New Generation Computing (2023) 41:697–722
https://doi.org/10.1007/s00354-023-00226-1

NEW GENERATION COMPUTING

Structured Matrices and Their Application in Neural Networks: A Survey

Matthias Kissel[1] · **Klaus Diepold**[1]

Received: 19 December 2022 / Accepted: 21 June 2023 / Published online: 26 July 2023

Abstract

Modern neural network architectures are becoming larger and deeper, with increasing computational resources needed for training and inference. One approach toward handling this increased resource consumption is to use structured weight matrices. By exploiting structures in weight matrices, the computational complexity for propagating information through the network can be reduced. However, choosing the right structure is not trivial, especially since there are many different matrix structures and structure classes. In this paper, we give an overview over the four main matrix structure classes, namely semiseparable matrices, matrices of low displacement rank, hierarchical matrices and products of sparse matrices. We recapitulate the definitions of each structure class, present special structure subclasses, and provide references to research papers in which the structures are used in the domain of neural networks. We present two benchmarks comparing the classes. First, we benchmark the error for approximating different test matrices. Second, we compare the prediction performance of neural networks in which the weight matrix of the last layer is replaced by structured matrices. After presenting the benchmark results, we discuss open research questions related to the use of structured matrices in neural networks and highlight future research directions.

Keywords Matrix structures · Neural network · Efficient propagation · Fast inference

✉ Matthias Kissel
matthias.kissel@tum.de

[1] TUM School of Computation, Information and Technology, Technical University of Munich, Arcisstr. 21, 80333 Munich, Bavaria, Germany

1 Introduction

1.1 Structured Matrices

When talking about structured matrices, we build on the notion of data-sparse matrices. Data sparsity means that the representation of an $n \times n$ matrix requires less than $\mathcal{O}(n^2)$ parameters. In contrast to sparse matrices, *data sparse* matrices must not contain zero entries. Instead, there is a relationship between the entries of the matrix. The simplest examples are rank 1 matrices of the form $u \cdot v^T$, for vectors $u, v \in \mathbb{R}^n$. Other easily identifiable examples of data sparse matrices are Toeplitz or Hankel matrices, which may hold $2n - 1$ parameters.

In other, less obvious cases, data sparsity implies that the entries of the respective structured matrices have an intrinsic relationship to each other. As an example for such a relationship, we can point at orthogonal matrices, which comprise $\frac{n(n-1)}{2}$ free parameters. However, orthogonal matrices have obviously $\mathcal{O}(n^2)$ parameters, which means that they do not belong to the class of data-sparse matrices.

We are particularly interested in data-sparse matrices, for which we can find efficient algorithms, for example, computing the matrix–vector product with an arbitrary vector with less than $\mathcal{O}(n^2)$ operations. There exist various matrix structures, which serve as candidates for accomplishing this goal. However, the knowledge on the subject area is quite fragmented containing many approaches originating from diverse fields. In this paper, we give an overview over the four most important structure classes. We put these classes in relation to each other, helping to reveal their boundaries and limitations. By that, we categorize the state-of-the-art in the field of structured matrices.

1.2 Computational Challenges for Neural Networks

Neural networks solve increasingly complex tasks of machine intelligence, like beating humans in the game of Go [78]. However, with increasing complexity of the problems, the complexity of the networks also increases significantly. This creates a trend toward deep networks [45, 90], which consist of a large number of layers and millions of parameters. This increase in complexity creates challenges for practical implementations, where the number of arithmetic operations grows disproportionally fast.

This trend results in the following list of technical challenges:

Training time
The training of deep neural networks can last several weeks even on modern computing architectures. For example, the training of the AlphaGo Zero network, which is able to beat the best human Go players, took 40 days (on specialized hardware) [78]. Long training times result in high costs, for example, due to high server costs. Moreover, long training periods effectively hinder to adapt quickly to new data.

Inference time
The more operations need to be performed in order to compute the output of a neural network, the more time is needed for the computation. Thus, the inference time directly

scales with the number of operations in the neural network (neglecting parallelization capabilities). If the inference takes too long, the applicability of a neural network is restricted to certain applications, where fast inference is not essential. For example, the AlphaGo Zero network needed specialized hardware to be able to answer with reasonably low response time, which is required for playing a game of Go. In the case of AlphaGo Zero, 4 tensor processing units were used in order to perform inference in at most 5 s, and previous versions were distributed on up to 176 GPUs for calculating the next move in real time [78].

Memory requirements
Large neural networks consist of many parameters, which need to be stored. For example, the popular pre-trained ResNet50 [45] network needs 98MB memory space (provided by the keras project[1]). This is by far not the upper limit—there are much bigger architectures available and in use. The required memory capacity can be problematic for resource constraint devices, such as mobile devices, smartphones or microcontrollers. For standard computers (PCs), the amount of memory required to load the whole model into RAM may also be prohibitive.

Memory bandwidth requirements
Besides the large memory needed to store the parameters of a given deep neural network, it is also an issue to provide the memory bandwidth necessary to facilitate fast learning or fast inference. Indeed, it has been shown that for deep neural networks memory access is the main bottleneck for processing [82]. Therefore, significant processing speedups can be achieved by optimizing the memory access to reduce bandwidth [46].

Power consumption
As the amount of operations for performing training or during inference increases, the power consumption also increases. Again, the increasing power consumption is challenging for mobile devices or, more generally, for all battery-driven systems. Besides the costs arising with increased power consumption, neural networks might thus contribute to today's climate change. For example, training big natural language processing models including hyper parameter search can produce up to twice the amount of CO_2 produced by an average American within one year [81]. Therefore, we are usually interested in reducing the power requirements.

1.3 Goals and Organization

Numerous researchers have contributed to mitigate the aforementioned problems. For example, a survey on increasing the efficiency of neural networks is given by Sze et al. [82]. In this paper, we focus on approaches using structured matrices in the domain of neural networks, which has the potential to overcome all mentioned problems.

We see two main advantages of using structured matrices in neural networks to save resources compared to other approaches. First, for many structures, it is possible to train the neural network end-to-end. This means that conventional, well-tested training algorithms such as backpropagation can be used for training. In comparison,

[1] https://keras.io/.

most methods to save resources in neural networks start only after the training, which may lead to worse results. Second, in contrast to the common mindset that resource savings in neural networks always lead to performance losses, we assume that the choice of the right structure can even *improve* the performance. This is the case if the chosen structure fits the problem, and thus the search space for the weights of the neural network is restricted in a meaningful way.

Contribution
Our main contribution is to give an overview over the most important matrix structure classes, and to present two benchmarks comparing the classes. We briefly introduce each structure class, and mention efficient algorithms. For each class, we analyze the computational requirements for computing the matrix–vector product, which plays a major role in neural networks. Moreover, we review approaches where each structure has been used in the domain of neural networks. Through this, our survey offers a starting point to choosing the right structure for a given problem.

Organization
The paper is organized as follows—we first introduce the four main structure classes which we identified from literature, namely semiseparable matrices, matrices of low displacement rank, hierarchical matrices, and products of sparse matrices. Subsequently, we set the structure classes into relation to each other, showing their boundaries. We present two benchmarks comparing the structure classes. The first benchmark investigates the error for approximating different test matrices. The second benchmark compares the prediction performance of neural networks, in which the weight matrix of the last layer is replaced by a structured matrix. In the following section, we point out open research questions and future research directions for using structured matrices in the domain of neural networks. Finally, we summarize our findings and draw a conclusion.

2 Classes of Structured Matrices

2.1 Semiseparable Matrices

The first notion of semiseparable matrices [87] appears in work published in 1937 by Gantmakher and Krein [33, 86]. Since then, there has been a number of publications and generalizations of results to the class of semiseparable matrices [86]. The motivation for research about semiseparable matrices originates from various application domains for computational science and engineering, such as for example time-varying system theory [22], where the matrices appear in the context of simulating physical phenomena and systems. The most prominent representatives in this class are tridiagonal matrices and other banded matrices along with their inverses.

Definition
We focus on the definition of sequentially semiseparable matrices [22]. A sequentially semiseparable matrix T has a block structure based on the matrices A_k, B_k, C_k, D_k, E_k, F_k and G_k

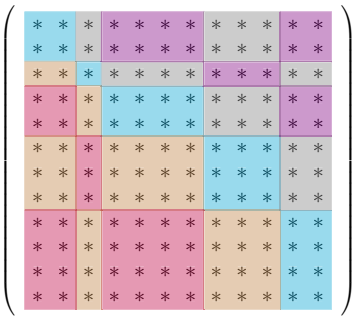

Fig. 1 Schematic Illustration of the partitioning of a sequentially semiseparable matrix. The rectangular shapes of the submatrices illustrate that the input, output, and state dimensions associated with the sequentially semiseparable matrix can change between timesteps

$$T_{i,j} = \begin{cases} D_i & \text{for} \quad i = j, \\ C_i A_{i-1} \dots A_{j+1} B_j & \text{for} \quad i < j, \\ G_i E_{i+1} \dots E_{j-1} F_j & \text{for} \quad i > j. \end{cases} \tag{1}$$

This structure arises in the transfer matrix of a time-varying system with state equations

$$x_{k+1} = A_k x_k + B_k u_k, \tag{2}$$
$$\hat{x}_k = E_k \hat{x}_{k+1} + F_k u_k, \tag{3}$$
$$a_k^{(1)} = C_k x_k + D_k u_k, \tag{4}$$
$$a_k^{(2)} = G_k \hat{x}_{k+1}, \tag{5}$$

and

$$a_k = a_k^{(1)} + a_k^{(2)}, \tag{6}$$

which reveals why this structure is closely related to the theory of time-varying systems. In the domain of time-varying systems, x_k refers to the state of the causal part of the system at timestep k ($\hat{x}_k$ to the anti causal part respectively), u_k are the inputs to the system at timestep k and a_k are the outputs respectively. Note that the dimensions of the A_k, B_k, C_k, D_k, E_k, F_k and G_k might change for different timesteps, which reflects the fact that the state, the input as well as the output dimension might change over time. This structure leads to a sequentially partitioning of the matrix as exemplary illustrated in Fig. 1. There are also other definitions for semiseparable matrices [87], for example, for quasiseparable matrices. *A matrix S is called a quasiseparable matrix if all the subblocks taken out of the strictly lower triangular part of the matrix (respectively the strictly upper triangular part) are of rank 1.*

Special Structures

The class of semiseparable matrices can be seen as collection of slightly different definitions for semiseparability [87], such as sequentially semiseparable, generator-representable semiseparable, semiseparable plus diagonal and quasiseparable matrices. For example, the class of semiseparable plus diagonal matrices extends the class of semiseparable matrices by adding a diagonal to the semiseparable matrix. The set of generator-representable semiseperable matrices includes all matrices, where

the upper and lower triangular parts are coming from a rank 1 matrix (in contrast to general semiseparable matrices, where the sub-blocks of the lower or upper triangular matrix may come from different rank 1 matrices). Another class of semiseparable matrices are hierarchically semiseparable matrices [87], which are closely connected with the class of hierarchical matrices introduced in Sect. 2.3. Examples for special matrices belonging to the class of semiseparable matrices are band matrices [24] or their inverses [75].

Efficient Algorithms
By exploiting the semiseparable structure, the number of operations for computing the matrix vector product can usually be reduced to $\mathcal{O}(nd^2)$, where d is the maximum state dimension

$$d = \max_k(\max(\dim(x_k), \dim(\hat{x}_k))). \tag{7}$$

This reduction comes from an efficient computational scheme exploiting the sequential structure, which is based on systematically using intermediate results of matrix–vector products of the submatrices. Depending on the structure at hand, there are numerous other fast algorithms available, which may not apply for the general class of semiseparable matrices. A rigorous historic overview of the results found for the class of semiseparable matrices is given by Vandebril et al. [86]. For example, there is a fast algorithm for calculating the inverse of a generator representable plus diagonal semiseparable matrix [23].

Application to Neural Networks
Kissel et al. [51, 52, 52] analyzed the effect of using sequentially semiseparable weight matrices in neural networks. They introduced the *Backpropagation through states* algorithm [51], which can be used to train neural networks with sequentially semiseparable weight matrices. Moreover, they showed how trained weight matrices can be approximated with sequentially semiseparable matrices [50, 52]. Their experiments showed that depending on the task at hand, neural networks with sequentially semiseparable weight matrices are able to outperform their standard counterparts in terms of generalization performance [51].

2.2 Matrices of Low Displacement Rank

The class of matrices with Low Displacement Rank (LDR) [67] unifies the probably most prominent matrix structures, including Toeplitz, Hankel, Vandermonde and Cauchy matrices. The idea of a displacement representation originates from modeling stochastic signals, which may exhibit mild forms of non-stationarity, leading to notions such as Toeplitz-like or Hankel-like displacements [67].

Definition
For analyzing the displacement rank [67] of a matrix M, either the displacement operators of the Sylvester type

$$L(M) = \nabla_{A,B}(M) = AM - MB, \tag{8}$$

$$\begin{pmatrix} u_0 & v_0 & v_1 & v_2 & v_3 & v_4 \\ u_1 & u_0 & v_0 & v_1 & v_2 & v_3 \\ u_2 & u_1 & u_0 & v_0 & v_1 & v_2 \\ u_3 & u_2 & u_1 & u_0 & v_0 & v_1 \\ u_4 & u_3 & u_2 & u_1 & u_0 & v_0 \\ u_5 & u_4 & u_3 & u_2 & u_1 & u_0 \end{pmatrix}$$

(a) Toeplitz Matrix $M_T(u, v)$

$$\begin{pmatrix} v_4 & v_3 & v_2 & v_1 & v_0 & u_0 \\ v_3 & v_2 & v_1 & v_0 & u_0 & u_1 \\ v_2 & v_1 & v_0 & u_0 & u_1 & u_2 \\ v_1 & v_0 & u_0 & u_1 & u_2 & u_3 \\ v_0 & u_0 & u_1 & u_2 & u_3 & u_4 \\ u_0 & u_1 & u_2 & u_3 & u_4 & u_5 \end{pmatrix}$$

(b) Hankel Matrix $M_H(u, v)$

$$\begin{pmatrix} 1 & u_0 & u_0^2 & u_0^3 & \dots & u_0^{n-1} \\ 1 & u_1 & u_1^2 & u_1^3 & \dots & u_1^{n-1} \\ 1 & u_2 & u_2^2 & u_2^3 & \dots & u_2^{n-1} \\ 1 & u_3 & u_3^2 & u_3^3 & \dots & u_3^{n-1} \\ 1 & u_4 & u_4^2 & u_4^3 & \dots & u_4^{n-1} \\ 1 & u_5 & u_5^2 & u_5^3 & \dots & u_5^{n-1} \end{pmatrix}$$

(c) Vandermonde Matrix $M_V(u)$

$$\begin{pmatrix} \frac{1}{u_0-v_0} & \frac{1}{u_0-v_1} & \frac{1}{u_0-v_2} & \frac{1}{u_0-v_3} & \frac{1}{u_0-v_4} \\ \frac{1}{u_1-v_0} & \frac{1}{u_1-v_1} & \frac{1}{u_1-v_2} & \frac{1}{u_1-v_3} & \frac{1}{u_1-v_4} \\ \frac{1}{u_2-v_0} & \frac{1}{u_2-v_1} & \frac{1}{u_2-v_2} & \frac{1}{u_2-v_3} & \frac{1}{u_2-v_4} \\ \frac{1}{u_3-v_0} & \frac{1}{u_3-v_1} & \frac{1}{u_3-v_2} & \frac{1}{u_3-v_3} & \frac{1}{u_3-v_4} \\ \frac{1}{u_4-v_0} & \frac{1}{u_4-v_1} & \frac{1}{u_4-v_2} & \frac{1}{u_4-v_3} & \frac{1}{u_4-v_4} \end{pmatrix}$$

(d) Cauchy Matrix $M_C(u, v)$

Fig. 2 Schematic drawings of the most popular low displacement rank special cases. The displacement structure can be seen in all four matrices: The same values (or modified values) appear in different positions of the matrices

or of the Stein type

$$L(M) = \Delta_{A,B}(M) = M - AMB, \tag{9}$$

can be used. A and B are operator matrices defining the displacement. A matrix has low displacement rank if the displacement matrix $L(M)$ is of low rank. There exists an abundance of possible definitions for the displacement operators and hence this class is quite big.

Efficient Algorithms

There exist efficient algorithms for certain tasks given that the rank of the displacement matrix $L(M)$ is small. This is based on the assumption that the matrix can be *compressed* using the displacements, and that operations can be performed faster using the compressed version. The original matrix can be *recovered (decompressed)* from the displacements. The overall operation scheme can be described as $Compress \rightarrow Operate \rightarrow Decompress$. By exploiting this scheme, inter alia the matrix–vector multiplication can be made more efficient. This leads, for example, to $\mathcal{O}(n \log(n))$ operations for Toeplitz and Hankel matrices, and $\mathcal{O}(n \log^2(n))$ operations for Vandermonde and Cauchy matrices in order to compute the matrix–vector product of a matrix $M \in \mathbb{R}^{n \times n}$ with an arbitrary n-dimensional vector [80].

Table 1 Operator matrices and rank of the corresponding displacements for Toeplitz, Hankel, Cauchy and Vandermonde matrices

Matrix type	Symbol	A	B	$\text{Rank}(\nabla_{A,B}(M))$
Toeplitz matrix	$M_T(u, v)$	Z_1	Z_0	≤ 2
Hankel matrix	$M_H(u, v)$	Z_1	Z_0^T	≤ 2
Vandermonde matrix	$M_V(u)$	$D(u)$	Z_0	≤ 1
Cauchy matrix	$M_C(u, v)$	$D(u)$	$D(v)$	≤ 1

Operator matrices are given with respect to the Sylvester displacement (Eq. 8)

Special Structures

The most popular special members of this structure class are Toeplitz, Hankel, Vandermonde and Cauchy matrices (depicted in Fig. 2). For these special cases, the operator matrices are based on f-circulant matrices

$$Z_f = \begin{pmatrix} 0 & & & f \\ 1 & \ddots & & \\ 0 & \ddots & \ddots & \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{10}$$

or diagonal matrices $D(u)$ defined by the vector u. For example, the operator matrices for a Toeplitz matrix $M_T(u, v)$ as depicted in Fig. 2 with respect to the Sylvester displacement operator are $A = Z_1$ and $B = Z_0$. Hence, the displacement matrix for a Toeplitz matrix $L(M_T(u, v))$ is given by

$$L(M_T(u, v)) = \nabla_{Z_1, Z_0}(M_T(u, v)) = Z_1 M_T(u, v) - M_T(u, v) Z_0. \tag{11}$$

For all Toeplitz matrices $M_T(u, v)$, the rank of the displacement matrix $L(M_T(u, v))$ fulfills

$$\text{rank}(L(M_T(u, v))) \leq 2. \tag{12}$$

The displacement operators for the other special cases are given in Table 1.

Application to Neural Networks

There are several approaches in literature using matrices of low displacement rank in neural networks. The most prominent example is the Convolutional Neural Network (CNN) architecture [66], which is based on sparse Toeplitz Matrices. Convolutional Neural Networks are due to their efficiency and prediction performance the number one choice in machine learning tasks related to images nowadays [45, 53, 79]. In CNNs, the structure is usually encoded implicitly by the connections between the neurons. There are also interesting approaches for improving traditional CNNs. For example, Quaternion CNNs [34, 68, 95] perform operations on images represented in the quaternion domain, which enables them to outperform standard real-valued CNNs on several benchmark tasks. Other approaches focused on matrix structures apart from neural network architectures. Liao and Yuan proposed to use matrices with

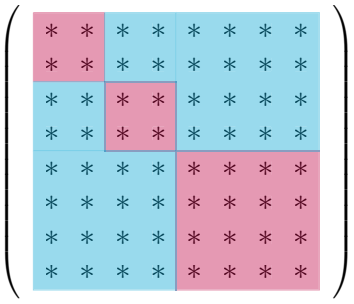

Fig. 3 Schematic Illustration of a (very simple) hierarchical matrix. The cyan parts are low-rank submatrices (admissible blocks), and the purple parts are full-rank submatrices (inadmissible blocks)

a circulant structure in Convolutional Neural Networks [60] and Cheng et al. replaced the linear projections in fully connected neural networks with circulant projections [13]. Appuswamy et al. [4] combined the efficient weight representation used in neuromorphic hardware with block Toeplitz matrices arising in discrete convolutions, which resulted in a family of convolution kernels that are naturally hardware efficient. It has also been proposed to replace weight matrices with general matrices of low displacement rank in neural networks. For example, Sindhwani et al. [80] used Toeplitz-like weight matrices, which include inter alia circulant matrices as well as Toeplitz matrices and their inverses. Moreover, Thomas et al. [84] introduced a class of low displacement rank matrices for which they trained the operators as well as their low-rank components in the neural network. Other works investigate the theoretical properties of neural networks with weight matrices of low displacement rank. For example, Zhao et al. [94] inter alia showed that the universal approximation theorem holds for these networks. Another proof showing that the universal approximation theorem holds for neural networks comprising Toeplitz or Hankel weight matrices is given by Liu et al. [61]. Their approach can be viewed as a Toeplitz-, Hankel-, or LU-based decomposition of neural networks. In particular, they present two proofs for the universal approximation theorem: One for neural networks with fixed depth and arbitrary width, and a second for neural networks with fixed width and arbitrary depth.

2.3 Hierarchical Matrices

Hierarchical matrices ($\mathcal{H}$-matrices) are based on the principle, that even if the overall matrix does not have a low rank, there might still be low-rank sub-blocks in the matrix. Therefore, the idea is to partition a matrix into sub-matrices using suitable (potentially complex) index sets and exploit the low-rank structure of the sub-matrices in this decomposition.

Definition

$\mathcal{H}$-matrices are defined by block cluster trees [9, 41, 43]. The block cluster tree decomposes the matrix into admissible and non-admissible blocks. Being admissible means that the regarded block has a low-rank structure, and therefore can be decomposed

into two matrices with at most rank r (with r being smaller than the dimensions of the block). The overall aim is to find a block cluster tree for a given matrix, such that large parts of the matrix can be approximated by low-rank matrices (and still be close to the original matrix). In Fig. 3, an example partitioning of a hierarchical matrix is depicted. In order to determine the block cluster tree, first the row and column indices of the regarded matrix are organized in cluster trees, i.e., set decomposition trees for the row and column index sets of the matrix. This can, for example, be done by geometric bisection or regular subdivision. Based on these cluster trees, the block cluster tree can be defined by forming pairs of clusters on the cluster trees recursively. The number of leaves in the block cluster tree determines the complexity for arithmetic operations. Therefore, while constructing the block cluster tree, it is desirable to ensure that blocks become admissible as soon as possible. Using these building blocks, $\mathcal{H}$-matrices are defined as follows [43]. Let $L \in \mathbb{R}^{I\times I}$ be a matrix and $\mathcal{T}_{I\times I}$ a block cluster tree for L consisting of admissible and non-admissible leaves. L is called $\mathcal{H}$-matrix of blockwise rank r, if for all admissible leaves $B \in \mathbb{R}^{\tau\times\sigma}$ defined by $\mathcal{T}_{I\times I}$

$$\operatorname{rank}(B) \leq k, \tag{13}$$

with $k \in \mathbb{N}$.

Efficient Algorithms

There are fast algorithms for the different sub-classes and special forms of this structure class. Moreover, for general $\mathcal{H}$-matrices, there is a fast algorithm for matrix–vector multiplication ($\mathcal{O}(kn\log(n))$ under moderate assumptions) [41, 43]. Efficient algorithms for arithmetic operations with $\mathcal{H}$-matrices exploit the fact, that the matrix is sub-divided into admissible and non-admissible smaller block-matrices. Based on this decomposition, arithmetic operations can be conducted faster by exploiting the low-rank structure of admissible blocks. The overall result can then be obtained by combining the results from the sub-blocks.

Special Structures

The class of $\mathcal{H}$-matrices unifies several other structures based on hierarchical decompositions. These classes include [2] hierarchically off-diagonal low-rank matrices (HOLDR) [3], hierarchically semi-separable matrices (HSS) [11, 87], $\mathcal{H}^2$-matrices [41, 42], and matrices based on the fast multipole method (FMM) [5, 6, 18, 30, 39, 40]. The relationships of the subclasses to each other as well as their separation from each other are described in [3].

Application to Neural Networks

Fan et al. [27] proposed to use hierarchical matrices in neural networks, which results in a multiscale structure inside the neural network. Later, they extended their approach to $\mathcal{H}^2$-matrices, which led to comparable results as with their original approach, but reduced number of parameters. Chen et al. [12] proposed to approximate the Generalized Gauss–Newton Hessian by a hierarchical matrix, which can be used during training, for analyzing neural networks, estimating learning rates or other applications. Hierarchical matrix approaches have also been used to analyze and compress trained neural networks. For example, Ithapu used a multi-resolution matrix factorization to

analyze the inter-class relationships of deep learning features [48]. Wu et al. [89] applied the Hierarchical Tucker decomposition [38] to neural networks in order to compress fully connected layers as well as convolutional kernels. They argued that the hierarchical matrix format obtained by the Hierarchical Tucker decomposition has advantages for compressing weight matrices in fully connected layers compared to the Tensor Train decomposition, which has been used before. Another approach is to use wavelets in neural networks [20, 25, 26, 32, 49, 71, 74, 93]. The resulting networks are called *wavelet networks* and make the time-frequency zooming property of wavelets usable in neural networks [49]. Wavelet networks are often constructed from multiresolution analysis or multiresolution approximation [25, 26, 49].

2.4 Products of Sparse Matrices

The structure classes presented in the previous sections represent data-sparse matrices. In contrast, the focus of this section are products of *sparse* matrices. While data-sparse matrices may be full matrices, i.e., all n^2 matrix entries are different from zero, we talk of sparse matrices if the matrices only contain few non-zero entries (for example, $\mathcal{O}(n)$ non-zero entries) [76]. This is an extremely important class of matrices with numerous applications and a long tradition. Exploiting the zero entries directly leads to faster algorithms for several arithmetic operations, since operations can potentially be omitted. This class is somewhat different then the ones mentioned before, as this type of sparse structure does not lend itself well for an algebraic characterization.

The product of sparse matrices is not sparse in general. Therefore, even many dense matrices can be represented as product of sparse matrices. For well known fast linear transforms, such as the Fast Fourier Transform [15], the Discrete Wavelet Transform [62] or the Hadamard transform [77], there is a structured representation as product of sparse matrices [1, 55]. In fact, the notion of sparsity and structure in linear maps seems to be fundamentally linked [16, 19]. It follows, that all efficient matrix–vector multiplication algorithms can be factorized into products of sparse matrices. The conclusion from these results is [17] that all forms of structure are captured by the representation of linear maps as product of sparse matrices (supported by results from arithmetic circuit theory [10]).

Definition

Sparse Matrices comprise only few nonzero elements [76]. This definition is somewhat vague, but in general the resulting fast algorithms are faster the fewer nonzero entries the matrix has. An example of a sparse matrix is depicted in Fig. 4. The sparsity pattern of a sparse matrix can either be structured (i.e., the nonzero elements are distributed following a regular pattern) or unstructured (with irregularly located nonzero entries). There are different storage schemes which can be used to store sparse matrices. Selecting the right storage scheme is crucial for implementing fast algorithms and depends on the application at hand (more specifically the arithmetic operations which should be performed with the sparse matrix as well as the sparsity pattern at hand). Popular storage scheme examples are the coordinate, compressed sparse row as well as the compressed sparse column matrix format. For example, the coordinate format consists of three arrays. The first array contains the values of the nonzero entries in

Fig. 4 Schematic example of a sparse matrix. Most of the entries are zero. The few non-zero elements are distributed without (obvious) regularity within the matrix

$$\begin{pmatrix} * & 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ * & 0 & * & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & * \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 \end{pmatrix}$$

the matrix, whereas the second and third array contain the row and column indices of the positions of these values in the matrix respectively.

Efficient Algorithms

Bounds on the complexity of efficient algorithms for sparse matrices depend on the number of non-zero elements in the matrix as well as the pattern of their distribution. Depending on the number of non-zero entries, there are fast algorithms for computing the matrix vector product. This does also apply for the product of sparse matrices, such that the number of operations for multiplying the product of sparse matrices with an arbitrary vector are proportional to the number of nonzero elements in the sparse matrices [16]. Fast algorithms for sparse matrix vector multiplication might suffer from several memory accessing problems [37]. This includes for example the irregular memory access for the vector with which the sparse matrix is multiplied [83] or the indirect memory references in the sparse matrix (due to the fact that only the non-zero elements of the matrix are stored) [73]. Since these problems can have a significant influence on the performance of considered arithmetic operations with sparse matrices, there have been several approaches proposed to overcome these problems [28, 35, 47, 72, 85] or tune sparse matrices for specific hardware [7, 29, 64].

Special Structures

A special form of sparse matrices are Butterfly matrices [59, 69], which encode the recursive divide-and-conquer structure of the Fast Fourier Transform [17]. Butterfly matrices are composed as a product of butterfly factor matrices. Kaleidoscope matrices [17], in turn, are the product of butterfly matrices. Dao et al. proposed Kaleidoscope matrices, because in general, it is difficult to find the best sparsity pattern for the sparse matrix factorization (since this is a discrete, non-differentiable search problem). They showed that Kaleidoscope matrices have a similar expressivity as general products of sparse matrices and that various common structured linear transforms lie within this structure class.

Application to Neural Networks

Sparsity has probably been the first structure applied to neural networks. Obtaining sparse weight matrices has for example been addressed by Hassibi and Stork [44] and Le Cun et al. [57]. Their approaches used information from second-order derivatives

in order to remove unimportant weights from the network. More recent work uses group lasso regularization for structured sparsity learning [88], pruning techniques [8], hand-tuned heuristics [21] or obtain sparse neural networks by chance [31]. Products of sparse matrices have also been used in neural networks. In Butterfly networks [1, 58], the inputs to a neural network are connected to the outputs of the network using the butterfly structure. It has been shown, that the regular Convolutional Neural Network architecture is a special Inflated-Butterfly-Net (where inflated means that there are dense cross-channel connections in the network) [91]. Moreover, Li et al. [58] showed that the approximation error of Butterfly networks representing the Fourier kernels exponentially decays with increasing network depth. Dao et al. [16] also incorporated Butterfly matrices into end-to-end Machine Learning pipelines and showed that they were able to recover several fast algorithm such as the Discrete Fourier Transform or the Hadamard Transform. To overcome the non-differentiable search problem of finding the right sparsity pattern, Dao et al. [17] proposed to use Kaleidoscope matrices in neural networks. By that, the optimization problem remains differentiable while having similar expressiveness as general sparse matrix products. Giffon et al. [36] showed that replacing weight matrices in deep convolutional neural networks by products of sparse matrices can yield a better compression-accuracy trade-off than other popular low-rank-based compression techniques. Their approach is based on the algorithm proposed by Magoarou et al. [55], which finds a sparse matrix product approximation of a given matrix using projected gradient steps.

3 Relations and Comparison

3.1 Structure Classes Overview and Boundaries

After introducing the four main structure classes, we give an overview over the sub-classes, which are contained in the main structure classes. Moreover, we show that the boundaries between the structure classes are not strict, which means that some matrices can be represented in the methodology of different structure classes.

We consider the four structure classes presented in the previous chapters as the main classes of structured matrices. These classes can be used to categorize particular matrix structures which can be found in literature. Since research about structured matrices is fragmented and approaches originate from different fields, there are sub-classes which are special cases of the four main structure classes. The relations of these sub-classes are depicted in Fig. 5.

Even though the four main structure classes are based on different mathematical concepts, there are still matrix classes that can be efficiently represented in multiple structure frameworks. Low-rank matrices are an example of this. These can be represented as semiseparable matrices (since the blocks taken out of a low-rank matrices are again of low rank), hierarchical matrices (by decomposing the whole matrix into a single admissible block), as well as matrices with low displacement rank [84]. Moreover, a rank r matrix $A \in \mathbb{R}^{n\times n}$ with $A = EP^T$ can straightforwardly be represented by a product of two sparse matrices $A = VM$ with $V, M \in \mathbb{R}^{n\times n}$ by setting the first r columns of V to E (and the first r rows of M to P^T respectively).

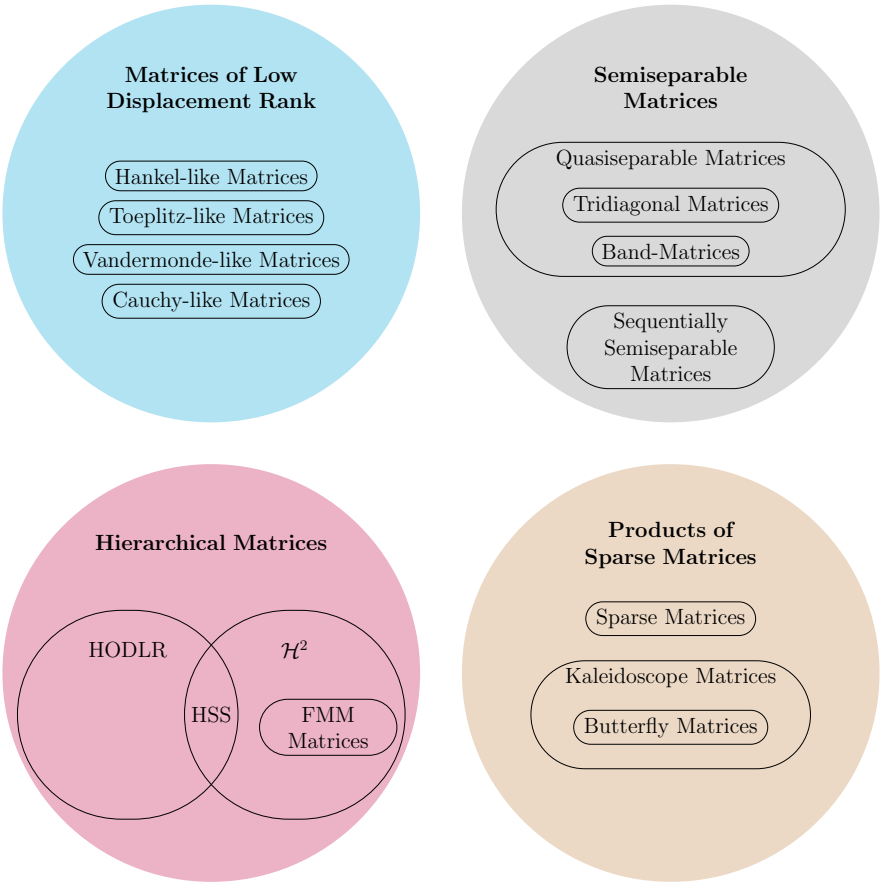

Fig. 5 Overview over the four main structure classes and structure sub-classes which they contain. The four main classes generalize concepts and approaches of special structure classes, which originated from different fields. The part about hierarchical matrices is redrawn after [2]

3.2 Benchmark: Test Matrix Approximation

One use case is that an arbitrary matrix is given, which is to be approximated with a structured matrix. If the approximated matrix is sufficiently close to the original matrix (in a metric suitable for the problem), then the original matrix can be replaced by the structured matrix. Thus, memory and potentially computational resources can be saved. In the domain of neural networks, this means that a weight matrix from a trained network is investigated to check if it possesses a certain structure. If a structure is (approximately) present, then the original weight matrix can be replaced with the new weight matrix represented in the structured matrix framework. The predictions of the neural network are then ideally similar to those before the modification, but memory and computational resources are saved.

Which structure is suited best for approximation depends on the task at hand as well as the selected metric. In this section, we give an overview over the approximation capabilities of the structure classes for different test matrices. We use the Frobenius norm as a metric for how close the approximated matrix is to the original, since this has been found to be a good surrogate for comparing weight matrices [52]. With our benchmark, we aim to give a notion in which structure classes are particularly suitable for approximating certain matrix types. However, this cannot be seen as a conclusive assessment that one structure class is always preferable to another. The choice of the right structure class still depends on the task and context at hand.

We use the following test matrices in our benchmark:

- Random Matrices (with randomly uniform distributed entries in the range $[-1, 1[$)
- Orthogonal Matrices
- Low Rank Matrices
- Matrices with linearly distributed singular values (in the interval [0.1, 1.0]).
- Sequentially Semiseparable Matrices (with statespace dimension set to 5)
- Products of Sparse Matrices (comprising 3 matrices each with 90% sparsity respectively)
- Hierarchical Matrices (with geometrically inspired block cluster trees as introduced in [42] with $\eta = 0.5$)
- Matrices with low displacement rank (Toeplitz, Hankel and Cauchy matrices)
- Weight matrices from Imagenet-pretrained vision models provided by PyTorch [70] (GoogleNet, InceptionV3, MobilenetV2, and Resnet18)

For each of the test matrix classes, we instantiate 3 matrices of shape 300×300 (except for the weight matrices taken out of the vision models), and approximate them using structured matrices of the presented classes. The code used for running our experiments and our test matrices (together with the scripts for generating them) are available on GitHub.[2]

For approximating the test matrices with sequentially semiseparable matrices, we use the approach described in [52] (performing a hyperparameter search for different number of stages), using the TVSCLib[3] implementation. Also, the approximation for products of sparse matrices is based on the approach presented in [52], which is in turn based on an algorithm proposed by Magoarou and Gribonval [55]. We treat the number of sparse factors as well es the sparsity distribution across the factors as hyperparameters, for which we perform a search. Our implementation for the $\mathcal{H}$-matrix approximation uses a greedy approach for assigning low-rank components to the leaf nodes of a block cluster tree. The block cluster tree is treated as hyperparameter, where we compared the admissibility criterion from Hackbusch and Börm [42] (for different values of η) with the approach of building block cluster trees with equally distributed low-rank patches of same size. For approximation with matrices of low displacement rank, we try multiple approaches. First, we investigate the approach presented in [52], which finds an approximation based on gradient-descent updates for the displacements as well as the operator matrices. Second, we employ a direct approximation scheme

[2] https://github.com/MatthiasKi/structurednets.

[3] https://github.com/MatthiasKi/tvsclib.

using fixed operator matrices for Toeplitz-like matrices inspired by Sindhwani et al. [80]. After applying the operator matrices, we find the truncated displacements by performing a Singular Value Decomposition (SVD) on the original displacements. We also show the approximation result for low-rank matrices as a baseline. This approximation is also based on the SVD.

As expected, the approximation error becomes smaller if more parameters are used for approximating the given matrix. Moreover, the approximation algorithms perform particularly good if the investigated matrix has the structure which is used by the approximation approach. For the methods we compared, the approximation approach of using products of sparse matrices resulted in consistently good results for all test matrices. This supports results from Dao et al. [17], stating that the structure class of products of sparse matrices is very powerful for approximating structured transforms. The results of our benchmark are depicted in Fig. 6.

For the approximated weight matrices of PyTorch vision models, we draw a similar conclusion. The products of sparse matrices achieved the best approximation results. This is in line with the findings in [52], where this observation has already been made for smaller weight matrices. For the considered weight matrices, using $\mathcal{H}$-matrices for approximation does not seem to provide much advantage over our baseline, low-rank matrices. In all cases considered, both produce similar approximation results. The approximation with sequentially semiseparable matrices led to the worst results. This was also observed in earlier experiments with smaller weight matrices [52].

We did not include the results for using matrices of low displacement rank in the plots for two reasons. First, the methods given in literature refer to square matrices, which renders them inapplicable for the considered weight matrices. This is not a general limitation, since the framework of matrices with low displacement rank is also applicable to non-square matrices [84]. However, the given algorithms for using matrices of low displacement rank, for example, for recovering a matrix from its displacements, cannot trivially be extended to non-square matrices. Second, the approach introduced by Kissel et al. [52] for approximating square weight matrices using matrices of low displacement rank is only practically usable for small matrices. This is, because the algorithm consumes too much memory and computing resources when the matrices are large (which is the case in our benchmark). Using less sophisticated approaches with fixed operator matrices (for example for Toeplitz-like or Hankel-like matrices) resulted in bad approximation results for all test matrices, except for the ones with the corresponding structure. Therefore, we conclude that the design of practically usable algorithms for the approximation of low displacement rank matrices is still an open task. However, note that apart from *approximating* given matrices, there are efficient algorithms for *training* (square) weight matrices with low displacement rank from scratch [80, 84].

Note that the approximation algorithms used in our benchmark are subject to ongoing research, and for each class there is still a lot room for improvement. Our goal was to show a fair comparison in which the hyperparameters of the individual approaches were tuned with comparable effort. Therefore, it is totally possible that improving the approximation algorithm for one of the structure classes (or developing better heuristics for finding hyperparameters) might render it superior to all other classes in the future.

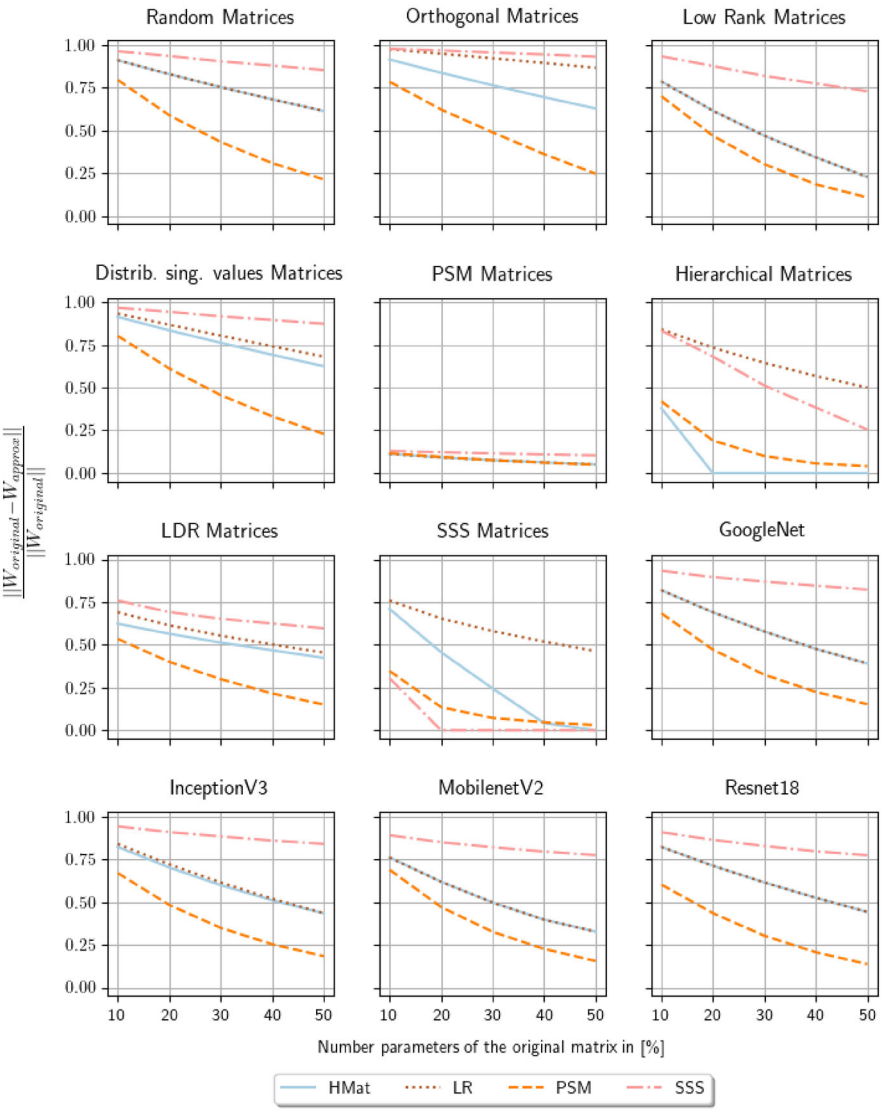

Fig. 6 Results of the approximation benchmark: We approximated several test matrices with structured matrices of different classes, namely hierarchical matrices (HMat), low-rank matrices (LR), products of sparse matrices (PSM), and sequentially semiseparable matrices (SSS). The approximation error becomes smaller if more parameters are available for approximation. If the test matrix has certain structure, we observe that the approach using the very structure performs best. In all other cases, the products of sparse matrices showed the best approximation capabilities

Table 2 Hyperparameters used during training in our fine-tuning benchmark

Hyperparameter	Value
Patience	2
Min. val. loss improvement	0.01
Loss	Cross entropy
Optimizer	Stochastic gradient descent
Number training runs η	10
Learning rate at run $i = 0, \ldots, \eta - 1$	0.5^i
Batch size	150.000
Number of parameters	$\sim$ 20% of original matrix

3.3 Benchmark: Fine-Tuning

The weight matrices of neural networks are typically trained using gradient-descent (backpropagation). Considering that the backpropagation-based training led to remarkable results for neural networks in the past, we investigate the effects of training a structured weight matrix using gradient-descent. For that, we replace the last layer of pretrained PyTorch vision models by structured matrices of different classes (as explained in the previous section). Then, we fine-tune the weight matrix on the same dataset on which the original model was trained. By that, we can compare the prediction accuracy of the model before and after the fine-tuning.

We report the prediction accuracy results on the validation set, with which the models were trained originally. This validation set is *not* used during our fine-tuning. For the fine-tuning, we use a portion of the training data (randomly split before the training begins) as validation set. This validation set is used to determine when the training stops. We stop the training when the validation loss does not improve by at least 0.01 over 2 steps. For each model, there are 10 training runs based on Stochastic Gradient Descent with different learning rates. We start with learning rate $\alpha = 1$, and multiply the learning rate with 0.5 after each training run. Between training runs, we restore the model with lowest validation loss from the previous training run. All important hyperparameters can be found in Table 2.

The gradients used for training are *not* determined by deriving the prediction loss with respect to the weight matrix entries. Instead, we take the derivative of the prediction loss with respect to the parameters determining the structured weight matrix. Details about how this can be done for sequentially semiseparable weight matrices are given by Kissel et al. [51]. The gradients for the other structures can be determined analogously (in our experiments, we use the PyTorch auto differentiation tools for determining the gradients). The code used for running our experiments is available on Github.[4]

For all models, the fine-tuning was able to improve the prediction accuracy compared to the non-fine-tuned version. The accuracy improvements were smaller for models, which achieved high prediction accuracy directly after approximation. For

[4] https://github.com/MatthiasKi/structurednets.

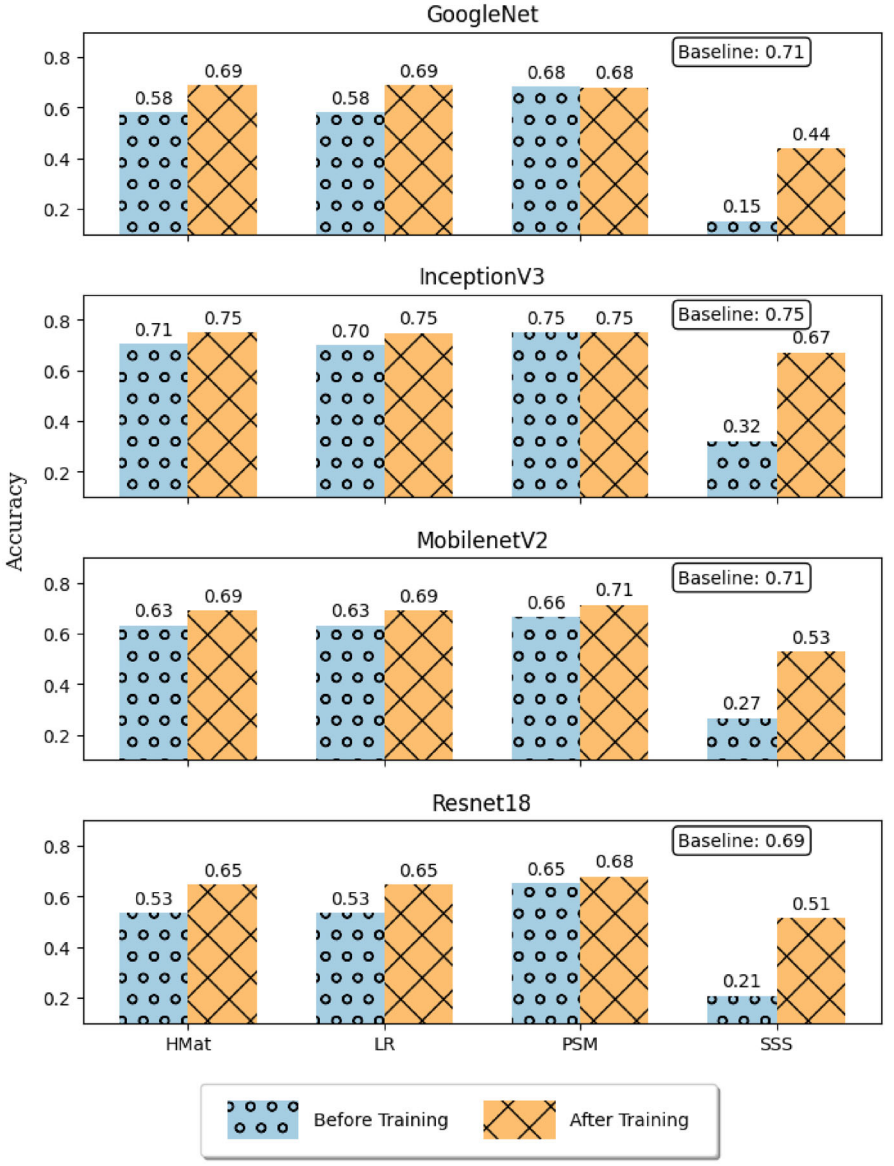

Fig. 7 Accuracy before and after fine-tuning of different PyTorch vision models. The weight matrix of the last layer is replaced with a hierarchical matrix (HMat), a low-rank matrix (LR), products of sparse matrices (PSM), or a sequentially semiseparable matrix (SSS). As expected, the fine-tuning improved the prediction accuracy in all cases. However, for the products of sparse matrices, the improvements are too small to be seen for some models in the figure (supposedly because they already showed good prediction accuracy before fine-tuning). The models with sequentially semiseparable weight matrix showed the biggest improvements. Nevertheless, their final prediction performance remains behind other structures

the products of sparse matrices, the improvements were so small, and that for some models, they are not even visible in Fig. 7.

Analogous to the results in Sect. 3.2, the models with products of sparse matrices achieved the best prediction accuracy after fine-tuning. This resulted in achieving almost the same performance as the baseline models for some of the vision models. Other structured matrices also achieved remarkable results after fine-tuning. This leads to the conclusion that for different types of structured matrices, many of the parameters can be spared while achieving almost the same results as the baseline. In this benchmark, the networks with products of sparse matrices consistently achieved the best results.

The neural networks with sequentially semiseparable weight matrix could not keep up with the performance of the other networks. They showed significant lower prediction accuracy after fine-tuning than the baseline. However, the fine-tuning led to remarkable improvements in the prediction accuracy. In all experiments, the accuracy was more than doubled after fine-tuning, which are much greater improvements than observed with other networks. This is in line with previous results, which showed that approximation of weight matrices with sequentially semiseparable matrices led to poor results [52], but by training such networks from scratch, it was possible to even increasing generalization performance [51].

4 Limitations and Discussion

The presented structures have been applied to neural networks, where they have been used for faster inference, faster training, or for network analysis. However, some questions remain unanswered to this day. In the following, we highlight two research areas in the context of neural networks with structured weight matrices for which we identified relevant unanswered questions.

Theoretical results for the use of structured matrices in neural networks are still very limited. For neural networks with weight matrices of low displacement rank, Zhao et al. [94] proved that the universal approximation theorem still holds and they gave upper bounds for the approximation error. However, proving similar results for other classes of structured matrices is still the subject of ongoing research. In particular, theoretical insights regarding approximation errors for problems with different data distributions can be helpful for selecting a suitable network. For example, they can help to decide whether a large network with structured weight matrices is preferable to a small network with standard weight matrices, depending on the problem at hand. Thus, the first research area we identified is about the question how the performance of neural networks with structured weight matrices depends on the target application. The first intuition is that the choice of a suitable structure used in the network depends very much on the application domain (as indicated by the success of CNNs in image-based domains). To our knowledge, however, this effect has not been explicitly studied yet. We consider our benchmarks as initial insights for selecting an appropriate weight matrix structure. In summary, if there is no indicator that a particular structure is suitable for the given problem, products of sparse matrices are a very good choice. These performed robustly very well in both of our benchmarks. However, we recommend to

perform a hyperparameter search considering different structure classes, if the computational resources needed for the training play a minor role. This hyperparameter search might reveal another structure class that fits the problem at hand particularly well.

The second area we identified is structure-aware training. By this we mean the methodology of how structures can be introduced into the weight matrices of neural networks. In the aforementioned preliminary work, various strategies were pursued in this regard: Regularization techniques, training using backpropagation or approximation of weight matrices with structured matrices after training. But there is still limited knowledge about which method to select for a given problem. Moreover, hybrid approaches for selecting and combining the right methods could be developed. We consider the development of algorithms that find the right structure without hand-tuning and excessive expert knowledge critical to make the overall approach useful for a wide range of problems.

The aim of this paper is to give an overview over the most important structure classes and relevant structure sub-classes. However, it is of course not possible to cover all structures that have ever been studied. Therefore, we would like to mention a few structures that we did not consider.

First, we would like to mention kernel-based approaches. These are not explicit structures, which can be represented by dependencies between the matrix elements. Rather, we consider kernel-based approaches as implicit structures, since operations are spared through the kernel trick. In this context, we consider approaches that learn kernel functions from data [54], kernel-based weight matrices or layers in neural networks [14, 65].

Second, we did not address complex tensor decompositions or factorizations. For example, Yang et al. [92] showed how the adaptive fastfood transform can be used to reparameterize the matrix–vector multiplication in neural networks. Lebedev et al. [56] used a polyadic decomposition (CP decomposition) to decompose convolution kernel tensors into a sum of rank-one tensors. Moczulski et al. [63] replaced linear transformations with a product of diagonal matrices combined with the discrete cosine transform. Their *ACDC* layers can be used to replace any linear transformation in the network and is able to reduce the number of parameters from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ as well as the number of operations from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log(n))$.

5 Conclusion

In this paper, we gave an overview over the four main matrix structures and special sub-classes which they contain. We introduced each of the structure classes by showing their definition, and giving reference to research papers in which the structure is used in the domain of neural networks. Each of the presented structure classes facilitates an efficient matrix–vector multiplication algorithm. Since matrix–vector multiplications are usually the dominant factor for the computational cost of neural networks, using such structures in neural networks has the potential to reduce the required computational cost immensely, finally leading to reduced CO_2 emissions as well as reduced electricity costs.

In the two benchmarks presented in this paper, we compared the approximation capabilities of structured matrices of different classes, as well as the prediction performance of deep vision models containing structured matrices. Products of sparse matrices showed to be the most promising structure class since this structure consistently achieved good results in both benchmarks. However, choosing the right structure still depends on the problem at hand.

Our survey illustrates that the use of structured matrices in neural networks is still a fairly young research area. There are still many open questions, and we presented two research areas we consider most important in the discussion section. These are structure-aware training algorithms as well as analyzing the relationship between structured weight matrices in neural networks and the target application.

Funding Open Access funding enabled and organized by Projekt DEAL. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data availability The code and data used for running the experiments in this paper can be found on GitHub: https://github.com/MatthiasKi/structurednets. This repository uses code from the TVSCLib repository: https://github.com/MatthiasKi/tvsclib. Moreover, the experiments are based on PyTorch vision models, which are available at https://pytorch.org/.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

Research involving human participants and/or animals Not applicable.

Informed consent Not applicable.

References

1. Ailon, N., Leibovitch, O., Nair, V.: Sparse linear networks with a fixed butterfly structure: theory and practice. In: Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, vol. 161, pp. 1174–1184. PMLR (2021)
2. Ambikasaran, S.: Fast algorithms for dense numerical linear algebra and applications. PhD thesis (2013)
3. Ambikasaran, S., Darve, E.: An o (n log n) fast direct solver for partial hierarchically semi-separable matrices. J. Sci. Comput. **57**(3), 477–501 (2013)
4. Appuswamy, R., Nayak, T., Arthur, J., Esser, S., Merolla, P., Mckinstry, J., Melano, T., Flickner, M., Modha, D.: Structured convolution matrices for energy-efficient deep learning. arXiv preprint arXiv:1606.02407 (2016)
5. Beatson, R.K., Newsam, G.N.: Fast evaluation of radial basis functions: I. Comput. Math. Appl. **24**(12), 7–19 (1992)

6. Beatson, R., Greengard, L.: A short course on fast multipole methods. Wavelets Multilevel Methods Elliptic PDEs **1**, 1–37 (1997)
7. Bell, N., Garland, M.: Efficient sparse matrix-vector multiplication on cuda. Nvidia Technical Report NVR-2008-004 **2**(5) (2008)
8. Blalock, D., Ortiz, J.J.G., Frankle, J., Guttag, J.: What is the state of neural network pruning? arXiv preprint arXiv:2003.03033 (2020)
9. Börm, S., Grasedyck, L., Hackbusch, W.: Hierarchical matrices. Lect. Notes **21**, 2003 (2003)
10. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory, vol. 315. Springer Science & Business Media, New York (2013)
11. Chandrasekaran, S., Ming, G., Pals, T.: A fast ULV decomposition solver for hierarchically semiseparable representations. SIAM J. Matrix Anal. Appl. **28**(3), 603–622 (2006)
12. Chen, C., Reiz, S., Yu, C.D., Bungartz, H.-J., Biros, G.: Fast approximation of the Gauss–Newton hessian matrix for the multilayer perceptron. SIAM J. Matrix Anal. Appl. **42**(1), 165–184 (2021)
13. Cheng, Y., Felix, X.Y., Feris, R.S., Kumar, S., Choudhary, A., Chang, S.-F.: Fast neural networks with circulant projections. arXiv preprint arXiv:1502.03436 (2015)
14. Cho, Y.: Kernel methods for deep learning. PhD thesis, UC San Diego (2012)
15. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**(90), 297–301 (1965)
16. Dao, T., Gu, A., Eichhorn, M., Rudra, A., Ré, C.: Learning fast algorithms for linear transforms using butterfly factorizations. In: International Conference on Machine Learning, pp. 1517–1527. PMLR (2019)
17. Dao, T., Sohoni, N., Gu, A., Eichhorn, M., Blonder, A., Leszczynski, M., Rudra, A., Ré, C.: Kaleidoscope: An efficient, learnable representation for all structured linear maps. In: International Conference on Learning Representations (2020)
18. Darve, E.: The fast multipole method: numerical implementation. J. Comput. Phys. **160**(1), 195–240 (2000)
19. De Sa, C., Cu, A., Puttagunta, R., Ré, C., Rudra, A.: A two-pronged progress in structured dense matrix vector multiplication. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1060–1079. SIAM (2018)
20. de Sousa, C., Hemerly, E.M., Galvão, R.K.H.: Adaptive control for mobile robot using wavelet networks. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **32**(4), 493–504 (2002)
21. Dettmers, T., Zettlemoyer, L.: Sparse networks from scratch: faster training without losing performance. arXiv preprint arXiv:1907.04840 (2019)
22. Dewilde, P., Van der Veen, A.-J.: Time-Varying Systems and Computations. Springer Science & Business Media, New York (1998)
23. Eidelman, Y., Gohberg, I.: Inversion formulas and linear complexity algorithm for diagonal plus semiseparable matrices. Comput. Math. Appl. **33**(4), 69–79 (1997)
24. Eidelman, Y., Gohberg, I.: On a new class of structured matrices. Integr. Equ. Oper. Theory **34**(3), 293–324 (1999)
25. Ejbali, R., Zaied, M.: A dyadic multi-resolution deep convolutional neural wavelet network for image classification. Multimed. Tools Appl. **77**(5), 6149–6163 (2018)
26. ElAdel, A., Ejbali, R., Zaied, M., Amar, C.B.: Dyadic multi-resolution analysis-based deep learning for Arabic handwritten character classification. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 807–812. IEEE (2015)
27. Fan, Y., Lin, L., Ying, L., Zepeda-Núnez, L.: A multiscale neural network based on hierarchical matrices. Multiscale Model. Simul. **17**(4), 1189–1213 (2019)
28. Flegar, G., Anzt, H.: Overcoming load imbalance for irregular sparse matrices. In: Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms, pp. 1–8 (2017)
29. Flegar, G., Quintana-Ortí, E.S.: Balanced CSR sparse matrix-vector product on graphics processors. In: European Conference on Parallel Processing, pp. 697–709. Springer (2017)
30. Fong, W., Darve, E.: The black-box fast multipole method. J. Comput. Phys. **228**(23), 8712–8725 (2009)
31. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2018)
32. Galvão, R.K.H., Yoneyama, T.: A competitive wavelet network for signal clustering. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **34**(2), 1282–1288 (2004)

33. Gantmakher, F., Krein, M.: Sur les matrices completement non négatives et oscillatoires. Compos. Math. **4**, 445–476 (1937)
34. Gaudet, C.J., Maida, A.S.: Deep quaternion networks. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2018)
35. Geus, R., Röllin, S.: Towards a fast parallel sparse symmetric matrix-vector multiplication. Parallel Comput. **27**(7), 883–896 (2001)
36. Giffon, L., Ayache, S., Kadri, H., Artières, T., Sicre, R.: Psm-nets: compressing neural networks with product of sparse matrices. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2021)
37. Goumas, G., Kourtis, K., Anastopoulos, N., Karakasis, V., Koziris, N.: Understanding the performance of sparse matrix-vector multiplication. In: 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), pp. 283–292. IEEE (2008)
38. Grasedyck, L.: Hierarchical singular value decomposition of tensors. SIAM J. Matrix Anal. Appl. **31**(4), 2029–2054 (2010)
39. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. J. Comput. Phys. **73**(2), 325–348 (1987)
40. Greengard, L., Rokhlin, V.: A new version of the fast multipole method for the Laplace equation in three dimensions. Acta Numer. **6**, 229–269 (1997)
41. Hackbusch, W.: Hierarchical Matrices: Algorithms and Analysis, vol. 49. Springer, New York (2015)
42. Hackbusch, W., Börm, S.: Data-sparse approximation by adaptive 2-matrices. Computing **69**(1), 1–35 (2002)
43. Hackbusch, W., Grasedyck, L., Börm, S.: An introduction to hierarchical matrices. Math. Bohem. **2**, 101–111 (2002)
44. Hassibi, B., Stork, D.G.: Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. Morgan Kaufmann, Burlington (1993)
45. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
46. Hill, P., Jain, A., Hill, M., Zamirai, B., Hsu, C.-H., Laurenzano, M.A., Mahlke, S., Tang, L., Mars, J.: DeftNN: addressing bottlenecks for DNN execution on GPUS via synapse vector elimination and near-compute data fission. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 786–799 (2017)
47. Im, E.-J.: Optimizing the performance of sparse matrix-vector multiplication. PhD thesis (2000)
48. Ithapu, V.K.: Decoding the deep: Exploring class hierarchies of deep representations using multiresolution matrix factorization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 45–54 (2017)
49. Jemai, O., Zaied, M., Amar, C.B., Alimi, M.A.: Fast learning algorithm of wavelet network based on fast wavelet transform. Int. J. Pattern Recognit. Artif. Intell. **25**(08), 1297–1319 (2011)
50. Kissel, M., Diepold, K.: Deep convolutional neural networks with sequentially semiseparable weight matrices. ESANN 2022 Proceedings (2022)
51. Kissel, M., Gottwald, M., Gjeroska, B., Paukner, P., Diepold, K.: Backpropagation through states: training neural networks with sequentially semiseparable weight matrices. In: Proceedings of the 21st EPIA Conference on Artificial Intelligence (2022)
52. Kissel, M., Gronauer, S., Korte, M., Sacchetto, L., Diepold, K.: Exploiting structures in weight matrices for efficient real-time drone control with neural networks. In: Proceedings of the 21st EPIA Conference on Artificial Intelligence (2022)
53. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. **25**, 1097–1105 (2012)
54. Le, L., Hao, J., Xie, Y., Priestley, J.: Deep kernel: learning kernel function from data using deep neural network. In: Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, pp. 1–7 (2016)
55. Le Magoarou, L., Gribonval, R.: Flexible multilayer sparse approximations of matrices and applications. IEEE J. Select. Top. Signal Process. **10**(4), 688–700 (2016)
56. Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In: 3rd International Conference on Learning Representations, ICLR 2015-Conference Track Proceedings (2015)
57. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605 (1990)

58. Li, Y., Cheng, X., Jianfeng, L.: Butterfly-net: optimal function representation based on convolutional neural networks. Commun. Comput. Phys. **28**(5), 1838–1885 (2020)
59. Li, Y., Yang, H., Martin, E.R., Ho, K.L., Ying, L.: Butterfly factorization. Multiscale Model. Simul. **13**(2), 714–732 (2015)
60. Liao, S., Yuan, B.: Circconv: a structured convolution with low complexity. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 4287–4294 (2019)
61. Liu, Y., Jiao, S., Lim, L.-H.: Lu decomposition and Toeplitz decomposition of a neural network. arXiv preprint arXiv:2211.13935 (2022)
62. Mallat, S.G.: A theory for multiresolution signal decomposition: the wavelet representation. IEEE Trans. Pattern Anal. Mach. Intell. **11**(7), 674–693 (1989)
63. Moczulski, M., Denil, M., Appleyard, J., de Freitas, N., Wang, Z., Zoghi, M., Hutter, F., Matheson, D., de Freitas, N., Reed, S., et al.: Acdc: a structured efficient linear layer. In: International Conference on Learning Representations (ICLR), vol. 55, pp. 1005–1014. Universities of Harvard, Oxford, and Google DeepMind
64. Monakov, A., Lokhmotov, A., Avetisyan, A.: Automatically tuning sparse matrix-vector multiplication for GPU architectures. In: International Conference on High-Performance Embedded Architectures and Compilers, pp. 111–125. Springer (2010)
65. Muller, L., Martel, J., Indiveri, G.: Kernelized synaptic weight matrices. In: International Conference on Machine Learning, pp. 3654–3663. PMLR (2018)
66. O'Shea, K., Nash, R.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 (2015)
67. Pan, V.: Structured Matrices and Polynomials: Unified Superfast Algorithms. Springer Science & Business Media, New York (2001)
68. Parcollet, T., Morchid, M., Linarès, G.: Quaternion convolutional neural networks for heterogeneous image processing. In: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 8514–8518. IEEE (2019)
69. Parker, D.S.: Random butterfly transformations with applications in computational linear algebra (1995)
70. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32, pp. 8024–8035. Curran Associates, Inc. (2019)
71. Pati, Y.C., Krishnaprasad, P.S.: Analysis and synthesis of feedforward neural networks using discrete affine wavelet transformations. IEEE Trans. Neural Netw. **4**(1), 73–85 (1993)
72. Pichel, J.C., Heras, D.B., Cabaleiro, J.C., Rivera, F.F.: Improving the locality of the sparse matrix-vector product on shared memory multiprocessors. In: 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2004. Proceedings, pp. 66–71. IEEE (2004)
73. Pinar, A., Heath, M.T.: Improving performance of sparse matrix-vector multiplication. In: SC'99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing, p. 30. IEEE (1999)
74. Postalcioglu, S., Becerikli, Y.: Wavelet networks for nonlinear system modeling. Neural Comput. Appl. **16**(4), 433–441 (2007)
75. Rózsa, P., Bevilacqua, R., Romani, F., Favati, P.: On band matrices and their inverses. Linear Algebra Appl. **150**, 287–295 (1991)
76. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia (2003)
77. Shanks, J.L.: Computation of the fast Walsh–Fourier transform. IEEE Trans. Comput. **100**(5), 457–459 (1969)
78. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
79. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
80. Sindhwani, V., Sainath, T.N., Kumar, S.: Structured transforms for small-footprint deep learning. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 2, pp. 3088–3096 (2015)

81. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in nlp. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 3645–3650 (2019)
82. Sze, V., Chen, Y.-H., Yang, T.-J., Emer, J.S.: Efficient processing of deep neural networks: a tutorial and survey. Proc. IEEE **105**(12), 2295–2329 (2017)
83. Temam, O., Jalby, W.: Characterizing the behavior of sparse algorithms on caches. PhD thesis, INRIA (1992)
84. Thomas, A.T., Albert, G., Dao, T., Rudra, A., Ré, C.: Learning compressed transforms with low displacement rank. Adv. Neural Inf. Process. Syst. **2018**, 9052 (2018)
85. Toledo, S.: Improving the memory-system performance of sparse-matrix vector multiplication. IBM J. Res. Dev. **41**(6), 711–725 (1997)
86. Vandebril, R., Van Barel, M., Golub, G., Mastronardi, N.: A bibliography on semiseparable matrices. Calcolo **42**(3), 249–270 (2005)
87. Vandebril, R., Van Barel, M., Mastronardi, N.: Matrix Computations and Semiseparable Matrices: Linear Systems, vol. 1. JHU Press, Baltimore (2007)
88. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, pp. 2082–2090 (2016)
89. Wu, B., Wang, D., Zhao, G., Deng, L., Li, G.: Hybrid tensor decomposition in neural network compression. Neural Netw. **132**, 309–320 (2020)
90. Xie, D., Xiong, J., Pu, S.: All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6176–6185 (2017)
91. Xu, Z., Li, Y., Cheng, X.: Butterfly-net2: simplified butterfly-net and Fourier transform initialization. In: Mathematical and Scientific Machine Learning, pp. 431–450. PMLR (2020)
92. Yang, Z., Moczulski, M., Denil, M., De Freitas, N., Smola, A., Song, L., Wang, Z.: Deep fried convnets. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1476–1483 (2015)
93. Zhang, Q., Benveniste, A.: Wavelet networks. IEEE Trans. Neural Netw. **3**(6), 889–898 (1992)
94. Zhao, L., Liao, S., Wang, Y., Li, Z., Tang, J., Yuan, B.: Theoretical properties for neural networks with weight matrices of low displacement rank. In: International Conference on Machine Learning, pp. 4082–4090. PMLR (2017)
95. Zhu, X., Xu, Y., Xu, H., Chen, C.: Quaternion convolutional neural networks. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 631–647 (2018)