Deep Neural Network Compression via Data-Driven Low-Rank Singular Value Decomposition

ABDELFATTAH TOULAOUI*, IDIA Laboratory, National School of Applied Sciences, Sultan Moulay Slimane University, Morocco
HAMZA KHALFI, LaMDa, National School of Applied Sciences, Sultan Moulay Slimane University, Morocco
IMAD HAFIDI, IDIA Laboratory, National School of Applied Sciences, Sultan Moulay Slimane University, Morocco

Deep neural network compression focuses on reducing the number of parameters and computational complexity of neural networks, enabling their use in resource-constrained environments where smaller networks, faster response times, and lower energy consumption are critical. In this paper, we propose a novel compression method that uses Singular Value Decomposition (SVD) informed by data examples. Unlike existing SVD-based approaches that operate solely on network weights, our method leverages input data to more effectively preserve essential information during compression. This data-driven approach significantly enhances performance compared to state-of-the-art techniques. Through experiments on standard benchmark datasets, we show that our method achieves substantial reductions in model size and computation with minimal impact on accuracy. Furthermore, we demonstrate its applicability in compressing deep neural networks to a fraction of their original size, achieving competitive accuracy of networks with a compression ratio of less than 3× with no fine-tuning, and at most 25 epochs of fine-tuning for higher compression ratios.

JAIR Associate Editor: Pablo Samuel Castro

JAIR Reference Format:
Abdelfattah Toulaoui, Hamza Khalfi, and Imad Hafidi. 2026. Deep Neural Network Compression via Data-Driven Low-Rank Singular Value Decomposition. *Journal of Artificial Intelligence Research* 86, Article 26 (July 2026), 28 pages. DOI: 10.1613/jair.1.18694

1 Introduction

In recent years, significant advancements have been made in the field of Deep Learning, leading to state-of-the-art achievements across various domains, including computer vision, time series prediction, and natural language processing (Sarker 2021). These progressions are largely attributed to the availability of extensive and diverse datasets, breakthroughs in data processing algorithms, and an enhanced understanding of the properties inherent to Deep Neural Networks (DNNs). Equally important to these developments is the advent of efficient hardware, which has facilitated the creation of larger models and accelerated training times (Kochura et al. 2019; Krizhevsky et al. 2017). The training and deployment of modern DNNs, however, demand substantial computational resources, a necessity exacerbated by the practice of over-parameterization. This practice, which has been shown to improve

*Corresponding Author.

Authors' Contact Information: Abdelfattah Toulaoui, ORCID: 0009-0005-6094-223X, abdelfattah.toulaoui1@usms.ac.ma, IDIA Laboratory, National School of Applied Sciences, Sultan Moulay Slimane University, Khouribga, Morocco; Hamza Khalfi, ORCID: 0000-0003-2426-8063, h.khalfi@usms.ma, LaMDa, National School of Applied Sciences, Sultan Moulay Slimane University, Khouribga, Morocco; Imad Hafidi, ORCID: 0000-0002-5039-2458, i.hafidi@usms.ma, IDIA Laboratory, National School of Applied Sciences, Sultan Moulay Slimane University, Khouribga, Morocco.

DOI: 10.1613/jair.1.18694

the generalization capabilities of DNNs (Ardalani et al. 2019; Du and J. Lee 2018), presents significant challenges when adapting state-of-the-art performance to low-power devices, due to the limitations in computational power, memory, or energy that plague real world environments. To address these challenges, the compression of DNNs has emerged as a potential solution, gaining considerable momentum in recent years (Deng et al. 2020). This research primarily focuses on enhancing the efficiency of deep neural networks by reducing both the number of parameters (weights) and the number of floating point operations FLOPs of DNNs while preserving their accuracy as much as possible.

Several studies have explored the application of low-rank tensor decomposition as a method for compressing neural networks (Astrid and S.-I. Lee 2018). These approaches involve decomposing each weight tensor into two or more small, which can be stored and used more efficiently. For example, a weight tensor $\mathcal{W}$ can be decomposed into n smaller tensors $\mathcal{W}_0, \mathcal{W}_1, \ldots, \mathcal{W}_n$ where the total elements is smaller than the total elements of $\mathcal{W}$, leading to reduced computational demands and faster operations. The effectiveness of this technique stems from the observation that DNN weights often exhibit low-rank properties (Denil et al. 2013), indicating that they contain less information than their size might suggest. When decomposing a weight tensor into two tensors, the compression ratio achieved by this method is dependent on the decomposition rank k for each layer, which represents the inner dimension of the resulting tensors. Choosing an appropriate rank k is crucial to balancing compression with performance, as it determines the level of information retention within the compressed network. Many decomposition-based methods propose the use of statistical methods to identify the proportion of information carried by each component, such as variational Bayesian matrix factorization (VBMF) (Kim et al. 2015). Other works have proposed the use training time estimation of the ranks on-the-fly, such as with nuclear norm rank minimization factorization (NRMF) (Ran et al. 2021).

Modern deep neural networks achieve remarkable performance at the cost of substantial computational and memory requirements. While network compression techniques address these limitations, they face a fundamental trade-off: aggressive compression typically degrades model accuracy by discarding critical information. Existing matrix decomposition methods optimize this trade-off by analyzing weight tensors in isolation, disregarding the statistical properties of input data distributions. This neglect risks removing parameters essential for processing real-world inputs, necessitating extensive fine-tuning to recover accuracy.

The central challenge lies in determining which components of a neural network's parameter space contain *task-critical information*. We posit that this determination must consider the interaction between network weights and input data patterns. For example, a convolutional filter's importance depends on both its learned features and the frequency of corresponding visual patterns in the input distribution (Martin and Mahoney 2021; Pastur 2022; Zeiler and Fergus 2014; B. Zhou et al. 2016). Traditional weight-based decomposition methods lack this perspective, potentially removing filters crucial for processing common input features.

Our formulation differs fundamentally from activation-oriented frameworks (Swaminathan et al. 2020; Tian et al. 2021). While these prior works minimize feature-map reconstruction error ($\|WX - \hat{W}X\|_F^2$) via a sparse SVD on the structural weight matrix W, they treat activations strictly as a proxy for parameter sensitivity. In contrast, we decouple the decomposition from the static weight matrix boundaries and apply SVD directly to the activation tensor Y itself. By shifting the factorization target from the parameter space (W) to the representation space (Y), our method directly captures low-rank latent dynamics that weight-centric approximations overlook.

We introduce a principled approach that aligns network compression with input data characteristics through output-preserving Singular Value Decomposition (SVD). Our key insight is that decomposing *layer outputs* rather than weights preserves the network's functional behavior under actual operating conditions. Given input examples X, we perform SVD on the induced output matrices $Y^{(j)} = L^{(j)}(X)$ for each layer $L^{(j)}$ with weights $\hat{W}^{(j)}$, then construct low-rank approximations that minimize the Frobenius norm of the error $\|Y^{(j)} - \hat{Y}^{(j)}\|_F$ rather than $\|W^{(j)} - \hat{W}^{(j)}\|_F$, where $\hat{Y}^{(j)}$ and $\hat{W}^{(j)}$ are the outputs and weights of the decomposed layer respectively.

This method allows the layers to be compressed based on the data that flows through them, instead of their inherent structure.

Our principal contributions are as follows:

- **Data-Driven SVD Framework**: A novel decomposition paradigm that processes layer outputs across representative inputs to determine compression parameters. Unlike prior work, our method explicitly minimizes output distortion rather than weight approximation error.
- **Convolutional Architecture Extension**: A structured decomposition scheme for CNNs that separates spatial and channel-wise redundancies using data-driven SVD. We introduce nested SVD stages that compress channel then spatial dimensions.
- **Comprehensive Empirical Validation**: Rigorous evaluation across vision tasks demonstrates our framework's versatility. We show consistent accuracy preservation at 4–8× compression ratios, outperforming traditional methods on various architectures.

The remainder paper is organized as follows. Section 2 is dedicated to a brief review of related works. In Section 3, we present our method including the objective function and the compression algorithm. Section 4 presents the various experiments that were conducted to test our method's efficacy against other recent methods. Finally, in Section 5, a brief conclusion and discussion of future works.

2 Related Works

Neural network compression is a research domain focused on devising methods to enhance the efficiency of existing models, either by reducing their memory footprint, computational complexity, or both (Menghani 2023). DNN compression is typically achieved through the efficient representation of neural network layers. These methods can be categorized into several families based on their respective approaches to the problem.

In their seminal work on DNN compression, (LeCun et al. 1989) identified significant redundancy in network weights. Their research indicated that weights with smaller magnitudes tend to be less critical to the network's performance. This insight led to the development of unstructured pruning techniques. These techniques enable the removal of a substantial number of parameters from the network with minimal impact on performance (J. Liu et al. 2020).

Pruning produces models that utilize sparse weight matrices, necessitating the use of specialized hardware or software capable of efficiently loading and operating on these sparse structures to optimize network performance (Dalton et al. 2015; Zhang et al. 2016).

In addition to unstructured pruning, other researchers have explored structured pruning as a method for neural network compression, often using filter importance (Anwar et al. 2017; Y. He et al. 2017; H. Li et al. 2016), regularization (Yang et al. 2019), filter similarity (G. Li et al. 2025), or other techniques (Z. Liu, J. Li, et al. 2017; Zhao et al. 2025). While other works with methods data instead of simple magnitude statistics, using first-order Taylor expansion applied to feature maps to estimate filter importance for instance (Molchanov et al. 2017).

Structured pruning involves the removal of entire neurons or filters from network layers, thereby enhancing the efficiency of the model. This class of pruning offers the advantage of producing smaller yet dense neural networks, making them more suitable for deployment on conventional consumer hardware, such as CPUs and GPUs (Z. Liu, Sun, et al. 2019). Structured pruning, however, remains subject to many constraints imposed by the network's architecture which necessitates certain trade-offs.

Another family of techniques for neural network compression is quantization, which seeks lower precision representations of the weights that still uphold the model's performance (Banner et al. 2018; B. Liu et al. 2023). Quantization have been proposed for many different applications, such as in compressing diffusion models (Fukushima and Kamata 2025) and large language models (Lang et al. 2024).

Certain quantization methods represent weights as small integers of 8-bits or even 4-bits by leveraging the statistical properties of the weights, facilitating memory efficiency on some devices and utilizing low precision compute units in modern hardware.

Other methods transform the network into a ternary network (B. Liu et al. 2023), wherein each weight w is represented as a ternary variable $w \in \{-1, 0, +1\}$. Quantization's compression capabilities are limited by the type of method used, and although aggressive quantization significantly compresses neural networks, it often results in performance degradation compared to the original model.

Recent studies have investigated the effects of quantization on the loss function (Nahshan et al. 2021), finding that aggressive post-training quantization is feasible when layers are quantized jointly rather than independently. Other approaches have proposed training models in a quantization-aware fashion to enhance performance (Hubara et al. 2018).

A family of methods employing neural low-rank factorization has also been proposed. These methods exploit the mathematical properties of tensor representations of layers to derive a low-rank approximation of each layer. Various algorithms have been tested, including CP decomposition (Astrid and S.-I. Lee 2017), QR decomposition (Schotthöfer et al. 2022), Tucker decomposition (Y. Liu and Ng 2022), and SVD (Chen et al. 2023; Swaminathan et al. 2020; Tian et al. 2021).

One of the earliest works have leveraged Alternating Least Squares (ALS) to remove spatial redundancies in CNNs, this worked by separating the spatial dimensions of convolutions and reducing the reconstruction error (Jaderberg et al. 2014). Their contemporaries, on the other hand, have used SVD to directly approximate the weight tensor by converting it to a 2-dimensional matrix (Denton et al. 2014).

Generally, these methods decompose the weights of a large layer into several smaller layers with low-rank weights, leveraging redundancies in the weights. Numerous works have explored the use of Singular Value Decomposition (SVD) in neural network compression, a matrix decomposition technique that operates on two-dimensional tensors.

SVD decomposes a real or complex matrix M into the product of three matrices U, Σ, and V:

$$M = U\Sigma V^*,$$

where U and V are unitary matrices (orthogonal matrices if M is a real matrix) (Golub and Van Loan 1996), and Σ is a rectangular diagonal matrix containing the singular values of M. V^* is the conjugate transpose of V, and when M is a real matrix $V^* = V^\top$. If M is of size $n \times m$ and rank r, then U and V are of sizes $n \times n$ and $m \times m$, respectively, and Σ is of size $n \times m$ with r non-zero values. The SVD of a matrix is not unique; however, Σ can be chosen such that $\Sigma_{i,i} \geq \Sigma_{i+1,i+1}, \forall i \in \{1, 2, \ldots, \min(n, m)\}$.

SVD is widely used in data science and numerical linear algebra (Stewart 1993) due to its efficacy in dimensionality reduction, data compression, and isolation of independent components. Such uses include reducing dimensions in NLP (Bullinaria and Levy 2012; Ismael Kadhim et al. 2017) and recommender systems (Guan et al. 2017; X. Zhou et al. 2015).

Building upon these classical foundations, recent literature has pivoted toward activation-aware compression to maintain model performance under extreme hardware constraints. In the domains of pruning and quantization, frameworks such as Wanda (Sun et al. 2024) and SmoothQuant (Xiao et al. 2023) have demonstrated that leveraging runtime activation statistics—rather than weights alone—is essential for preserving the nuances of Large Language Models (LLMs).

Similarly, Knowledge Distillation (KD) has evolved to utilize intermediate activation maps to align student-teacher representations more effectively (Heo et al. 2019). Within the specific context of low-rank approximation, modern approaches like asvd (Yuan et al. 2025) have begun incorporating activation scales to weight the importance of specific weight dimensions.

A significant gap remains in the literature: most existing methods treat activations merely as a pre-processing or post-hoc calibration signal to adjust the weight matrix before decomposition. There is a distinct lack of research exploring the direct integration of the activation manifold into the matrix decomposition process itself, where the factorization objective is reformulated to jointly optimize the latent weight structure and dynamic activation variance within a single, unified numerical framework.

3 Proposed Method

In this section, we introduce our Data-Driven Singular Value Decomposition (DDSVD) framework for neural network compression. Our method combines two key concepts to enable practical implementation: representative sampling from active learning to efficiently capture critical data patterns, and incremental SVD algorithms to process data in batched streaming fashion. We first formalize the mathematical foundation of our approach for fully connected layers, then extend it to convolutional architectures. Throughout this section, we maintain consistent notation and emphasize the relationship between theoretical constructs and practical implementation considerations.

3.1 Notational Conventions

Scalars are denoted with lowercase letters (e.g., n), vectors with bold lowercase (e.g., $\boldsymbol{x}$), matrices with uppercase Roman letters (e.g., A), and tensors with calligraphic uppercase (e.g., $\mathcal{W}$). Neural network layers appear as bold capitals (e.g., $\boldsymbol{L}$). For any matrix $A \in \mathbb{R}^{m\times n}$, the sub-matrix $A_{[i:j,k:l]}$ contains rows i through j and columns k through l, while $A_{[:,k]}$ denotes the k-th column vector. The Frobenius norm $\|A\|_F$ will govern our spectral energy retention analysis. M^* and M^T denote the conjugate transpose of matrix M, and the transpose of M respectively. For real matrices (values in $\mathbb{R}$), which is the case for a in this work unless otherwise indicated, we note $M^* = M^\top$ and use them interchangeably.

For neural network weights, we express a linear layer's weights as a matrix $W \in \mathbb{R}^{n\times m}$ and biases $\mathbf{b} \in \mathbb{R}^n$, with m as the input dimension and n the output dimension. Similarly, all convolutional weights biases are written as as tensor $\mathcal{W} \in \mathbb{R}^{h_k\times w_k\times c_{\text{in}}\times c_{\text{out}}}$ and a bias $\mathbf{b} \in \mathbb{R}^{c_{\text{out}}}$, with h_k and w_k as the kernel width and height respectively, and c_{in} and c_{out} are the layer's input and output channels respectively. These notations will be used throughout the work consistently.

3.2 Output-Preserving Decomposition

Traditional weight matrix decomposition methods approximate $W \in \mathbb{R}^{n\times m}$ as a product $W \approx W_1W_2$, optimizing for arbitrary inputs rather than the actual data distribution $\mathcal{D}$. Our approach extends SVD-based decomposition (Gabor and Zdunek 2023) by preserving layer outputs $Y = XW^\top$ for inputs $X \sim \mathcal{D}$ through data-driven SVD. For an input batch $X \in \mathbb{R}^{b\times m}$, the unbiased output $Y = XW^\top \in \mathbb{R}^{b\times n}$ undergoes transposed SVD:

$$Y^\top = U\Sigma V^\top, \quad U \in \mathbb{R}^{n\times n},\ V \in \mathbb{R}^{b\times b},\ \Sigma \in \mathbb{R}^{n\times b},$$

note that we decompose $Y^\top$ instead of Y which is mathematically equivalent but simplified operations. Truncating to the top-k singular values yields compressed components:

$$\bar{\Sigma} = \Sigma_{[1:k,1:k]}, \quad \bar{U} = U_{[:,1:k]}, \quad \bar{V} = V_{[1:k,:]},$$

since $\bar{V}$ will not be used in the following steps, the implementation can be made more efficient by skipping calculation of V. The semi-orthogonal property $\bar{U}^\top\bar{U} = I_k$ enables a two-layer approximation through algebraic manipulation:

$$Y^\top \approx \bar{U}\bar{\Sigma}\bar{V}^\top \approx \bar{U}\bar{U}^\top Y^\top = \bar{U}(\bar{U}^\top W)X^\top.$$

This induces the parameterization $W_1 = \bar{U}^\top W \in \mathbb{R}^{k\times m}$ and $W_2 = \bar{U} \in \mathbb{R}^{n\times k}$, producing the approximated layer output:

$$\boldsymbol{L}(X) \approx (XW_1^\top)W_2^\top + \boldsymbol{b} = X(W_1W_2)^\top + \boldsymbol{b}.$$

The approximation error $\|Y - \hat{Y}\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2$ is directly controlled by our adaptive rank selection criterion. The approximation is exact when the discarded values of Σ are all zeros. Crucially, V does not participate in the decomposition phase and can be discarded, reducing memory overhead.

3.3 Adaptive Rank Selection

The truncation rank k is determined through spectral energy analysis of the singular values σ_i from the layer output decomposition. For a user-specified energy retention ratio $r \in (0, 1)$, we select the smallest k satisfying:

$$k = \min \left\{ p \in \mathbb{N} \;\middle|\; \frac{\sum_{i=1}^{p} \sigma_i^2}{\sum_{j=1}^{n} \sigma_j^2} \geq (1 - r) \right\}. \tag{1}$$

This criterion ensures the compressed layer preserves at least fraction r of the original output's spectral energy while maximizing parameter reduction. The singular values σ_i are obtained directly from the SVD of representative input batches, making the selection adaptive to the true data distribution.

3.4 Convolutional Architecture Extension

For convolutional layers with weights $\mathcal{W} \in \mathbb{R}^{h_k\times w_k\times c_{\text{in}}\times c_{\text{out}}}$, we first express the operation as a tensor contraction. Let $\mathcal{X} \in \mathbb{R}^{b\times h_{\text{in}}\times w_{\text{in}}\times c_{\text{in}}}$ be an input batch. The output at position (i, j, c) is computed as:

$$O_{i,j,c} = \sum_{m=0}^{h_k-1} \sum_{n=0}^{w_k-1} \sum_{d=0}^{c_{\text{in}}-1} \mathcal{W}_{[m,n,d,c]} \cdot \mathcal{X}_{[s_h i+m, s_w j+n, d]},$$

where s_h, s_w denote stride parameters. Reformulating as matrix multiplication via input unrolling:

$$O_{\text{flat}} = X_{\text{col}} W_{\text{flat}} \in \mathbb{R}^{(h_{\text{out}} w_{\text{out}} b)\times c_{\text{out}}},$$

where $X_{\text{col}} \in \mathbb{R}^{(h_{\text{out}} w_{\text{out}} b)\times(h_k w_k c_{\text{in}})}$ contains flattened receptive fields and $W_{\text{flat}} \in \mathbb{R}^{(h_k w_k c_{\text{in}})\times c_{\text{out}}}$ is the reshaped weight tensor.

Applying DDSVD to O_{flat} creates two matrices that can be reshaped into two convolutional weights: a spatial convolution with weight $\mathcal{W}_1 \in \mathbb{R}^{h_k\times w_k\times c_{\text{in}}\times k}$ and a point-wise convolution with weight $\mathcal{W}_2 \in \mathbb{R}^{1\times 1\times k\times c_{\text{out}}}$.

A secondary decomposition phase applies SVD to $\mathcal{W}_1$ after unfolding via the operator $\textbf{UNFOLD}(\mathcal{W}_1) = W_1 \in \mathbb{R}^{(h_k c_{\text{in}})\times(w_k k)}$ similar to the one used in other works (Chen et al. 2023). The unfolding operation is formally defined as:

$$\begin{aligned} \textbf{UNFOLD} : \mathbb{R}^{h_k\times w_k\times c_{\text{in}}\times k} &\longrightarrow \mathbb{R}^{(h_k c_{\text{in}})\times(w_k k)} \\ \mathcal{W}_1 &\mapsto W_1 \end{aligned} \tag{2}$$

where the element-wise mapping for 0-indexed coordinates is given by:

$$(W_1)_{(l\cdot c_{\text{in}}+n),\,(m\cdot k+o)} = (\mathcal{W}_1)_{l,\,m,\,n,\,o} \tag{3}$$

This particular formulation helps separate spatial kernel dimensions, making it easier for SVD to isolate spatial redundancies. Truncating to rank k' via (1) yields:

$$W_1 \approx U\Sigma V^\top = W_1' W_1'', \tag{4}$$

which folds back into spatial convolutions with kernels $h_k \times 1$ and $1 \times w_k$ respectively. We note that k' is chosen using the same method described in 1, with a small ratio $r = 0.01$ to remove the noise, applied to the Σ of the secondary SVD. The folding operators are formally defined as:

$$\begin{aligned} \mathbf{FOLD}_1 : \mathbb{R}^{(h_k c_{\text{in}}) \times k'} &\to \mathbb{R}^{h_k \times 1 \times c_{\text{in}} \times k'} \\ W_1' &\mapsto \mathcal{W}_1' \end{aligned} \tag{5}$$

$$(\mathcal{W}_1')_{l,0,n,p} = (W_1')_{(l \cdot c_{\text{in}} + n),p}, \tag{6}$$

and:

$$\begin{aligned} \mathbf{FOLD}_2 : \mathbb{R}^{k' \times (w_k k)} &\to \mathbb{R}^{1 \times w_k \times k' \times k} \\ W_1'' &\mapsto \mathcal{W}_1'' \end{aligned} \tag{7}$$

$$(\mathcal{W}_1'')_{0,m,p,o} = (W_1'')_{p,(m \cdot k + o)}. \tag{8}$$

This hierarchical decomposition preserves spatial structure while progressively removing redundancies, as visualized in Figure 1.

This process yields three compressed convolutions:

- Vertical convolution $\mathcal{W}_1' \in \mathbb{R}^{h_k \times 1 \times c_{\text{in}} \times k'}$
- Horizontal convolution $\mathcal{W}_1'' \in \mathbb{R}^{1 \times w_k \times k' \times k}$
- Point-wise convolution $\mathcal{W}_2 \in \mathbb{R}^{1 \times 1 \times k \times c_{\text{out}}}$

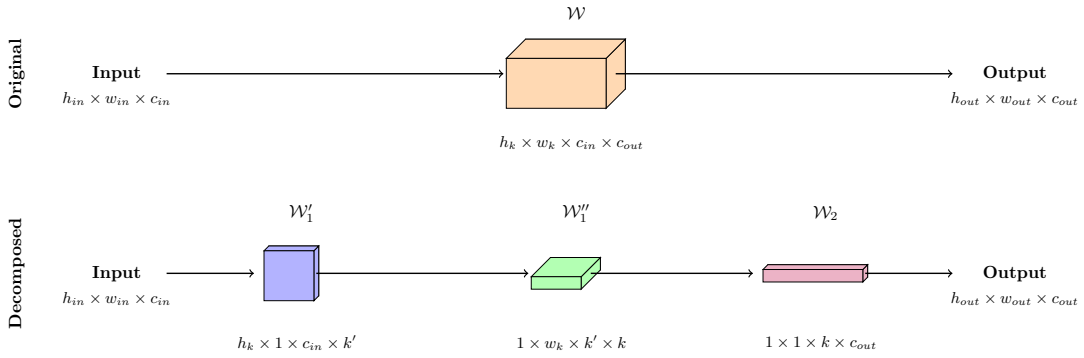

Fig. 1. Convolutional compression via nested decomposition: Original kernel (orange) separates into vertical (blue) and horizontal (green) spatial components via unfolding operations, followed by a point-wise convolution (purple). The three-stage factorization reduces parameters through successive SVD applications while maintaining output fidelity.

3.5 Computational Optimization Strategies

The computational complexity of direct SVD application becomes prohibitive for large neural network layers. Our framework addresses this through two synergistic techniques: representative sampling for dimensionality reduction and incremental SVD for batched processing. Algorithm 2 formalizes this integrated approach.

3.5.1 Representative Sample Selection. Drawing from active learning principles (Jabrane, Hafidi, et al. 2023; Jabrane, Tabbaa, et al. 2024; Mourad et al. 2024; Wang et al. 2015), we employ clustering-based sample selection to identify critical data patterns while discarding redundant observations. For layer outputs $Y \in \mathbb{R}^{N \times d}$ where N exceeds practical decomposition limits, we select $n \ll N$ representative samples through K-Medoid clustering(Kaufman and Rousseeuw 1990; Park and Jun 2009):

Algorithm 1 K-Medoids Clustering with Greedy Dispersion

Require: Dataset X, Number of clusters n
Ensure: Set of medoids M, Cluster assignments C
Select first medoid $m_1 \in X$ randomly
for $i = 2$ **to** n **do**
  $m_i \leftarrow \arg\max_{x \in X} \left(\min_{j<i} d(x, m_j)\right)$ ▷ Select most distant point
end for
$M \leftarrow \{m_1, \ldots, m_n\}$
repeat
  for each sample $x \in X$ **do**
    $C(x) \leftarrow \arg\min_{m \in M} d_{\cos}(x, m)$ ▷ Assign via cosine distance
  end for
  $M_{old} \leftarrow M$
  for each cluster C_k **do**
    $m_k \leftarrow \arg\min_{i \in C_k} \sum_{j \in C_k} d(i, j)$ ▷ Find point with minimum total distance
  end for
  $M \leftarrow \{m_1, \ldots, m_n\}$
until $M = M_{old}$ ▷ Termination

This K-Medoids variant balances computational efficiency with cluster quality. For this work, we define N_{max} as the maximum value of N that can be decomposed directly without representative sampling. For practical purposes, we set $N_{\max}$ to be double the output dimension of Y ($N_{\max} = 2 \cdot d$), which preserves as much information from the matrix after SVD, with larger values not yielding any improvement in performance, while drastically increasing the memory overhead.

3.5.2 Batched Incremental Decomposition. SVD is a memory and compute-intensive process, particularly for larger matrices (Golub and Van Loan 1996). Our proposed method requires a sufficient number of examples to approximate the original model's accuracy closely. To achieve this, we propose using incremental SVD, which operates on chunks of the input matrix along one of its dimensions to approximate its SVD. We utilize an incremental algorithm (Zha and Simon 1999) to find an approximation with an error E:

$$M = U\Sigma V^* = \hat{U}\hat{\Sigma}\hat{V}^* + E.$$

The SVD of a given series of n matrices A_i with $i = 1, 2, \ldots n$ can be approximated by first finding the rank d decomposition of A_1:

$$A_1 = U_1\Sigma_1 V_1^*,$$

and for each subsequent step $k + 1$, we find the QR decomposition of the matrix:

$$(I - U_k U_k^*)A_{k+1} = QR,$$

and finding the rank d singular value decomposition of the matrix:

$$\begin{bmatrix} \Sigma_k & U_k^* A_{k+1} \\ 0 & R \end{bmatrix} = \hat{U}\hat{\Sigma}\hat{V}^*.$$

Finally, the rank d SVD of the matrix at the step $k+1$ can be approximated as:

$$\begin{bmatrix} A_1 & A_2 & \cdots & A_k & A_{k+1} \end{bmatrix} = U_{k+1}\Sigma_{k+1}V_{k+1}^*$$

With:

$$U_{k+1} = \begin{bmatrix} U_k & Q \end{bmatrix} \hat{U}, \tag{9}$$

$$\Sigma_{k+1} = \hat{\Sigma}, \tag{10}$$

$$V_{k+1} = \begin{bmatrix} V_k & 0 \\ 0 & I \end{bmatrix} \hat{V}. \tag{11}$$

In our case, each A_i corresponds to the transposed output $Y_i^{(j)\top}$ of batch i for layer j. While the standard incremental SVD algorithm updates both U_k and V_k in each step, V_k is not a necessary component of the pipeline, and its computation can be ignored to save on memory and time.

With incremental SVD, the approximation of the model's layers can be done in batches. Each batch of inputs is propagated through the network, and the output at each fully-connected or convolutional layer is captured in order to update the approximation of SVD. As such, the decomposition process is made into an iterative one, with the final iteration producing the SVD components required for creating the decomposed layer (W_1 and W_2).

4 Experimental Results

To evaluate the effectiveness of our work, we evaluate DDSVD on several well-known architectures, including ResNet (K. He et al. 2015) and VGG (Simonyan and Zisserman 2015), applied to datasets such as CIFAR (Krizhevsky 2009), and the 1000-class variant of ImageNet (Russakovsky et al. 2015). VGG is particularly noted for its intensive memory requirements due to its large fully connected layers. Table 1 presents basic information on the datasets used for these experiments.

For CIFAR-10 and CIFAR-100, we use smaller variants of ResNet models implemented in PyTorch (Ansel et al. 2024), trained for 200 epochs. The smaller models are chosen because they produce a mode pronounced accuracy reduction in the resulting models, which facilitates comparisons. For ILSVRC2012, we use the pre-trained weights provided by the TorchVision library (maintainers and contributors 2016). Table 2 presents the characteristics of the models used in our evaluation.

Table 1. Basic information on the datasets used for the experiments.

Dataset	Number of Classes	Description
CIFAR-10	10	32x32 RGB images
CIFAR-100	100	32x32 RGB images
ImageNet	1000	Large-scale RGB image classification dataset

We evaluate the methods based on three criteria: accuracy, number of parameters, and FLOPs. Model training, compression, fine-tuning, and evaluation were performed on an NVIDIA RTX 3060 with 12GB of GPU memory.

The proposed approach is compared to TUCKER Decomposition (Kim et al. 2015), and RIGHT JOINT-SVD and LEFT JOINT-SVD (Chen et al. 2023). First, the target models are compressed with DDSVD, then compression ratio (CR) of each model at different values for r is calculated. Finally, the other three methods are applied as described

Algorithm 2 Data-Driven SVD Network Compression

Require:
Neural network with layers $\{\boldsymbol{L}^{(j)}\}_{j=1}^{m}$
Input dataset $\{X_i\}_{i=1}^{N}$ (batches or individual samples)
$N_{\max}$: Maximum rows for decomposition matrix
r: Spectral energy ratio for rank selection
Ensure: Compressed network with decomposed layer structure
Initialize $U^{(j)} \leftarrow \emptyset$, $\Sigma^{(j)} \leftarrow \emptyset$ for all layers j
for each sample $X_i \in \{X_1, \ldots, X_N\}$ **do**
  for each layer $\boldsymbol{L}^{(j)}$ in topological order **do**
    Forward propagate X_i to get layer output $Y_i^{(j)} \in \mathbb{R}^{b_i \times d_j}$
    if $\boldsymbol{L}^{(j)}$ is convolutional **then**
      Flatten spatial dimensions: $Y_i^{(j)} \leftarrow \text{reshape}(Y_i^{(j)}, (b_i h_j w_j) \times d_j)$
    end if
    if $\text{rows}(Y_i^{(j)}) > N_{\max}$ **then**
      Compute representative subset $Y_i'^{(j)} \in \mathbb{R}^{N_{\max} \times d_j}$ via Algorithm 1
    else
      $Y_i'^{(j)} \leftarrow Y_i^{(j)}$
    end if
    Update $U^{(j)}, \Sigma^{(j)}$ via incremental SVD (Equations 9-10)
  end for
end for
for each layer $\boldsymbol{L}^{(j)}$ **do**
  Compute rank k_j using energy threshold r (Equation 1)
  Truncate $U_k^{(j)} \leftarrow U_{:,:k_j}^{(j)}$, $\Sigma_k^{(j)} \leftarrow \Sigma_{1:k_j,1:k_j}^{(j)}$
  Decompose original weights $W^{(j)}$:
    $W_1^{(j)} \leftarrow (U_k^{(j)})^\top W^{(j)}$ ▷ Projection to latent space
  if $\boldsymbol{L}^{(j)}$ is convolutional **then**
    Apply secondary decomposition to $W_1^{(j)}$ (Equation 4)
  end if
  $W_2^{(j)} \leftarrow U_k^{(j)}$ ▷ Reconstruction matrix
  Replace $\boldsymbol{L}^{(j)}$ with $\boldsymbol{L}_1^{(j)} \rightarrow \boldsymbol{L}_2^{(j)}$ cascade
end for

in their respective papers to achieve compression ratio as close to our method as possible. The resulting models are compared based on their accuracy before fine-tuning (Raw Acc.) and their accuracy after fine-tuning. We do not decompose the first and the last layer, as those layers often make up a small fraction of the overall size of the network and contain valuable information that may be lost during compression.

After the compression, we fine-tune the resulting models for a set number of epochs. For CIFAR-10 and CIFAR-100, we fine-tune for 25 epochs with 6 epochs of warm-up (2 epochs are 0.1% of LR, 2 at 1%, and 2 at 10%). After the warm-up phase, the LR is divided by 10 at epochs 18 and 24. For larger (ImageNet) models, we fine-tune for only 20 epochs with three epoch of warm-up, since the fine-tuning of such models is expensive from a time and resource standpoints. We use the same settings for fine-tuning all experiments in order to conduct fair

Table 2. Models used in evaluation on different datasets

Dataset	Model	Parameters	FLOPs	Accuracy (%)
CIFAR-10	ResNet18	703,886	35.78×10^6	91.48%
	ResNet34	1,338,910	74.04×10^6	92.72%
	ResNet50	1,497,519	85.40×10^6	93.39%
	VGG9 BN	5,038,224	107.34×10^6	91.77%
	VGG11 BN	9,761,938	154.58×10^6	91.77%
	VGG13 BN	9,947,220	230.67×10^6	93.55%
	VGG16 BN	15,262,039	315.77×10^6	93.85%
	VGG19 BN	20,576,858	400.85×10^6	93.67%
CIFAR-100	ResNet18	2,818,289	133.25×10^6	70.43%
	ResNet34	5,350,913	285.27×10^6	72.07%
	ResNet50	6,013,329	328.17×10^6	72.47%
ImageNet	ResNet34	21,797,672	3.68×10^9	73.31%

comparisons with the same learning rate, number of epochs, and scheduling. Fine-tuning is done with automatic mixed precision to bfloat16 in order to accelerate training (Micikevicius et al. 2018). We set the base learning rate of fine-tuning to 0.01 for all experiments.

For all experiments, we set the ratio for the second decomposition step to 0.01 and use that to calculate k' for each layer. Although the optimal parameters for fine-tuning may vary between methods, We have not tuned the fine-tuning hyper-parameters for each method, which is a limitation of this experimental evaluation.

4.1 Results on CIFAR-10 and CIFAR-100

By setting the r parameter to a very small value, we can decompose the network while keeping much of its accuracy intact. Table 3 shows the results of applying DDSVD with $r = 0.01$. From the results we can see that our method can compress most of the tested models with a small degradation in performance even without, demonstrating potential for applications where training the resulting network is not possible either due to lack of powerful hardware or the lack of data. For VGG models, which are known to be heavily over-parameterized, the compression ratio reaches upwards of 6.37× while the accuracy reduction is only 2.6%. We also present the recovery of the accuracy of the same models in Figure 2 when decomposed with small values of r, demonstrating the model's ability to keep its accuracy with and without fine-tuning.

For both datasets, we take 128 batches of 128 random images from the training set to sample for DDSVD. We decompose both convolutional and fully connected layers other than the first and last layers of VGG models to increase the compression ratio of DDSVD, since most of the parameters in the VGG architecture are stored in the fully connected layers.

To demonstrate the ability of our scheme to compress CNNs with high compression ratios, we compare DDSVD to other state-of-the-art tensor and matrix decomposition methods (Table 4 and Table 5). We note that DDSVD achieves the best accuracy out of the tested methods in compressing classification models on CIFAR-10. Across the board, our DDSVD achieves accuracy higher than the second best method. For example, on ResNet18, it outperforms TUCKER (Kim et al. 2015) by a margin of 4.49%. We note that while the reduction in FLOPs is not as great as TUCKER, it is greater than both RJSVD an LJSVD.

On CIFAR-10, our method demonstrates superior performance compared to other tested methods, yielding better results both before and after fine-tuning. On CIFAR-100, DDSVD achieves better results than the other

Table 3. Results of decomposing various CIFAR-10 models with $r = 0.01$, showing their original accuracy scores before decomposition, the number of floating point operations (FLOPs) of the decomposed models, their compression ratio (CR), and their accuracy scores before and after fine-tuning. *FT Acc.* represents the accuracy of the network after fine-tuning.

Model	Original Acc.	CR	FLOPs ($\times 10^6$)	Raw Acc.	FT Acc.
ResNet-18	91.48%	1.99×	21.91	84.35%	90.44%
ResNet-34	92.72%	2.18×	43.13	72.52%	91.92%
ResNet-50	93.39%	1.70×	61.65	91.80%	93.18%
VGG9 BN	91.77%	2.39×	49.79	90.01%	90.73%
VGG11 BN	91.77%	3.36×	64.25	90.34%	90.93%
VGG13 BN	93.55%	3.72×	90.81	91.28%	92.73%
VGG16 BN	93.85%	4.84×	107.92	90.46%	92.88%
VGG19 BN	93.67%	6.37×	120.48	91.06%	92.78%

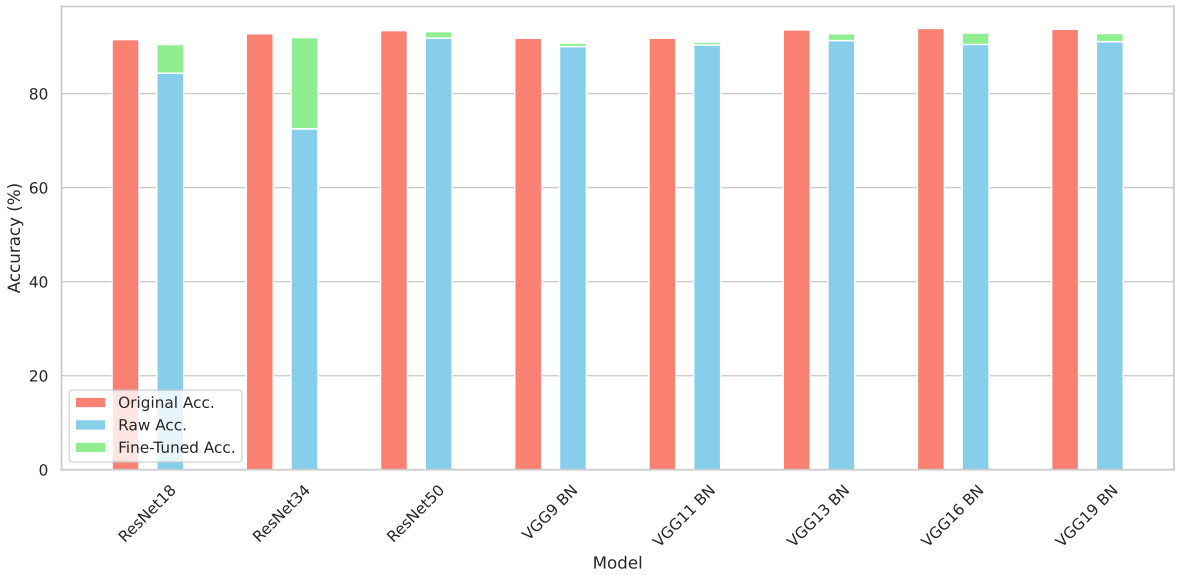

Fig. 2. Visualization of the accuracy of various models with and without fine-tuning. The models were decomposed using DDSVD with $r = 0.01$, resulting in reduced parameters with small reduction in accuracy before fine-tuning.

methods tested on both ResNet18 and ResNet34. However, it is surpassed by RJSVD and LJSVD on ResNet50, which has more layers and therefore benefits from the joint decomposition. On the other hand, our decomposition scheme achieves much lower computational footprint across all experiments in this particular dataset, demonstrating an accuracy to speed trade-off that might be beneficial in certain scenarios.

Figure 3 presents the raw accuracy and the accuracy of the fine-tuned models at different values of r ranging from 0.01 to 0.30. Figure 4 presents the correlation between r and compression ratios for each network.

Table 4. Comparative evaluation of DDSVD with other methods on CIFAR-10.

Model	Method	Raw Acc.	FT Acc.	Parameters (CR)	FLOPs
ResNet18	TUCKER (Kim et al. 2015)	9.94%	75.90%	62,717 (11.16×)	4.21×10^6
	RJSVD (Chen et al. 2023)	10.00%	73.01%	63,546 (11.04×)	5.44×10^6
	LJSVD (Chen et al. 2023)	11.20%	72.64%	63,546 (11.04×)	5.44×10^6
	DDSVD (Ours)	**11.62%**	**80.39%**	62,879 (11.20×)	5.04×10^6
	TUCKER (Kim et al. 2015)	10.00%	74.40%	49,727 (14.10×)	3.53×10^6
	RJSVD (Chen et al. 2023)	10.00%	70.64%	51,498 (13.62×)	4.50×10^6
	LJSVD (Chen et al. 2023)	10.00%	71.42%	51,498 (13.62×)	4.50×10^6
	DDSVD (Ours)	**10.27%**	**75.79%**	49,724 (14.10×)	4.17×10^6
ResNet34	TUCKER (Kim et al. 2015)	9.99%	78.90%	114,289 (11.68×)	7.01×10^6
	RJSVD (Chen et al. 2023)	10.05%	74.60%	115,034 (11.60×)	11.69×10^6
	LJSVD (Chen et al. 2023)	17.87%	75.98%	115,034 (11.60×)	11.69×10^6
	DDSVD (Ours)	**10.51%**	**84.60%**	113,107 (11.81×)	9.53×10^6
	TUCKER (Kim et al. 2015)	10.00%	76.97%	90,483 (14.75×)	5.84×10^6
	RJSVD (Chen et al. 2023)	10.00%	72.76%	90,794 (14.71×)	9.18×10^6
	LJSVD (Chen et al. 2023)	10.00%	72.82%	90,794 (14.71×)	9.18×10^6
	DDSVD (Ours)	**10.12%**	**81.23%**	90,253 (14.79×)	7.89×10^6
ResNet50	TUCKER (Kim et al. 2015)	10.04%	84.30%	215,423 (6.89×)	17.76×10^6
	RJSVD (Chen et al. 2023)	**10.12%**	86.41%	214,666 (6.83×)	21.46×10^6
	LJSVD (Chen et al. 2023)	10.00%	84.82%	215,050 (6.90×)	19.21×10^6
	DDSVD (Ours)	10.10%	**87.48%**	215,468 (6.89×)	18.53×10^6
	TUCKER (Kim et al. 2015)	10.09%	83.79%	196,352 (7.56×)	16.43×10^6
	RJSVD (Chen et al. 2023)	10.42%	85.84%	198,954 (7.46×)	20.51×10^6
	LJSVD (Chen et al. 2023)	**12.07%**	80.45%	196,842 (7.54×)	17.70×10^6
	DDSVD (Ours)	10.00%	**86.07%**	195,975 (7.58×)	16.81×10^6

The experiments show the sensitivity of DDSVD to the parameters r. At $r = 0.01$, the accuracy after decomposition is close to the original. At larger values of r, the accuracy drop is greater (Figure 3), and the compression ratio is larger (Figure 4). ResNet50 shows a lower compression ratio than other networks.

4.2 Results on ImageNet

We evaluate our method on the ILSVRC2012 dataset, which is used as a benchmark for real world vision applications. We use the pre-trained ResNet34 network provided by PyTorch(Ansel et al. 2024) as a starting point and decompose compare the results as earlier experiments. We present our findings in table 6, with the top-1 and top-5 accuracy scores of the models.

We note that DDSVD tends to create models with better accuracy at the same compression ratios than other methods. While TUCKER (Kim et al. 2015) decomposition compresses the models with a lower FLOPs than DDSVD, LJSVD and RJSVD have a much larger computational footprint than both of the other methods, making them not optimal for deployment on low-end devices.

Table 5. Comparative evaluation of DDSVD with other methods on CIFAR-100.

Model	Method	Raw Acc.	FT Acc.	Parameters (CR)	FLOPs
ResNet18	TUCKER (Kim et al. 2015)	0.96%	55.01%	237, 763 (11.83×)	13.16×10^6
	RJSVD (Chen et al. 2023)	1.00%	55.30%	241, 420 (11.66×)	15.93×10^6
	LJSVD (Chen et al. 2023)	1.00%	56.78%	241, 420 (11.66×)	15.93×10^6
	DDSVD (Ours)	**1.56%**	**60.04%**	237, 809 (11.83×)	11.87×10^6
	TUCKER (Kim et al. 2015)	1.00%	53.20%	192, 932 (14.58×)	10.75×10^6
	RJSVD (Chen et al. 2023)	1.00%	50.16%	189, 100 (14.88×)	12.44×10^6
	LJSVD (Chen et al. 2023)	**1.06%**	50.25%	189, 100 (14.88×)	12.44×10^6
	DDSVD (Ours)	0.79%	**56.52%**	192, 860 (14.60×)	9.68×10^6
ResNet34	TUCKER (Kim et al. 2015)	1.00%	60.55%	436, 079 (12.25×)	25.11×10^6
	RJSVD (Chen et al. 2023)	**1.17%**	60.53%	437, 900 (12.20×)	39.05×10^6
	LJSVD (Chen et al. 2023)	1.00%	59.78%	437, 900 (12.20×)	39.05×10^6
	DDSVD (Ours)	1.00%	**64.16%**	436, 505 (12.24×)	22.87×10^6
	TUCKER (Kim et al. 2015)	**1.06%**	55.42%	342, 800 (15.53×)	19.89×10^6
	RJSVD (Chen et al. 2023)	1.00%	56.80%	342, 572 (15.62×)	29.56×10^6
	LJSVD (Chen et al. 2023)	1.00%	56.72%	342, 572 (15.62×)	29.56×10^6
	DDSVD (Ours)	0.80%	**59.92%**	343, 829 (15.53×)	17.69×10^6
ResNet50	TUCKER (Kim et al. 2015)	**2.43%**	62.13%	895, 139 (6.69×)	60.03×10^6
	RJSVD (Chen et al. 2023)	1.50%	**67.85%**	870, 012 (6.88×)	73.27×10^6
	LJSVD (Chen et al. 2023)	1.31%	66.51%	872, 828 (6.86×)	63.86×10^6
	DDSVD (Ours)	0.99%	65.94%	874, 964 (6.84×)	46.74×10^6
	TUCKER (Kim et al. 2015)	**2.32%**	60.50%	781, 857 (7.66×)	51.81×10^6
	RJSVD (Chen et al. 2023)	1.01%	**66.87%**	793, 116 (7.55×)	66.18×10^6
	LJSVD (Chen et al. 2023)	1.28%	66.11%	792, 540 (7.55×)	57.92×10^6
	DDSVD (Ours)	1.00%	64.48%	790, 723 (7.57×)	43.77×10^6

Table 6. Comparative evaluation of DDSVD with other methods on ILSVRC2012. *FT Acc.* is the accuracy of the model after fine-tuning.

Model	Method	Raw Top-1 Acc.	Raw Top-5 Acc.	FT Top-1 Acc.	FT Top-5 Acc.	Parameters (CR)	FLOPs
ResNet34	TUCKER (Kim et al. 2015)	0.17%	0.83%	63.34%	85.33%	2, 793, 101 (7.81×)	417.0×10^6
	RJSVD (Chen et al. 2023)	0.15%	0.70%	62.24%	84.49%	2, 795, 432 (7.80×)	786.9×10^6
	LJSVD (Chen et al. 2023)	0.10%	0.67%	62.61%	84.74%	2, 795, 432 (7.80×)	786.9×10^6
	DDSVD (Ours)	**0.20%**	**1.02%**	**63.61%**	**85.35%**	2, 792, 069 (7.81×)	490.7×10^6
	TUCKER (Kim et al. 2015)	0.14%	0.59%	53.76%	78.32%	1, 609, 954 (13.53×)	206.8×10^6
	RJSVD (Chen et al. 2023)	0.10%	0.51%	53.16%	77.70%	1, 610, 408 (13.53×)	446.1×10^6
	LJSVD (Chen et al. 2023)	0.10%	0.51%	54.11%	78.37%	1, 610, 408 (13.53×)	446.1×10^6
	DDSVD (Ours)	**0.16%**	**0.60%**	**55.00%**	**78.84%**	1, 610, 486 (13.53×)	285.2×10^6

We also compare the average decomposition time between DDSVD and TUCKER model compression. In this test, we use 32 batches of 32 items each. The results in Table **??** show that DDSVD takes significantly more time to process and decompose the model than TUCKER. This observation owes to the complexity of iterative SVD and

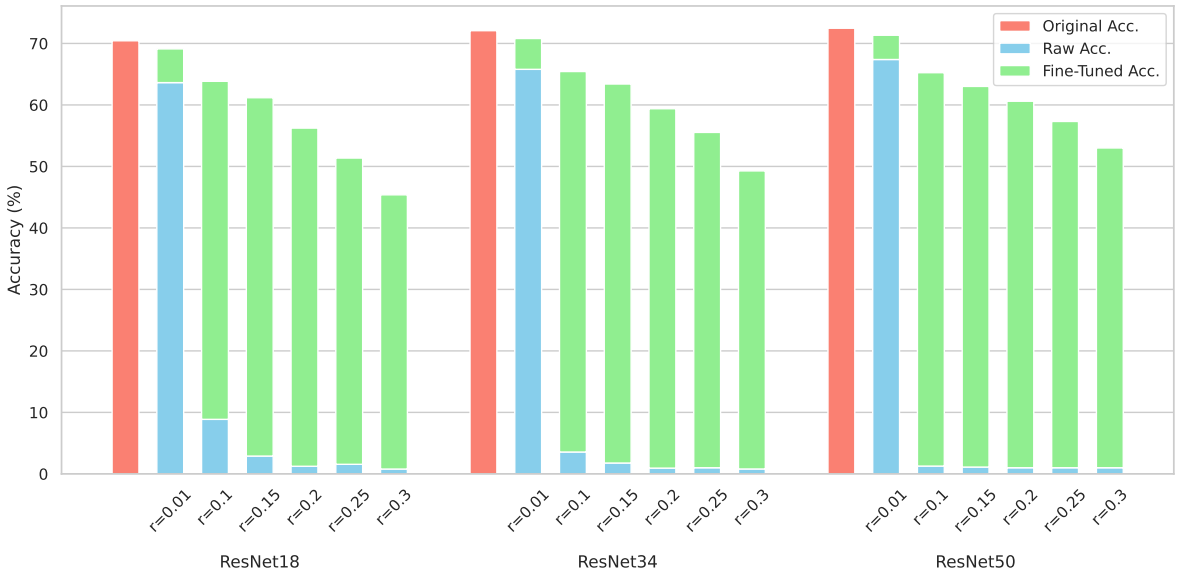

Fig. 3. Raw and fine-tuned accuracy of each model at different values of r (0.01 to 0.30)

Table 7. Average decomposition times (in seconds) for Tucker-based decomposition and the proposed DDA method across different architectures.

Model	Tucker	DDSVD
VGG11-BN	14.4290	23.1359
ResNet34	2.7987	30.8928
ResNet50	2.4383	214.3867
DeiT-Tiny	0.4593	90.3747
DeiT-Small	1.1811	516.9829
DeiT-Base	4.7076	3442.9344
ConvNeXt-Small	1.6103	1271.4475
Swin-Tiny	1.5941	533.7790

representative sampling. We also observe that the decomposition time scales with both the number of layers and with output size, with VGG11 and ResNet34 having the smallest decomposition time, as well as DeiT-Tiny, DeiT-Small, and DeiT-Base having different compression times despite having the same number of layers and structure but different output sizes. While the compression time of DDSVD is much greater than that of Tucker, it is still small compared to the time required for fine-tuning the models.

We compare the layers resulting from DDSVD and the ones resulting from Tucker(Kim et al. 2015) to the original layers when in table 8. We use demonstrate their compression ratio, and similarity to the original using 32 randomly selected images from the training dataset.

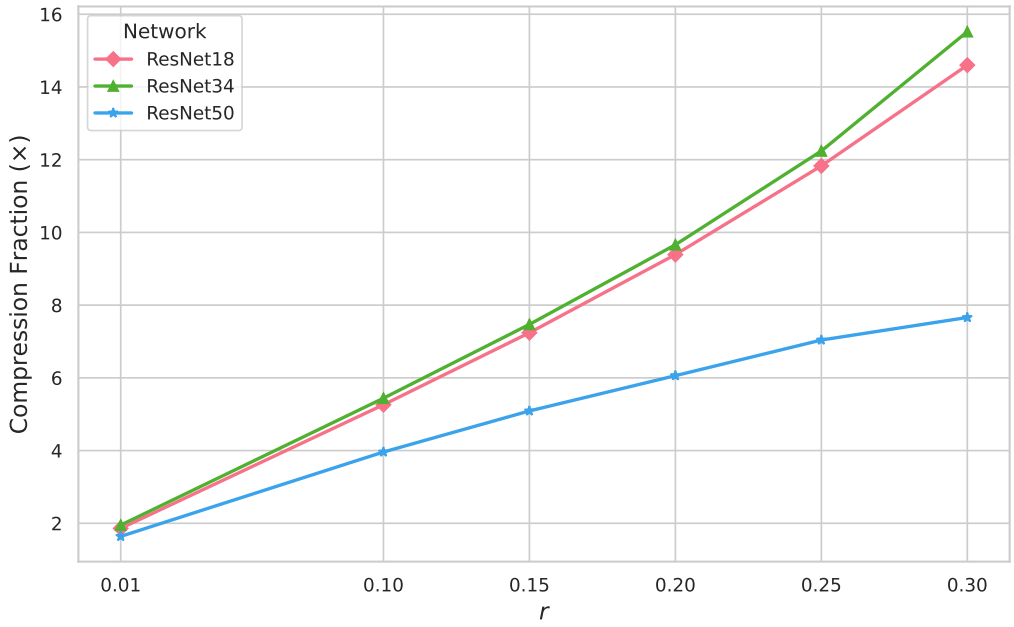

Fig. 4. Compression fractions of different ResNet models at different values of r

4.3 Statistical Tests

To assess the statistical significance of performance variations, we utilize the non-parametric Friedman test (Demšar 2006) to rank accuracy scores across all datasets, models, and compression ratios, providing a robust comparison that accounts for cross-dataset variability without assuming normality.

The results of the Friedman test revealed a test statistic of $X^2 = 16.54$ and a corresponding p-value of 8.774×10^{-4}, which is less than the conventional significance threshold of $\alpha = 0.05$. This p-value strongly suggests that the null hypothesis of no significant difference between the compression methods can be rejected at the $\alpha = 0.05$ significance level. Thus, the test confirms that there are statistically significant differences in performance between the compression methods across the datasets.

To further examine the pairwise differences between methods, we employed the post-hoc Nemenyi test, visualized through a critical difference (CD) diagram (see Figure 5). The CD diagram illustrates the comparisons among the methods, where methods not connected by a horizontal line are significantly different from each other. As depicted in the diagram, our proposed method demonstrates a statistically significant improvement over the other methods tested. In contrast, the remaining methods do not exhibit significant differences in performance when compared to one another.

By utilizing both the Friedman test and the CD diagram, we provide robust statistical evidence that the proposed method offers superior performance in comparison to the alternatives, underlining its effectiveness across different datasets and compression ratios.

Table 8. Properties of compressed layers of ResNet34 compared to the original layers. Propagated similarity (P. Sim.) is the average cosine similarity of the layer's output to the original when the input when data is propagated naturally through the network, while aligned similarity (A. Sim.) is the average cosine similarity when the layer is fed the same input as the original

Layer	Method	#Params.	CR	FLOPs Ratio	P. Sim.	A. Sim.
layer1.0.conv1	Tucker (Kim et al. 2015)	7.9k	4.65×	4.65×	0.924	0.924
	DDSVD	0.8k	46.37×	46.37×	0.932	0.932
layer1.0.conv2	Tucker (Kim et al. 2015)	5.0k	7.38×	7.38×	0.635	0.791
	DDSVD	1.9k	19.01×	19.01×	0.746	0.885
layer1.1.conv1	Tucker (Kim et al. 2015)	3.9k	9.37×	9.37×	0.818	0.833
	DDSVD	0.8k	46.37×	46.37×	0.926	0.928
layer1.1.conv2	Tucker (Kim et al. 2015)	4.1k	8.93×	8.93×	0.442	0.649
	DDSVD	2.9k	12.89×	12.89×	0.71	0.853
layer1.2.conv1	Tucker (Kim et al. 2015)	3.9k	9.37×	9.37×	0.641	0.677
	DDSVD	2.9k	12.89×	12.89×	0.742	0.784
layer1.2.conv2	Tucker (Kim et al. 2015)	3.3k	11.13×	11.13×	0.559	0.702
	DDSVD	5.9k	6.2×	6.2×	0.377	0.871
layer2.0.conv1	Tucker (Kim et al. 2015)	9.2k	8.03×	5.42×	0.519	0.657
	DDSVD	9.8k	7.52×	5.11×	0.721	0.904
layer2.0.conv2	Tucker (Kim et al. 2015)	17.3k	8.51×	8.51×	0.732	0.839
	DDSVD	12.3k	12.02×	12.02×	0.738	0.908
layer2.0.downsample.0	Tucker (Kim et al. 2015)	2.0k	4.06×	2.08×	0.592	0.731
	DDSVD	1.8k	4.53×	3.43×	0.828	0.923
layer2.1.conv1	Tucker (Kim et al. 2015)	13.8k	10.67×	10.67×	0.739	0.836
	DDSVD	9.8k	15.12×	15.12×	0.836	0.908
layer2.1.conv2	Tucker (Kim et al. 2015)	13.1k	11.27×	11.27×	0.704	0.849
	DDSVD	13.4k	11.03×	11.03×	0.758	0.898
layer2.2.conv1	Tucker (Kim et al. 2015)	12.0k	12.26×	12.26×	0.515	0.699
	DDSVD	21.1k	6.99×	6.99×	0.637	0.871
layer2.2.conv2	Tucker (Kim et al. 2015)	8.1k	18.2×	18.2×	0.594	0.739
	DDSVD	24.8k	5.95×	5.95×	0.607	0.89
layer2.3.conv1	Tucker (Kim et al. 2015)	11.6k	12.69×	12.69×	0.648	0.735
	DDSVD	12.3k	12.02×	12.02×	0.8	0.895
layer2.3.conv2	Tucker (Kim et al. 2015)	12.5k	11.78×	11.78×	0.478	0.71
	DDSVD	26.5k	5.56×	5.56×	0.409	0.862

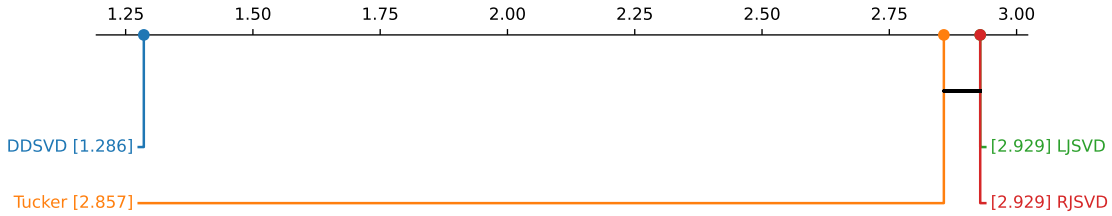

Fig. 5. Critical difference diagram of compression methods

4.4 Impact of Number of Batches

We investigate the correlation between the number of batches used in the approximation of Singular Value Decomposition (SVD) and the quality of the resulting decomposition. For this analysis, we apply Data-Driven SVD (DDSVD) on the three datasets previously employed in our experiments. The batch size is fixed at 64, with the rank parameter set to $r = 0.01$, and the number of batches varying from 1 to 512. To evaluate the effect of batch

Table 9. Properties of compressed layers of ResNet34 compared to the original layers (continued).

Layer	Method	#Params.	CR	FLOPs Ratio	P. Sim.	A. Sim.
layer3.0.conv1	Tucker (Kim et al. 2015)	34.2k	8.63×	6.09×	0.681	0.828
	DDSVD	26.4k	11.16×	7.4×	0.749	0.912
layer3.0.conv2	Tucker (Kim et al. 2015)	59.1k	9.97×	9.97×	0.591	0.796
	DDSVD	72.2k	8.17×	8.17×	0.608	0.907
layer3.0.downsample.0	Tucker (Kim et al. 2015)	3.5k	9.26×	4.69×	0.451	0.583
	DDSVD	9.6k	3.4×	2.6×	0.544	0.805
layer3.1.conv1	Tucker (Kim et al. 2015)	61.1k	9.65×	9.65×	0.785	0.885
	DDSVD	37.4k	15.76×	15.76×	0.798	0.911
layer3.1.conv2	Tucker (Kim et al. 2015)	38.0k	15.52×	15.52×	0.679	0.826
	DDSVD	66.2k	8.9×	8.9×	0.63	0.896
layer3.2.conv1	Tucker (Kim et al. 2015)	44.4k	13.3×	13.3×	0.802	0.879
	DDSVD	30.2k	19.55×	19.55×	0.784	0.906
layer3.2.conv2	Tucker (Kim et al. 2015)	36.4k	16.22×	16.22×	0.624	0.79
	DDSVD	77.3k	7.63×	7.63×	0.606	0.885
layer3.3.conv1	Tucker (Kim et al. 2015)	38.9k	15.15×	15.15×	0.787	0.871
	DDSVD	26.6k	22.15×	22.15×	0.776	0.905
layer3.3.conv2	Tucker (Kim et al. 2015)	45.0k	13.1×	13.1×	0.607	0.816
	DDSVD	50.0k	11.8×	11.8×	0.582	0.851
layer3.4.conv1	Tucker (Kim et al. 2015)	37.7k	15.63×	15.63×	0.776	0.868
	DDSVD	29.0k	20.35×	20.35×	0.773	0.898
layer3.4.conv2	Tucker (Kim et al. 2015)	41.9k	14.09×	14.09×	0.594	0.81
	DDSVD	63.5k	9.28×	9.28×	0.565	0.876
layer3.5.conv1	Tucker (Kim et al. 2015)	37.2k	15.84×	15.84×	0.718	0.854
	DDSVD	47.4k	12.44×	12.44×	0.697	0.905
layer3.5.conv2	Tucker (Kim et al. 2015)	42.5k	13.89×	13.89×	0.598	0.823
	DDSVD	53.8k	10.95×	10.95×	0.61	0.864
layer4.0.conv1	Tucker (Kim et al. 2015)	120.9k	9.76×	6.91×	0.521	0.846
	DDSVD	106.5k	11.08×	7.36×	0.45	0.874
layer4.0.conv2	Tucker (Kim et al. 2015)	242.9k	9.71×	9.71×	0.387	0.771
	DDSVD	396.4k	5.95×	5.95×	0.345	0.882
layer4.0.downsample.0	Tucker (Kim et al. 2015)	10.2k	12.91×	6.51×	0.327	0.587
	DDSVD	58.6k	2.24×	1.72×	0.365	0.871
layer4.1.conv1	Tucker (Kim et al. 2015)	220.2k	10.71×	10.71×	0.848	0.943
	DDSVD	53.1k	44.45×	44.45×	0.846	0.913
layer4.1.conv2	Tucker (Kim et al. 2015)	234.4k	10.06×	10.06×	0.439	0.803
	DDSVD	404.4k	5.83×	5.83×	0.327	0.899
layer4.2.conv1	Tucker (Kim et al. 2015)	245.1k	9.63×	9.63×	0.896	0.958
	DDSVD	53.1k	44.45×	44.45×	0.906	0.942
layer4.2.conv2	Tucker (Kim et al. 2015)	538.6k	4.38×	4.38×	0.168	0.897
	DDSVD	444.1k	5.31×	5.31×	0.155	0.904

variation, we measure both the raw accuracy and the compression ratio (CR) for each combination of dataset and model. The results are visualized in Figures 6, 7, and 8, where accuracy and CR are plotted as functions of the number of batches.

The results reveal a clear and consistent correlation between the number of batches and both accuracy and compression ratio. Specifically, we observe that as the number of batches increases, the compression ratio decreases, indicating a more compact representation of the model. Simultaneously, model accuracy improves with the increasing number of batches. This behavior suggests that larger batch sizes lead to better approximations, enhancing model performance at the cost of reduced compression efficiency.

For the specific case of the CIFAR-10 dataset using the VGG13 BN model, the optimal balance between accuracy and compression ratio appears to occur when the number of batches is between 32 and 64. In this range, the

model achieves near-maximum accuracy while maintaining a favorable compression ratio. Beyond 64 batches, the accuracy begins to plateau, while the compression ratio continues to decrease, reflecting diminishing returns in accuracy improvement relative to the cost in parameter overhead.

Table 10 presents the detailed performance metrics for VGG13 BN on CIFAR-10 as the number of batches increases. As shown, with 1 batch, the model achieves a top-1 accuracy of 89.58% with a compression ratio (CR) of 5.03×. As the number of batches increases to 32, accuracy improves to 91.45% with a reduced CR of 3.90×. Notably, beyond 64 batches, further improvements in accuracy become marginal, with the maximum top-1 accuracy of 91.61% observed at 256 batches, while the compression ratio drops to 3.77×. At 512 batches, however, a slight degradation in accuracy is observed, dropping to 91.00%, suggesting that excessively large batch sizes may introduce noise or instability in the decomposition process.

These findings underscore the trade-off between model size and accuracy, where increasing the number of batches enhances the quality of the decomposition at the expense of compression efficiency. The observed optimal range of 32 to 64 batches provides a practical guideline for balancing these two competing factors in the context of DDSVD.

Table 10. Accuracy and compression ratio for VGG13 BN on CIFAR-10 with varying numbers of batches.

Batches	Number of Parameters	CR	Accuracy
1	1,978,049	5.03×	89.58%
2	2,250,152	4.42×	90.45%
4	2,400,827	4.14×	90.84%
8	2,485,707	4.00×	91.15%
16	2,525,302	3.94×	91.26%
32	2,551,507	3.90×	91.45%
64	2,562,251	3.88×	91.57%
128	2,575,729	3.86×	91.51%
256	2,639,963	3.77×	91.61%
512	2,737,243	3.63×	91.00%

4.5 Impact of the Batch Size

In a further investigation of the impact of batch size on model performance, we fix the number of batches at 4 while varying the batch size from 1 to 512. This analysis focuses on the ResNet34 model trained on the CIFAR-100 dataset. Similar to the previous analysis, we assess the correlation between CR and top-1 accuracy, seeking to understand the trade-offs between these two metrics as batch size increases. Figures 9, 10, and 11 shows the complete results of this experiment.

Taking ResNet34 on CIFAR-100 as an example, table 11 presents the results of this experiment. Initially, with a batch size of 1, the model exhibits a very low top-1 accuracy of 2.65%, along with a relatively high compression factor of 5.75×. This result indicates that with a small batch size, the model's performance is severely constrained, likely due to the limitations in the ability of the SVD approximation to capture the key structures of the data. As the batch size increases to 8, we observe a significant boost in performance, with the top-1 accuracy rising to 58.32% and the compression factor reducing to 2.65×. This dramatic improvement suggests that larger batch sizes allow for a more precise decomposition, which in turn improves the model's ability to generalize.

Beyond a batch size of 16, the gains in accuracy begin to plateau, with the maximum top-1 accuracy of 64.13% observed at a batch size of 64. However, further increases in batch size beyond 64 yield diminishing returns. For

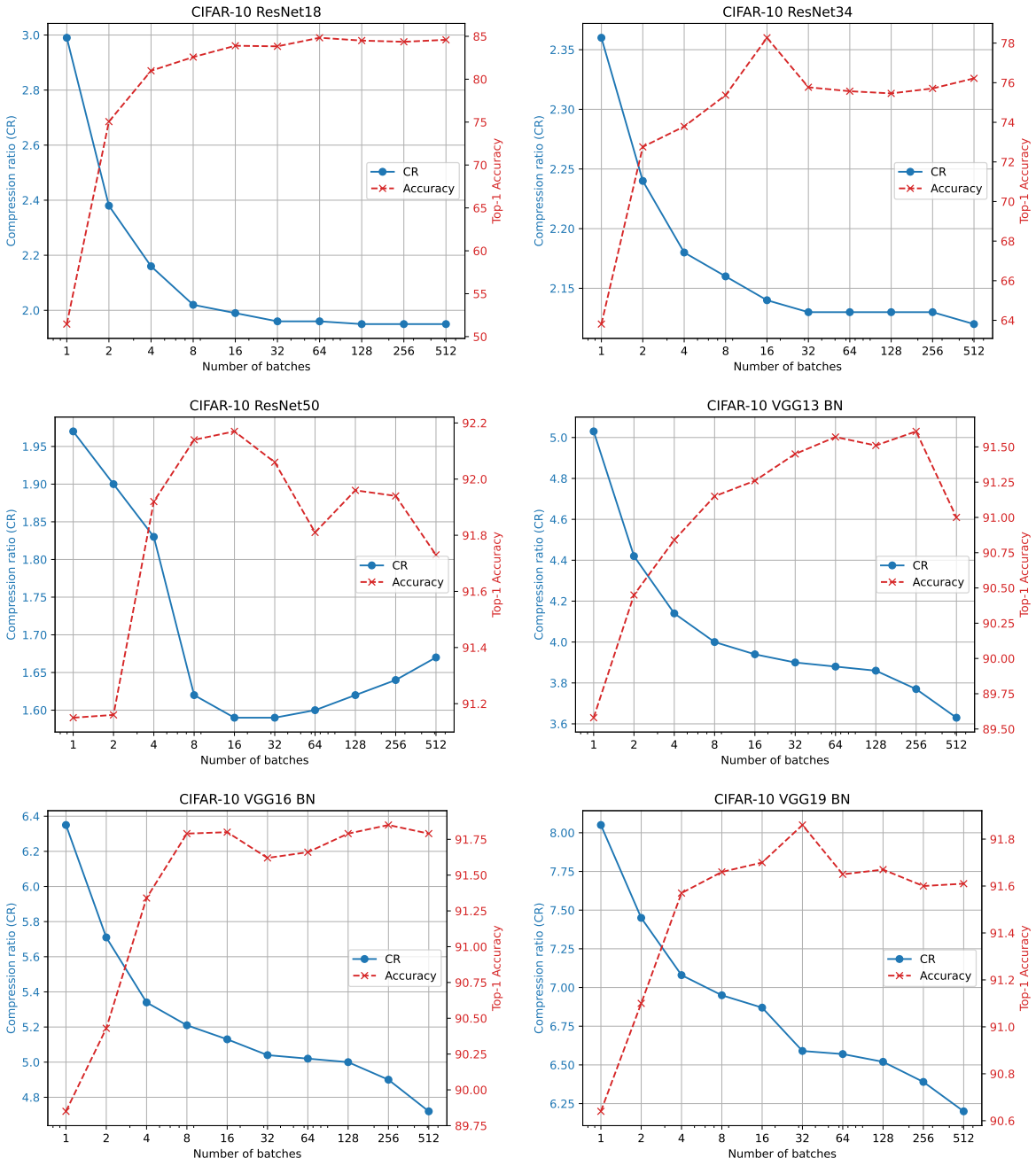

Fig. 6. Impact of the number of batches on the accuracy and compression ratio of the models on CIFAR-10

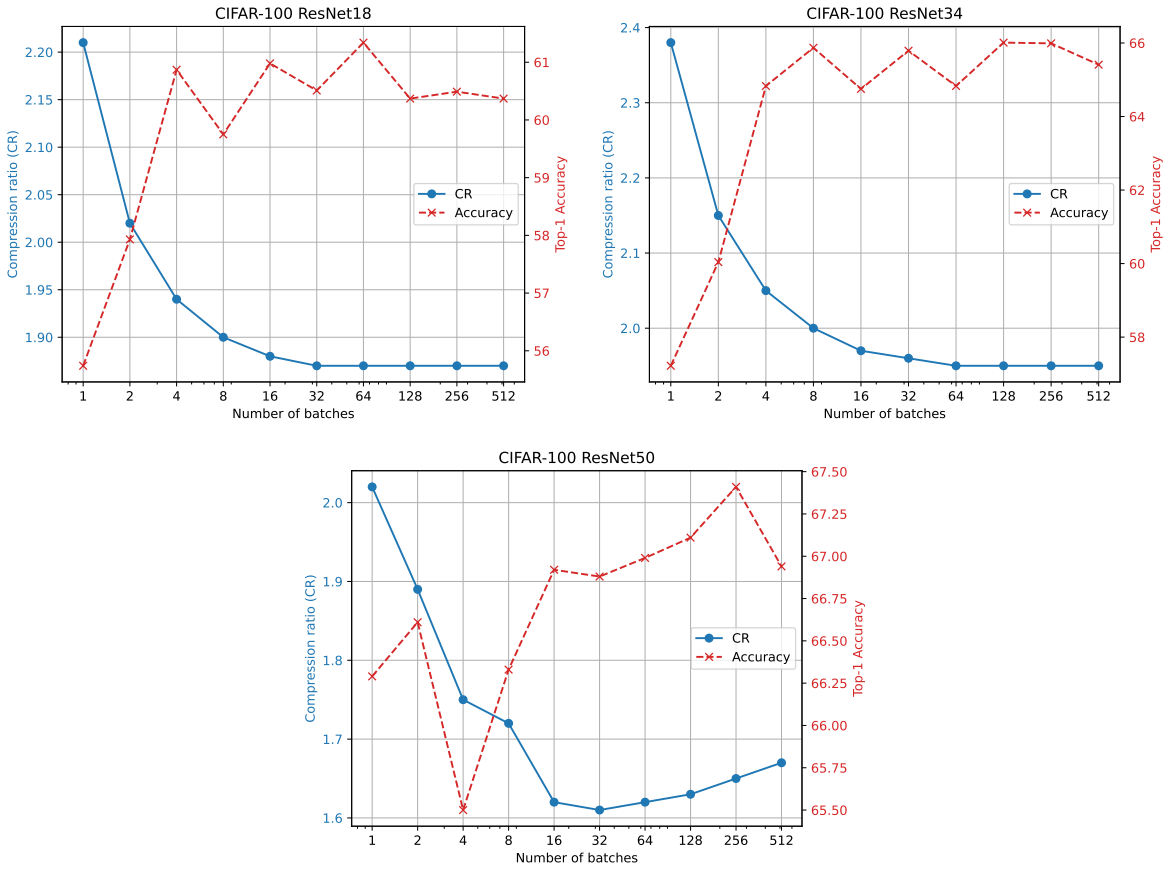

Fig. 7. Impact of the number of batches on the accuracy and compression ratio of the models on CIFAR-100

instance, at a batch size of 128, the accuracy drops slightly to 63.63%, and at 512, it further declines to 63.26%. Despite these marginal decreases, the compression factor remains stable, hovering around 2.01× to 2.05×. This stability indicates that the model's size remains consistent, suggesting that increasing batch size beyond a certain point has little effect on model compression, while accuracy can start to degrade.

These findings indicate that, unlike the number of batches, the batch size presents a more nuanced relationship with model performance. For smaller batch sizes (e.g., 1–16), increases lead to substantial performance gains, but these gains taper off at larger batch sizes. Therefore, batch sizes between 16 and 64 seem to offer an optimal balance between compression and accuracy, beyond which there is little to no improvement in performance. Moreover, the results suggest that extremely large batch sizes can negatively impact accuracy, possibly due to over-fitting or noise introduced during the incremental SVD approximation.

In conclusion, while increasing batch size enhances the quality of the decomposition and model performance up to a certain point, excessively large batch sizes yield diminishing returns. These insights highlight the importance of selecting an appropriate batch size for optimal model compression and accuracy, reinforcing that tuning batch-related parameters is crucial in practical applications of DDSVD.

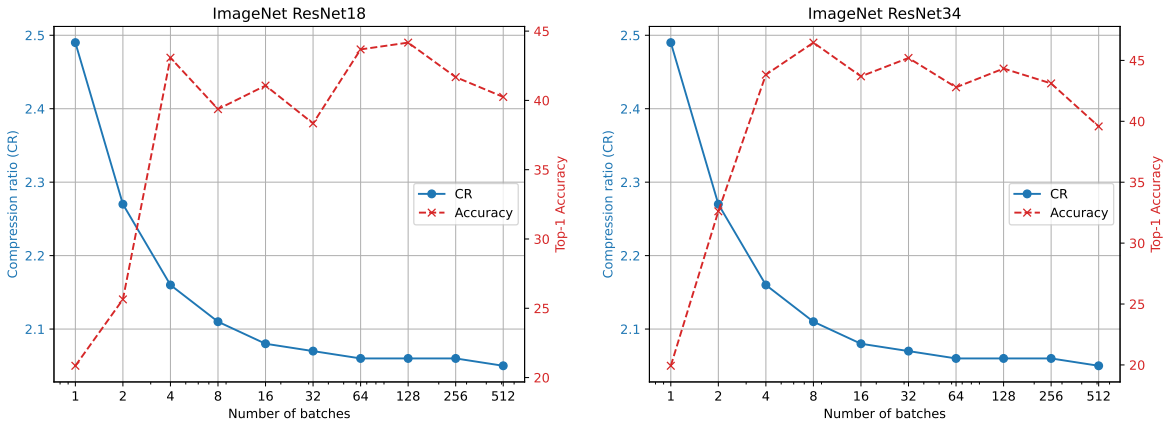

Fig. 8. Impact of the number of batches on the accuracy and compression ratio of the models on ImageNet-1k

Table 11. Accuracy and compression ratio for ResNet34 on CIFAR-100 with a fixed number of batches (4) and varying batch sizes.

Batch Size	Number of Parameters	CR	Accuracy
1	928,957	5.75×	2.65%
2	1,266,918	4.22×	12.19%
4	1,652,026	3.23×	39.94%
8	2,018,624	2.65×	58.32%
16	2,325,682	2.30×	63.99%
32	2,505,919	2.13×	62.95%
64	2,608,974	2.05×	64.13%
128	2,648,382	2.02×	63.63%
256	2,657,438	2.01×	63.45%
512	2,652,922	2.01×	63.26%

4.6 Discussion

The experimental results obtained in this study demonstrate the superior performance of the Data-Driven SVD (DDSVD) method in achieving more optimal decompositions than alternative approaches, thereby validating our initial hypothesis that the intrinsic patterns within the data significantly influence the quality of the decomposition. This observation highlights the capacity of DDSVD to adapt to the underlying structure of the data, facilitating more effective decomposition. Moreover, DDSVD proves to be an effective tool for compressing neural networks with minimal need for additional fine-tuning, a property that can be particularly advantageous in scenarios where computational resources are constrained or rapid deployment is necessary.

Our experiments have shown that our method can be applied quickly to the neural networks. We have observed that smaller models can be compressed in as little as a few seconds, and larger models on larger datasets requiring no longer than a few minutes. As highlighted in Table 8 and continued in Table 9, DDSVD consistently produces layers that are up to 10 times more compact compared to those obtained via TUCKER decomposition (Kim et al.

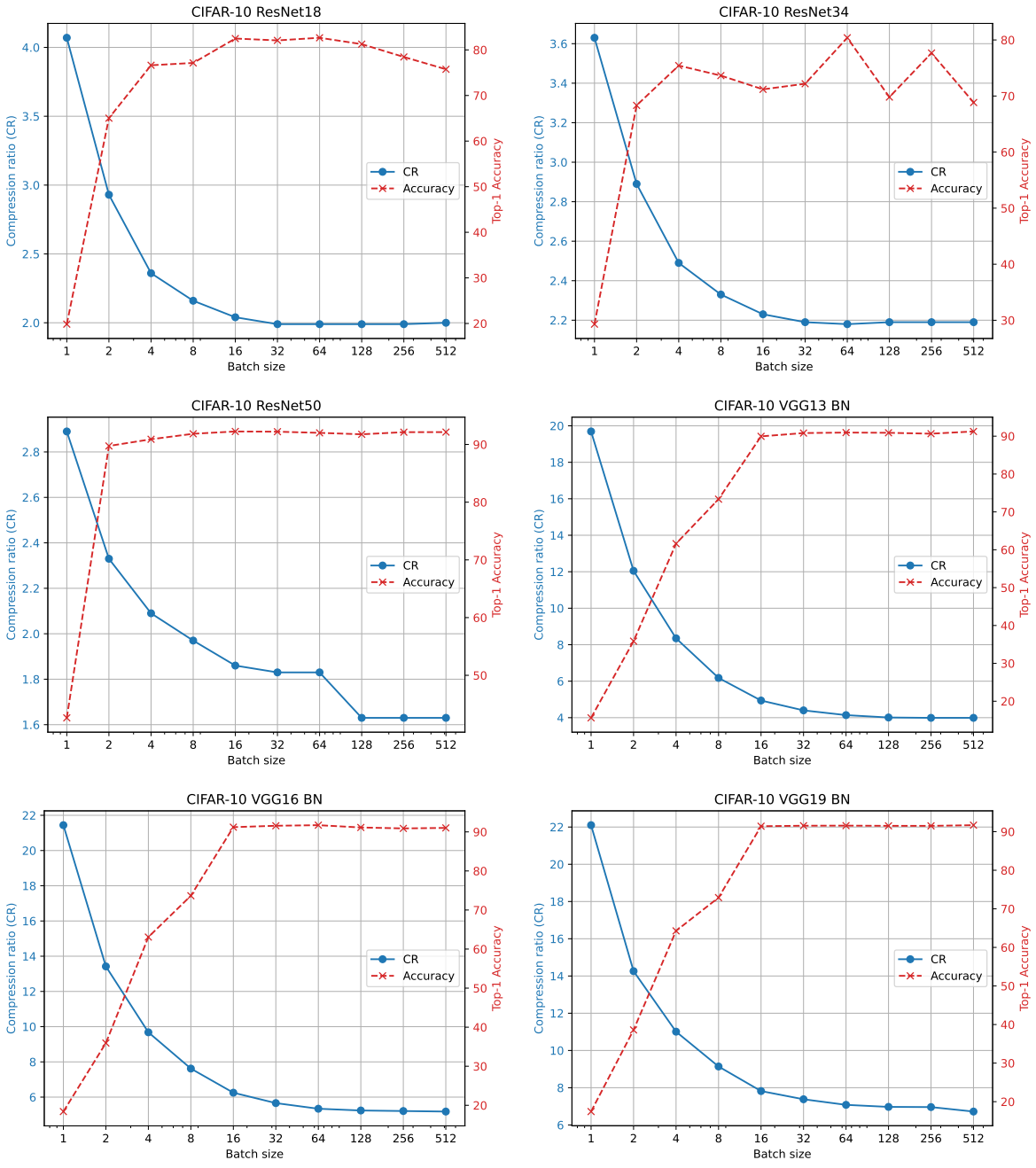

Fig. 9. Impact of batch size on the accuracy and compression ratio of the models on CIFAR-10

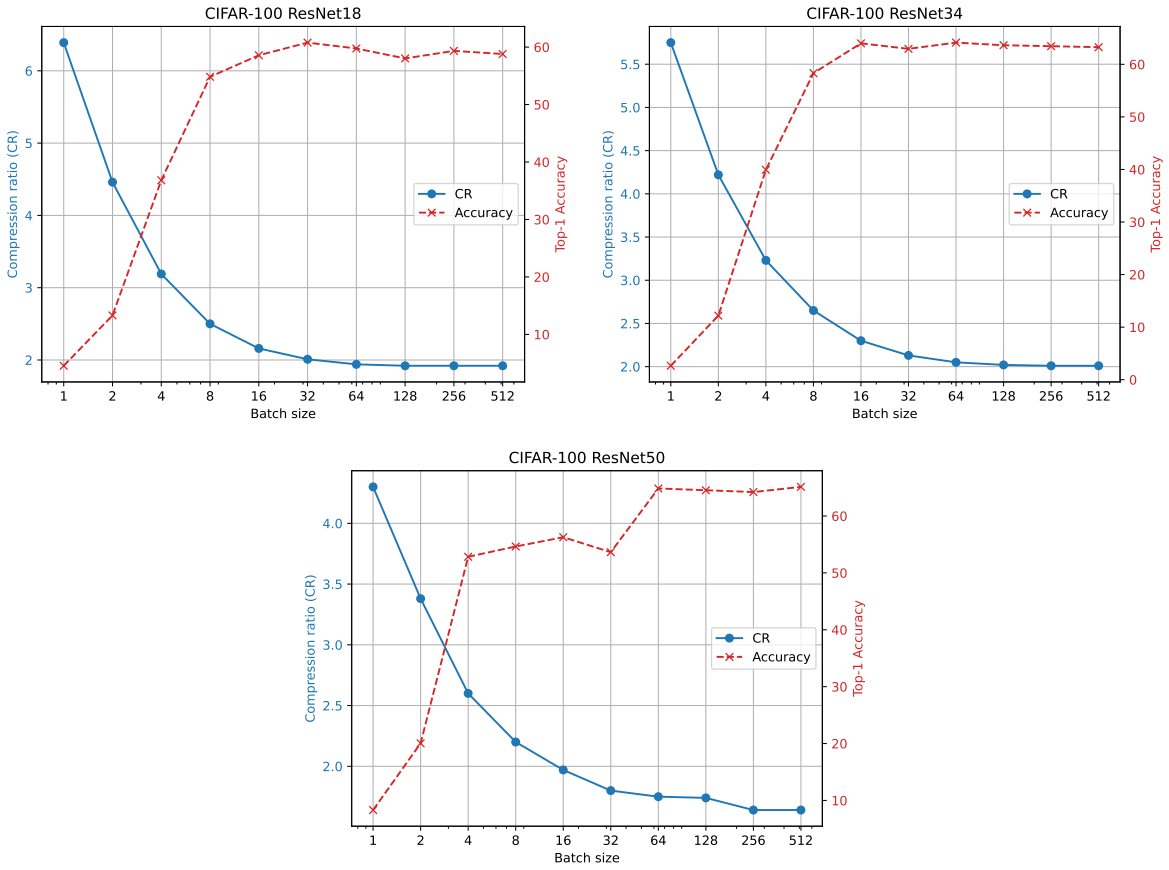

Fig. 10. Impact of batch size on the accuracy and compression ratio of the models on CIFAR-100

2015), while maintaining superior similarity scores. This is exemplified by the results for ***layer1.0.conv1***, where DDSVD achieves significantly higher compression without sacrificing representational accuracy. Such findings confirm that DDSVD not only compresses neural network layers more effectively, but also preserves critical structural properties of the layers, leading to decompositions that closely approximate the original layers rather than merely reducing their parameter count. This characteristic is essential for maintaining the functional integrity of the network post-decomposition, ensuring that compression does not degrade model performance.

When compared with other state-of-the-art methods, DDSVD consistently outperforms in terms of accuracy while maintaining a comparable number of parameters. Although the reduction in floating point operations (FLOPs) achieved by DDSVD is on par with TUCKER decomposition (Kim et al. 2015), DDSVD substantially outperforms RJSVD and LJSVD (Chen et al. 2023) in terms of both accuracy and model efficiency. The ability of DDSVD to compress models without compromising on accuracy positions it as a favorable method for practical neural network compression tasks.

However, it should be noted that DDSVD does not surpass RJSVD in certain cases, particularly in scenarios where joint decompositions are advantageous, such as with networks containing a large number of identical layers, as seen in architectures like ResNet50 on CIFAR-100. In such cases, RJSVD benefits from its joint decomposition

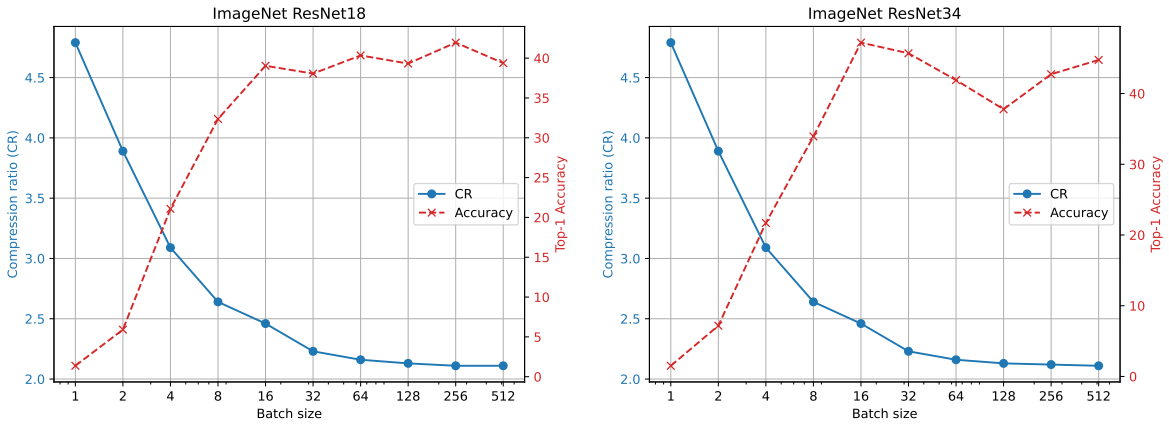

Fig. 11. Impact of the batch size on the accuracy and compression ratio of the models on ImageNet-1k

strategy, which exploits redundancies across multiple layers to achieve higher compression rates. This limitation suggests that while DDSVD excels in layer-specific decompositions, its efficacy in networks with substantial layer redundancy may be constrained compared to methods that leverage joint decomposition techniques.

In summary, DDSVD offers a promising and efficient approach for neural network compression, particularly in cases where minimal fine-tuning is required. Its ability to preserve the structural integrity of layers while achieving substantial compression underscores its potential for deployment in resource-constrained environments. Nonetheless, further exploration of hybrid approaches that integrate the strengths of both DDSVD and joint decomposition methods could yield even more efficient compression techniques in the future.

5 Conclusion and Future Works

In this study, we introduced a novel method for data-driven Singular Value Decomposition (DDSVD) aimed at deep neural network compression. Our approach, motivated by the observation that patterns in the weights do not necessarily reflect patterns in the data, operates on the outputs of the layers instead of their weights. Our method has demonstrated significant capacity to compress both convolutional and fully connected layers with great efficacy. Our experimental evaluations, conducted on well-known architectures such as VGG and ResNet, highlight the efficacy of our approach. DDSVD is able to extract an optimal sub-network, with up to 6.3× compression ratio and 3.3× reduction in FLOPs with small drop in accuracy with little to no fine-tuning in the case of VGG19. In some cases, such as ResNet34, the accuracy drop is steeper, but the accuracy can be recovered with fine-tuning.

These findings suggest that DDSVD is a powerful tool for neural network optimization, offering a pathway to deploy high-performance models on resource-constrained devices. In future works, we would like to extend this work with global importance estimation such as with the genetic algorithm. We would also like to extend this work with a framework adapted to the compression of transformer-based language models and other recent architectures.

References

J. Ansel et al.. Apr. 2024. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation." In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, (Apr. 2024). doi:10.1145/3620665.3640366.

S. Anwar, K. Hwang, and W. Sung. Feb. 2017. "Structured Pruning of Deep Convolutional Neural Networks." *J. Emerg. Technol. Comput. Syst.*, 13, 3, Article 32, (Feb. 2017), 18 pages. doi:10.1145/3005348.

N. Ardalani, J. Hestness, and G. Diamos. 2019. *Empirically Characterizing Overparameterization Impact on Convergence.* (2019). https://openreview.net/forum?id=S1lPShAqFm.

M. Astrid and S.-I. Lee. Feb. 2017. "CP-decomposition with Tensor Power Method for Convolutional Neural Networks compression." In: *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, (Feb. 2017). doi:10.1109/bigcomp.2017.7881725.

M. Astrid and S.-I. Lee. Aug. 2018. "Deep compression of convolutional neural networks with low-rank approximation." *ETRI Journal*, 40, 4, (Aug. 2018), 421–434. doi:10.4218/etrij.2018-0065.

R. Banner, I. Hubara, E. Hoffer, and D. Soudry. 2018. "Scalable methods for 8-bit training of neural networks." In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (NIPS'18). Curran Associates Inc., Montréal, Canada, 5151–5159.

J. A. Bullinaria and J. P. Levy. Sept. 2012. "Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and SVD." en. *Behav. Res. Methods*, 44, 3, (Sept. 2012), 890–907.

S. Chen, J. Zhou, W. Sun, and L. Huang. Jan. 2023. "Joint matrix decomposition for deep convolutional neural networks compression." *Neurocomputing*, 516, (Jan. 2023), 11–26. doi:10.1016/j.neucom.2022.10.021.

S. Dalton, L. Olson, and N. Bell. Oct. 2015. "Optimizing Sparse Matrix—Matrix Multiplication for the GPU." *ACM Trans. Math. Softw.*, 41, 4, Article 25, (Oct. 2015), 20 pages. doi:10.1145/2699470.

J. Demšar. 2006. "Statistical Comparisons of Classifiers over Multiple Data Sets." *Journal of Machine Learning Research*, 7, 1, 1–30. http://jmlr.org/papers/v7/demsar06a.html.

B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Apr. 2020. "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey." *Proceedings of the IEEE*, 108, 4, (Apr. 2020), 485–532. doi:10.1109/jproc.2020.2976475.

M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. 2013. "Predicting parameters in deep learning." In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (NIPS'13). Curran Associates Inc., Lake Tahoe, Nevada, 2148–2156.

E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. 2014. "Exploiting linear structure within convolutional networks for efficient evaluation." In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (NIPS'14). MIT Press, Montreal, Canada, 1269–1277.

S. Du and J. Lee. Oct. 2018. "On the Power of Over-parametrization in Neural Networks with Quadratic Activation." In: *Proceedings of the 35th International Conference on Machine Learning* (Proceedings of Machine Learning Research). Ed. by J. Dy and A. Krause. Vol. 80. PMLR, (Oct. 2018), 1329–1338. https://proceedings.mlr.press/v80/du18a.html.

K. Fukushima and S. Kamata. Mar. 2025. "Stochastic Quantization and Diffusion Models." *Journal of the Physical Society of Japan*, 94, 3, (Mar. 2025). doi:10.7566/jpsj.94.031010.

M. Gabor and R. Zdunek. 2023. "Compressing convolutional neural networks with hierarchical Tucker-2 decomposition." *Applied Soft Computing*, 132, 109856.

G. H. Golub and C. F. Van Loan. 1996. *Matrix computations (3rd ed.)* Johns Hopkins University Press, USA. ISBN: 0801854148.

X. Guan, C.-T. Li, and Y. Guan. 2017. "Matrix Factorization With Rating Completion: An Enhanced SVD Model for Collaborative Filtering Recommender Systems." *IEEE Access*, 5, 27668–27678. doi:10.1109/access.2017.2772226.

K. He, X. Zhang, S. Ren, and J. Sun. 2015. "Deep Residual Learning for Image Recognition." *CoRR*, abs/1512.03385. http://arxiv.org/abs/1512.03385 arXiv: 1512.03385.

Y. He, X. Zhang, and J. Sun. Oct. 2017. "Channel Pruning for Accelerating Very Deep Neural Networks." In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, (Oct. 2017). doi:10.1109/iccv.2017.155.

B. Heo, J. Kim, S. Yun, H. Park, N. Kwak, and J. Y. Choi. Oct. 2019. "A Comprehensive Overhaul of Feature Distillation." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. (Oct. 2019).

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. 2018. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations." *Journal of Machine Learning Research*, 18, 187, 1–30. http://jmlr.org/papers/v18/16-456.html.

A. Ismael Kadhim, Y.-N. Cheah, I. Abbas Hieder, and R. Ahmed Ali. 2017. "Improving TF-IDF with singular value decomposition (SVD) for feature extraction on twitter." In: *IEC2017 Proceedings Book*. Ishik University.

"An Improved Active Machine Learning Query Strategy for Entity Matching Problem." *Advances in Machine Intelligence and Computer Science Applications*. Springer Nature Switzerland, 317–327. ISBN: 9783031293139. doi:10.1007/978-3-031-29313-9_28.

M. Jabrane, H. Tabbaa, A. Hadri, and I. Hafidi. Nov. 2024. "Enhancing Entity Resolution with a hybrid Active Machine Learning framework: Strategies for optimal learning in sparse datasets." *Information Systems*, 125, (Nov. 2024), 102410. doi:10.1016/j.is.2024.102410.

M. Jaderberg, A. Vedaldi, and A. Zisserman. 2014. "Speeding up Convolutional Neural Networks with Low Rank Expansions." In: *Proceedings of the British Machine Vision Conference 2014* (BMVC 2014). British Machine Vision Association, 88.1–88.13. doi:10.5244/c.28.88.

"Partitioning Around Medoids (Program PAM)." *Finding Groups in Data*. John Wiley & Sons, Ltd. Chap. 2, 68–125. ISBN: 9780470316801. doi:https://doi.org/10.1002/9780470316801.ch2.

Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. 2015. "Compression of deep convolutional neural networks for fast and low power mobile applications." *arXiv preprint arXiv:1511.06530*.

"Batch Size Influence on Performance of Graphic and Tensor Processing Units During Training and Inference Phases." *Advances in Intelligent Systems and Computing*. Springer International Publishing, (Mar. 2019), 658–668. ISBN: 9783030166212. doi:10.1007/978-3-030-16621-2_61.

A. Krizhevsky. 2009. "Learning Multiple Layers of Features from Tiny Images." In: https://api.semanticscholar.org/CorpusID:18268744.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. May 2017. "ImageNet classification with deep convolutional neural networks." *Communications of the ACM*, 60, 6, (May 2017), 84–90. doi:10.1145/3065386.

J. Lang, Z. Guo, and S. Huang. Dec. 2024. "A Comprehensive Study on Quantization Techniques for Large Language Models." In: *2024 4th International Conference on Artificial Intelligence, Robotics, and Communication (ICAIRC)*. IEEE, (Dec. 2024), 224–231. doi:10.1109/icairc64177.2024.10899941.

Y. LeCun, J. Denker, and S. Solla. 1989. "Optimal brain damage." *Advances in neural information processing systems*, 2.

G. Li, R. Li, T. Li, C. Shen, X. Zou, J. Wang, C. Wang, and N. Li. Jan. 2025. "SFP: Similarity-based filter pruning for deep neural networks." *Information Sciences*, 689, (Jan. 2025), 121418. doi:10.1016/j.ins.2024.121418.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. 2016. *Pruning Filters for Efficient ConvNets*. (2016). doi:10.48550/ARXIV.1608.08710.

B. Liu, F. Li, X. Wang, B. Zhang, and J. Yan. June 2023. "Ternary Weight Networks." In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, (June 2023). doi:10.1109/icassp49357.2023.10094626.

J. Liu, S. Tripathi, U. Kurup, and M. Shah. 2020. "Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications: A Survey." *CoRR*, abs/2005.04275. https://arxiv.org/abs/2005.04275 arXiv: 2005.04275.

Y. Liu and M. K. Ng. Apr. 2022. "Deep neural network compression by Tucker decomposition with nonlinear response." *Knowledge-Based Systems*, 241, (Apr. 2022), 108171. doi:10.1016/j.knosys.2022.108171.

Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Oct. 2017. "Learning Efficient Convolutional Networks Through Network Slimming." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. (Oct. 2017).

Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. 2019. *Rethinking the Value of Network Pruning*. (2019). arXiv: 1810.05270 (cs.LG).

T. maintainers and contributors. 2016. *TorchVision: PyTorch's Computer Vision library*. (2016). https://github.com/pytorch/vision.

C. H. Martin and M. W. Mahoney. 2021. "Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning." *Journal of Machine Learning Research*, 22, 165, 1–73. http://jmlr.org/papers/v22/20-410.html.

G. Menghani. Mar. 2023. "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better." *ACM Comput. Surv.*, 55, 12, Article 259, (Mar. 2023), 37 pages. doi:10.1145/3578938.

P. Micikevicius et al.. 2018. "Mixed Precision Training." In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=r1gs9JgRZ.

P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. 2017. "Pruning Convolutional Neural Networks for Resource Efficient Inference." In: *International Conference on Learning Representations*. https://openreview.net/forum?id=SJGCiw5gl.

J. Mourad, T. Hiba, R. Yassir, and H. Imad. Mar. 2024. "ERABQS: entity resolution based on active machine learning and balancing query strategy." *Journal of Intelligent Information Systems*, (Mar. 2024). doi:10.1007/s10844-024-00853-0.

Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson. Oct. 2021. "Loss aware post-training quantization." *Machine Learning*, 110, 11–12, (Oct. 2021), 3245–3262. doi:10.1007/s10994-021-06053-z.

H.-S. Park and C.-H. Jun. Mar. 2009. "A simple and fast algorithm for K-medoids clustering." en. *Expert Syst. Appl.*, 36, 2, (Mar. 2009), 3336–3341.

L. Pastur. 2022. "Eigenvalue distribution of large random matrices arising in deep neural networks: Orthogonal case." *Journal of Mathematical Physics*, 63, 6. doi:10.1063/5.0085204.

J. Ran, R. Lin, H. K. H. So, G. Chesi, and N. Wong. 2021. "Exploiting Elasticity in Tensor Ranks for Compressing Neural Networks." *CoRR*, abs/2105.04218. https://arxiv.org/abs/2105.04218 arXiv: 2105.04218.

O. Russakovsky et al.. 2015. "ImageNet Large Scale Visual Recognition Challenge." *International Journal of Computer Vision (IJCV)*, 115, 3, 211–252. doi:10.1007/s11263-015-0816-y.

I. H. Sarker. Aug. 2021. "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions." *SN Computer Science*, 2, 6, (Aug. 2021). doi:10.1007/s42979-021-00815-1.

S. Schotthöfer, E. Zangrando, J. Kusch, G. Ceruti, and F. Tudisco. 2022. "Low-rank lottery tickets: finding efficient low-rank neural networks via matrix differential equations." In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 20051–20063. https://proceedings.neurips.cc/paper_files/paper/2022/file/7e98b00eeafcdaeb0c5661fb9355be3a-Paper-Conference.pdf.

K. Simonyan and A. Zisserman. 2015. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. (2015). arXiv: 1409.1556 (cs.CV).

G. W. Stewart. 1993. "On the Early History of the Singular Value Decomposition." *SIAM Rev.*, 35, 551–566. https://api.semanticscholar.org/CorpusID:15315509.

M. Sun, Z. Liu, A. Bair, and Z. Kolter. 2024. "A Simple and Effective Pruning Approach for Large Language Models." In: *International Conference on Learning Representations*. Ed. by B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun. Vol. 2024, 4942–4964. https://proceedings.iclr.cc/paper_files/paper/2024/file/14c856c7a41297804de4c4890e846b25-Paper-Conference.pdf.

S. Swaminathan, D. Garg, R. Kannan, and F. Andres. July 2020. "Sparse low rank factorization for deep neural network compression." *Neurocomputing*, 398, (July 2020), 185–196. doi:10.1016/j.neucom.2020.02.035.

N. Tian, Y. Liu, W. Wang, and D. Meng. July 2021. "Fast CNN Inference by Adaptive Sparse Matrix Decomposition." In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, (July 2021). doi:10.1109/ijcnn52387.2021.9533546.

L. Wang, X. Hu, B. Yuan, and J. Lu. 2015. "Active learning via query synthesis and nearest neighbour search." *Neurocomputing*, 147, 426–434. Advances in Self-Organizing Maps Subtitle of the special issue: Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012). doi:https://doi.org/10.1016/j.neucom.2014.06.042.

G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. 23–29 Jul 2023. "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models." In: *Proceedings of the 40th International Conference on Machine Learning* (Proceedings of Machine Learning Research). Ed. by A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett. Vol. 202. PMLR, (23–29 Jul 2023), 38087–38099. https://proceedings.mlr.press/v202/xiao23c.html.

C. Yang, Z. Yang, A. M. Khattak, L. Yang, W. Zhang, W. Gao, and M. Wang. 2019. "Structured Pruning of Convolutional Neural Networks via L1 Regularization." *IEEE Access*, 7, 106385–106394. doi:10.1109/access.2019.2933032.

Z. Yuan, Y. Shang, Y. Song, D. Yang, Q. Wu, Y. Yan, and G. Sun. 2025. *ASVD: Activation-aware Singular Value Decomposition for Compressing Large Language Models*. (2025). https://arxiv.org/abs/2312.05821 arXiv: 2312.05821 [cs.CL].

"Visualizing and Understanding Convolutional Networks." *Computer Vision – ECCV 2014*. Springer International Publishing, 818–833. ISBN: 9783319105901. doi:10.1007/978-3-319-10590-1_53.

H. Zha and H. D. Simon. Jan. 1999. "On updating problems in latent semantic indexing." *SIAM J. Sci. Comput.*, 21, 2, (Jan. 1999), 782–791.

S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. Oct. 2016. "Cambricon-X: An accelerator for sparse neural networks." In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, (Oct. 2016). doi:10.1109/micro.2016.7783723.

X. Zhao, A. Farjudian, and A. Bellotti. Jan. 2025. "Pruning convolutional neural networks for inductive conformal prediction." *Neurocomputing*, 611, (Jan. 2025), 128704. doi:10.1016/j.neucom.2024.128704.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. June 2016. "Learning Deep Features for Discriminative Localization." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (June 2016).

X. Zhou, J. He, G. Huang, and Y. Zhang. June 2015. "SVD-based incremental approaches for recommender systems." *Journal of Computer and System Sciences*, 81, 4, (June 2015), 717–733. doi:10.1016/j.jcss.2014.11.016.

Received 24 March 2025; accepted 14 May 2026