
[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.
[14] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.
[15] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.
[16] S. Bosse, D. Maniry, K.-R. Müller, T. Wiegand, and W. Samek, "Deep neural networks for no-reference and full-reference image quality assessment," *IEEE Trans. Image Process.*, vol. 27, no. 1, pp. 206–219, Jan. 2018.
[17] W. Dai, C. Dai, S. Qu, J. Li, and S. Das. (2016). "Very deep convolutional neural networks for raw waveforms." [Online]. Available: https://arxiv.org/abs/1610.00087
[18] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. Int. Conf. Represent. Learn. (ICLR)*, 2015, pp. 1–11.
[19] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature Commun.*, vol. 5, Jul. 2014, Art. no. 4308.
[20] I. Sturm, S. Lapuschkin, W. Samek, and K.-R. Müller, "Interpretable deep neural networks for single-trial eeg classification," *J. Neurosci. Methods*, vol. 274, pp. 141–145, Dec. 2016.
[21] K. T. Schütt, F. Arbabzadah, S. Chmiela, K.-R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nature Commun.*, vol. 8, no. 4, p. 13890, 2017.
[22] S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller, "Machine learning of accurate energy-conserving molecular force fields," *Sci. Adv.*, vol. 3, no. 5, 2017, Art. no. e1603015.
[23] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.
[24] G. Hinton, O. Vinyals, and J. Dean. (2015). "Distilling the knowledge in a neural network." [Online]. Available: https://arxiv.org/abs/1503.02531
[25] S. Han, H. Mao, and W. J. Dally. (2015). "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding." [Online]. Available: https://arxiv.org/abs/1510.00149
[26] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 2498–2507.
[27] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. (2017). "A survey of model compression and acceleration for deep neural networks." [Online]. Available: https://arxiv.org/abs/1710.09282
[28] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann, 1990, pp. 598–605.
[29] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 1, 1993, pp. 293–299.
[30] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[31] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learn. Unsupervised Feature Learn. Workshop*, 2011, pp. 1–8.
[32] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 2849–2858.
[33] F. Li, B. Zhang, and B. Liu. (2016). "Ternary weight networks." [Online]. Available: https://arxiv.org/abs/1605.04711
[34] Y. Choi, M. El-Khamy, and J. Lee. (2016). "Towards the limit of network quantization." [Online]. Available: https://arxiv.org/abs/1612.01543
[35] Y. Choi, M. El-Khamy, and J. Lee. (2018). "Universal deep neural network compression." [Online]. Available: https://arxiv.org/abs/1802.02271
[36] K. Ullrich, E. Meeds, and M. Welling. (2017). "Soft weight-sharing for neural network compression." [Online]. Available: https://arxiv.org/abs/1702.04008
[37] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3290–3300.
[38] M. Federici, K. Ullrich, and M. Welling. (2017). "Improved Bayesian compression." [Online]. Available: https://arxiv.org/abs/1711.06494
[39] S. Wiedemann, A. Marban, K.-R. Müller, and W. Samek, "Entropy-constrained training of deep neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, to be published.
[40] K. Simonyan and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556
[41] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5145–5153.
[42] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan. (2018). "Training deep neural networks with 8-bit floating point numbers." [Online]. Available: https://arxiv.org/abs/1812.08011
[43] *TensorFlow Lite*. Accessed: Feb. 28, 2019. [Online]. Available: https://www.tensorflow.org/lite
[44] *QNNPACK Open Source Library for Optimized Mobile Deep Learning*. Accessed: Feb. 28, 2019. [Online]. Available: https://github.com/pytorch/QNNPACK
[45] N. Mellempudi, A. Kundu, D. Das, D. Mudigere, and B. Kaul. (2017). "Mixed low-precision deep learning inference using dynamic fixed point." [Online]. Available: https://arxiv.org/abs/1701.08978
[46] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. (2016). "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1." [Online]. Available: https://arxiv.org/abs/1602.02830
[47] M. Kim and P. Smaragdis. (2016). "Bitwise neural networks." [Online]. Available: https://arxiv.org/abs/1601.06071
[48] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. (2016). "XNOR-Net: ImageNet classification using binary convolutional neural networks." [Online]. Available: https://arxiv.org/abs/1603.05279
[49] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li. (2017). "GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework." [Online]. Available: https://arxiv.org/abs/1705.09283
[50] T. Yang, Y. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5687–5695.
[51] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
[52] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.
[53] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2016, pp. 4700–4708.
[54] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.
[55] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[56] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas. (2016). "Communication-efficient learning of deep networks from decentralized data." [Online]. Available: https://arxiv.org/abs/1602.05629
[57] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, to be published.
[58] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," *ITU J., ICT Discoveries*, vol. 1, no. 1, pp. 39–48, 2018.
[59] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller, "Unmasking clever Hans predictors and assessing what machines really learn," *Nature Commun.*, vol. 10, Mar. 2019, Art. no. 1096.

Simon Wiedemann received the M.Sc. degree in applied mathematics from Technische Universität Berlin, Berlin, Germany.

He is currently with the Machine Learning Group, Fraunhofer Heinrich Hertz Institute, Berlin. His major research interests include machine learning, neural networks, and information theory.

Klaus-Robert Müller (M'12) studied physics in Karlsruhe, Germany, from 1984 to 1989, and received the Ph.D. degree in computer science from Technische Universität Karlsruhe, Karlsruhe, in 1992.

After completing a postdoctoral position at GMD FIRST, Berlin, Germany, he was a Research Fellow with The University of Tokyo, Tokyo, Japan, from 1994 to 1995. In 1995, he founded the Intelligent Data Analysis Group, GMD-FIRST (later Fraunhofer FIRST), and directed it until 2008. From 1999 to 2006, he was a Professor with the University of Potsdam, Potsdam, Germany. He has been a Professor of computer science with Technische Universität Berlin, Berlin, since 2006; at the same time, he is co-directing the Berlin Big Data Center. His current research interests include intelligent data analysis, machine learning, signal processing, and brain–computer interfaces.

Dr. Müller was a recipient of the 1999 Olympus Prize by the German Pattern Recognition Society, DAGM, and he received the SEL Alcatel Communication Award in 2006, the Science Prize of Berlin awarded by the Governing Mayor of Berlin in 2014, and the Vodafone Innovation Award in 2017. In 2012, he was elected as a member of the German National Academy of Sciences Leopoldina, and in 2017, a member of the Berlin Brandenburg Academy of sciences, and an External Scientific Member of the Max Planck Society.

Wojciech Samek (M'13) received the Diploma degree in computer science from the Humboldt University of Berlin, Berlin, Germany, in 2010, and the Ph.D. degree in machine learning from Technische Universität Berlin, Berlin, in 2014.

In 2014, he founded the Machine Learning Group, Fraunhofer Heinrich Hertz Institute, Berlin, which he currently directs. He is associated with the Berlin Big Data Center and was a Scholar of the German National Academic Foundation and a Ph.D. Fellow at the Bernstein Center for Computational Neuroscience Berlin. He was visiting Heriot-Watt University, Edinburgh, U.K., and The University of Edinburgh, Edinburgh, from 2007 to 2008. In 2009, he was with the Intelligent Robotics Group, NASA Ames Research Center, Mountain View, CA, USA. His research interests include interpretable machine learning, neural networks, signal processing, and computer vision.