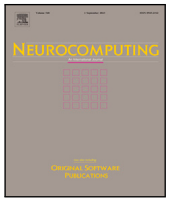
Building efficient CNNs using Depthwise Convolutional Eigen-Filters (DeCEF)

Neurocomputing 609 (2024) 128461

Contents lists available at ScienceDirect

Neurocomputing

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



Building efficient CNNs using Depthwise Convolutional Eigen-Filters (DeCEF)

Yinan Yu [a,c,d,*], Samuel Scheidegger [c,d], Tomas McKelvey [b]

[a] Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden
[b] Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden
[c] Asymptotic AI, Gothenburg, Sweden
[d] Lumilogic, Gothenburg, Sweden

ARTICLE INFO

Communicated by A. Mukherjee

Keywords:
Convolutional neural network
Low rank approximation
Subspace method
Network complexity
Efficient network
Deep learning

ABSTRACT

Deep Convolutional Neural Networks (CNNs) have been widely used in various domains due to their impressive capabilities. These models are typically composed of a large number of 2D convolutional (Conv2D) layers with numerous trainable parameters. To manage the complexity of such networks, compression techniques can be applied, which typically rely on the analysis of trained deep learning models. However, in certain situations, training a new CNN from scratch may be infeasible due to resource limitations. In this paper, we propose an alternative parameterization to Conv2D filters with significantly fewer parameters without relying on compressing a pre-trained CNN. Our analysis reveals that the effective rank of the vectorized Conv2D filters decreases with respect to the increasing depth in the network. This leads to the development of the Depthwise Convolutional Eigen-Filter (DeCEF) layer, which is a low rank version of the Conv2D layer with significantly fewer trainable parameters and floating point operations (FLOPs). The way we define the effective rank is different from previous work, and it is easy to implement and interpret. Applying this technique is straightforward – one can simply replace any standard convolutional layer with a DeCEF layer in a CNN. To evaluate the effectiveness of DeCEF layers, experiments are conducted on the benchmark datasets CIFAR-10 and ImageNet for various network architectures. The results have shown a similar or higher accuracy using about 2/3 of the original parameters and reducing the number of FLOPs to 2/3 of the base network. Additionally, analyzing the patterns in the effective rank provides insights into the inner workings of CNNs and highlights opportunities for future research.

1. Introduction

Deep CNN is one of the most commonly used data-driven techniques. Typically, the large number of trainable parameters in deep learning models result in high demands on the computational power and memory capacities, which requires renting or purchasing expensive infrastructure for training. The high power consumption during training and inference is not environmentally friendly [1]. Moreover, the size of the network and the number of FLOPs play an important role for the inference process, where a small edge device may be used with restrictions on the complexity of the runtime. Therefore, building an efficient network is beneficial in terms of saving computational resources and reducing the overall cost for deep learning while achieving similar performances.

One topic on constructing an efficient CNN is the Neural Architecture Search (NAS), where the focus is to search for an optimal architecture given certain criteria. In this paper, however, we assume that the wiring of the layers is pre-determined. Our focus is on how to improve the efficiency of a CNN for a given architecture.

The literature primarily highlights two strategies to achieve this. The first strategy is to take a trained network and reduce the most insignificant parameters. This refers to as *compression* or *pruning* in the literature. This is often a reasonable approach since many applications are using pre-trained networks as backbones in their networks.

However, this strategy depends on the availability of a reusable pre-trained network. Since the significance of network weights is often data-dependent, factors such as variations in data, restrictive licensing,

* Corresponding author at: Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden.
E-mail address: yinan@chalmers.se (Y. Yu).

[1] Although not being the main focus of this work, the proposed method can also be applied as a compression technique derived from a pre-trained network. This aspect is elaborated in Appendix B.

https://doi.org/10.1016/j.neucom.2024.128461
Received 25 October 2022; Received in revised form 11 April 2023; Accepted 21 August 2024
Available online 3 September 2024

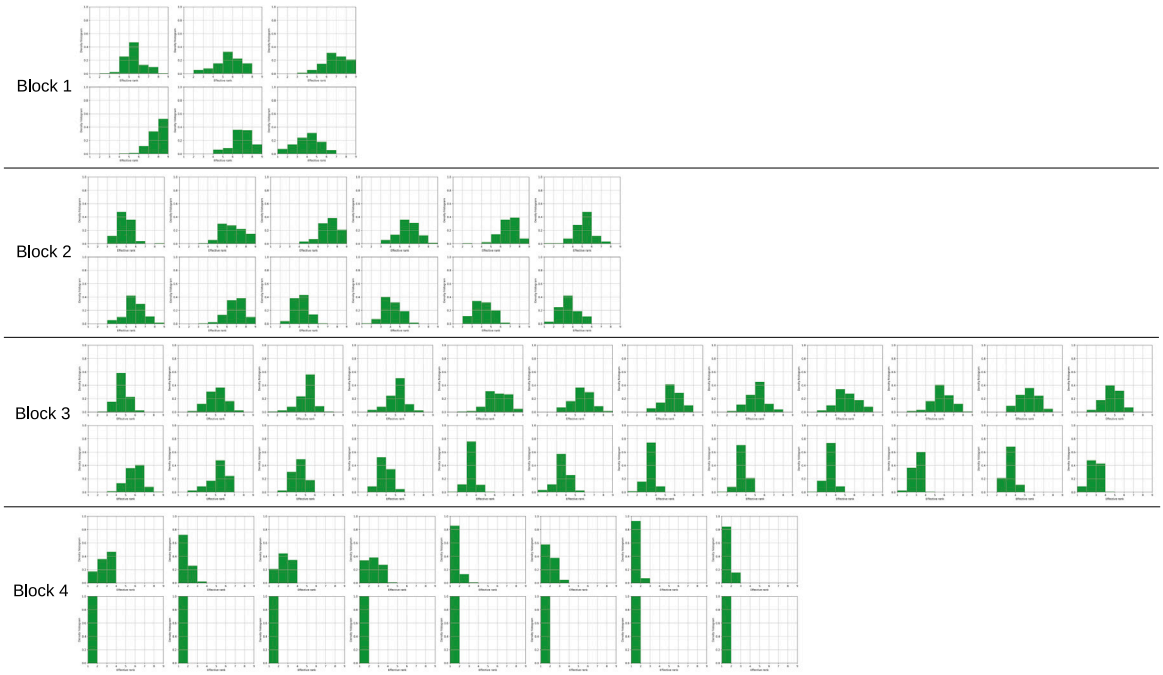


Fig. 1. The density histogram (y-axis$\in [0,1]$) of the effective rank (x-axis$\in [1,9]$) estimated using Eq. (2) in Procedure 1 for DenseNet-121 trained on ImageNet. The statistics are computed over all input channels i in that layer. We see the decreasing trend of the effective ranks with respect to the depth of the network.

or other constraints can make it impractical to reuse a pre-trained network. This makes training a neural network from scratch unavoidable. Nonetheless, training the original network from scratch may not be feasible due to the potentially high resource requirements. In this case, after the overall architecture is established, one may re-parameterize the CNN to make it more efficient before training. That is, the network is still aimed to accomplish what the original CNN is supposed to achieve but with significantly fewer trainable parameters and FLOPs. The approximation is on the functional level instead of relying on trained parameters. This is the main focus of this work.[1]

The main hypothesis for finding an efficient re-parameterization strategy is that there is significant redundancy in Conv2D layers, which means that it may be sufficient to express a Conv2D layer with fewer parameters in order to achieve similar performances. One of the most commonly used function approximation techniques is the subspace low rank representation [2–4]. It is a family of very well studied and widely used techniques in the area of signal processing and machine learning. To put it in the context of CNN, the main idea is to rearrange the trainable variables into a vector space and find a subspace spanned by the most significant singular vectors of these variables. This new representation typically results in fewer trainable variables and inference FLOPs during runtime with potentially better robustness.

There are two key steps involved to achieve this approximation: (1) find a **representative vector space** for each layer, and (2) estimate the **effective rank** without training. To find a representative vector space for Conv2D filters in a CNN, we have designed experiments where we observe that 1) vectorized Conv2D filters exhibit low rank behaviors, and (2) the effective ranks are different for each layer and they have a decreasing tendency with respect to the depth of the network. Given these observations, we propose a new convolutional filter DeCEF. DeCEF is parameterized by a new hyperparameter we call rank, where a full rank DeCEF is equivalent to a Conv2D filter, whereas a rank one DeCEF is equivalent to a depthwise separable convolutional layer. To avoid the common problem of over-tuning, we use a rule-based approach for finding the ranks, where the rules are predetermined by cross-validation on a small dataset trained on a small network. The rules are then applied to larger datasets and networks without further adjustments or tuning.

The paper is organized as follows. First, to motivate our work, we present the experiments and methodologies being used to observe and analyze the low rank behaviors in several trained CNNs in Section 2.1. We then propose the definition of a new type of filter parameterization DeCEF in Section 2.2. To further illustrate the advantages of using a DeCEF layer, we show two key properties, robustness and complexity, in Section 2.3. In Section 2.4, we present the training strategies for DeCEF. In Section 4, we show experiments to evaluate the effectiveness of DeCEF. First, we run ablation studies on the smaller dataset CIFAR-10 using DeCEF to gain empirical insights of its behaviors in Section 4.2. To further evaluate the two properties of DeCEF, we conduct experiments using the benchmark network ResNet-50 on ImageNet for comparing complexity versus accuracy. Moreover, in Section 4.3, we run further experiments on two additional popular network architectures DenseNet and HRNet. These results are also compared to other state-of-the-art model reduction techniques in Section 4.3.

2. DeCEF layers

2.1. Motivation

First, let us formally define what a layer is in this context.

Definition 1. In the scope of this paper, a Conv2D layer (or a **layer** for short)

$$\mathcal{W} = \left\{ \mathbf{w}_j^{(i)} \in \mathbb{R}^{h\times h} \,:\, i = 1\cdots c_{\text{in}}, j = 1\cdots c_{\text{out}} \right\}$$

is a set of trainable units that are characterized by the following attributes: (1) number of input channels c_{in}; (2) number of output channels c_{out}, and (3) parameterization: $\mathbf{w}_j^{(i)} \in \mathbb{R}^{h\times h}$, i.e. the Conv2D filter.

Note that there are multiple layers in a network, but we ignore the layer index in this definition for simplicity. When multiple layers appear in the same context, we use $\mathcal{W}_l$ to denote the indexed layer, where the subscript $l \in \{1,\dots,L\}$ is the layer index and L is the **depth**[2]

[2] To clarify, this depth refers to the depth of the network. The *depthwise* in DeCEF refers to the depth (i.e. input channels) of a layer, which is a different concept.

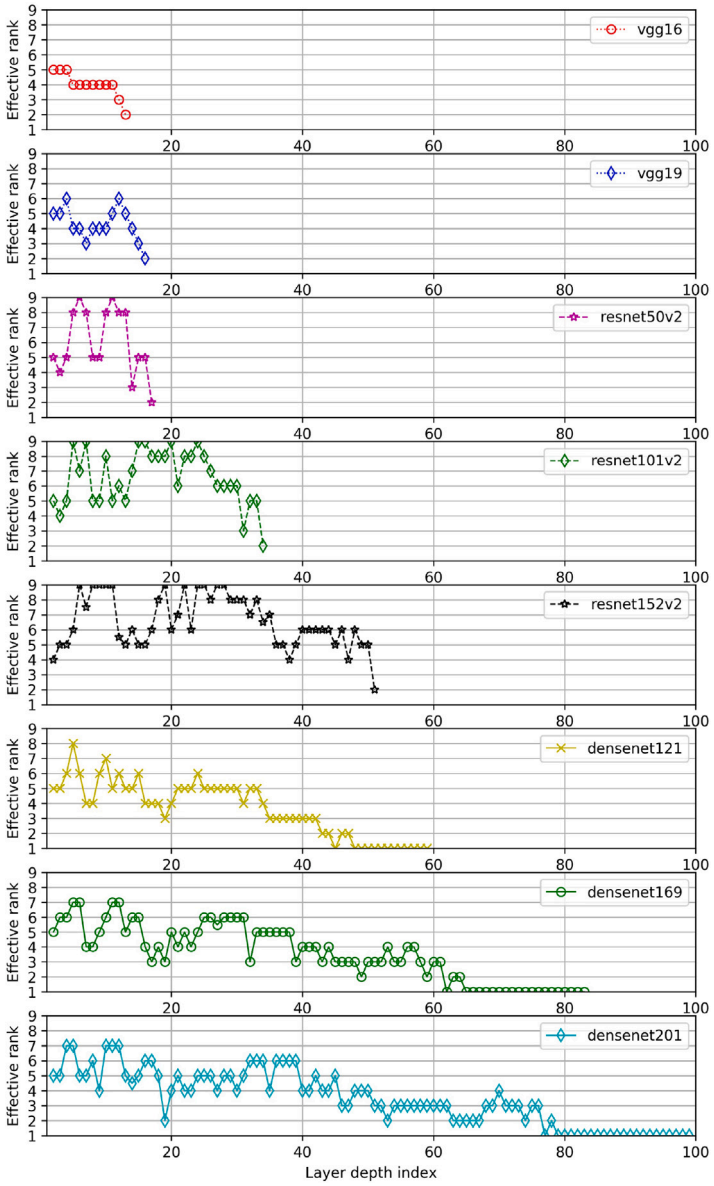


Fig. 2. Effective rank (cf. Eq. (3)) versus layer depth. In these networks, we observe decreasing trend of the effective ranks when a network goes deeper. In this figure, we show this effect in the networks VGG, ResNet and DenseNet.

of the network. In addition, we denote $K := h^2$. Note that in practice, the filter shape may be rectangular. Moreover, for the sake of both consistency and convenience, we use i and j to denote the input channel index and the output channel index, respectively.

Our motivation of this work has originated from the low rank behaviors we have observed in the vectorized filter parameters, so let us start with this experimental procedure to illustrate our findings.

Procedure 1. *Observing low rank behaviors*

- *Apply vectorization* $\bar{\mathbf{w}}_j^{(i)} := \mathrm{vec}(\mathbf{w}_j^{(i)}) \in \mathbb{R}^K$ *and compute the truncated Singular Value Decomposition (SVD):*

$$\bar{\mathbf{U}}^{(i)}\mathbf{S}^{(i)}\mathbf{V}^{(i)T} = \begin{bmatrix} \bar{\mathbf{w}}_1^{(i)} & \cdots & \bar{\mathbf{w}}_{c_{out}}^{(i)} \end{bmatrix}, \tag{1}$$

  where matrices $\bar{\mathbf{U}}^{(i)}$ *and* $\mathbf{V}^{(i)}$ *are the left and right singular matrix, respectively; and* $\mathbf{S}^{(i)}$ *is a diagonal matrix that contains the singular values in a descending order. The implementation of this procedure is well supported by any linear algebra libraries in most programming languages.*
- *Identify the effective rank for each input channel* i*:*

$$r_i =| \{\mathbf{S}^{(i)}[k,k] : \mathbf{S}^{(i)}[k,k] \geq \gamma \mathbf{S}^{(i)}[1,1], \quad k = 1, \ldots, \min(K, c_{out}), \ \gamma \in [0,1]\} | \tag{2}$$

  where $|\cdot|$ *denotes the cardinality of a set and* $\mathbf{S}[k,k]$ *is the* k*th diagonal element of matrix* $\mathbf{S}$*.*
- *The effective rank of one layer* l*:*

$$r^l =| \{s^l : s^l \geq \gamma, k = 1, \ldots, \min(K, c_{out}), \ \gamma \in [0,1]\} | \tag{3}$$

  where $s^l = \mathbb{E}_i \left(\frac{\mathbf{S}^{(i)}[k,k]}{\mathbf{S}^{(i)}[1,1]} \right)$ *and the expected value can be estimated by averaging over all input channels* i*.*

To illustrate the empirical values, examples can be found in Figs. 1 and 2. Fig. 1 shows the density histogram of singular values computed using Eq. (1). The histogram is calculated from all input channels in each convolutional layer with $K > 1$. The maximum ranks of the layers in these example networks are $\min(K, c_{\text{out}}) = K$, where $K = 9$. Similar low rank behaviors can be observed in Fig. 2.

To summarize what we have observed:

(1) the vectorized Conv2D filters in a trained CNN exhibit low rank properties (cf. Fig. 1);
(2) the effective ranks of vectorized filters show a decreasing tendency when the network goes deeper (cf. Fig. 2);

(3) the effective ranks of vectorized filters converge over training steps (see video in supplementary material).

Given these observations, we propose a new layer called DeCEF as an alternative parameterization to Conv2D layers for the purpose of reducing the redundancy.

2.2. Definition

In this section, we introduce the definition of DeCEF followed by its two properties. Generally speaking, subspace techniques bring better robustness to the learning system due to their reduced model complexity. Motivated by these observations and analyses, we define a DeCEF layer as follows:

Definition 2 (*DeCEF Layer*)**.** A DeCEF layer is defined by

$$\Theta = \left\{ \mathbf{w}_j^{(i)}, \mathbf{w}_j^{(i)} \in \mathbb{R}^{h\times h}, i = 1 \cdots c_{\text{in}}, j = 1 \cdots c_{\text{out}} \right\}$$

with the following parameterization

$$\mathbf{w}_j^{(i)} = \sum_{k=1}^{r} a_{k,j}^{(i)} \mathbf{u}_k^{(i)}, \ r \in [1, h^2] \tag{4}$$

where $a_{k,j}^{(i)} \in \mathbb{R}$ and $\mathbf{u}_k^{(i)} \in \mathbb{R}^{h\times h}$, which satisfies

$$\bar{\mathbf{u}}_l^{(i)T} \bar{\mathbf{u}}_m^{(i)} = \begin{cases} 1 & \text{if } l = m \\ 0 & \text{otherwise} \end{cases}$$

for $\bar{\mathbf{u}}_k^{(i)} = \text{vec}(\mathbf{u}_k^{(i)}) \in \mathbb{R}^{h^2}$. The parameters $\mathbf{u}_k^{(i)} \in \mathbb{R}^{h\times h}$ are called the **eigen-filters**.

Note that for the sake of clarity, we use Θ to denote the DeCEF layer, instead of the generic notation $\mathcal{W}$ in Definition 1.

2.3. Properties

In this section, we present two key properties of the DeCEF layer. These properties are then empirically evaluated in the experiment section.

Property 1. *Complexity (one layer)*

- *Number of trainable parameters (N)*
  - *N(Conv2D):* $c_{in}c_{out}h^2$
  - *N(DeCEF): :* $N_u + N_a$*, where*
    - *Eigen-filters:* $N_u = c_{in}h^2 r$
    - *Coefficients:* $N_a = c_{in}c_{out}r$

  For $r = h^2$*, it is trivial to randomly initialize eigen-filters that span the whole* h^2 *dimensional vector space and hence the eigen-filters do not need to be trainable, i.e.* $N_u = 0$ *and* $N_a = c_{in}c_{out}h^2$*. Therefore, Conv2D and DeCEF are equivalent for* $r = h^2$*.*
  For $r < h^2$*,* $N(\text{DeCEF}) < N(\text{Conv2D})$ *if* $r \le \left\lfloor \frac{c_{out}h^2}{c_{out}+h^2} \right\rfloor$*.*

  Example. Given $c_{in} = c_{out} = 128$ *and* $h = 3$*, we have* $N(\text{Conv2D}) = 147456$*. If* $r \le 8 < h^2 = 9$*, then* $N(\text{DeCEF}) < N(\text{Conv2D})$*. For* $r = 8$*,* $N(\text{DeCEF}) = 140288$ *and for* $r = 4$*,* $N(\text{DeCEF}) = 70144$*.*
- *FLOPs (F)*
  We count the multiply-accumulate operations (macc) and we do not include bias in our calculations. Given the dimension of the input layer $H \times W \times c_{in}$*, let* $t = \left\lfloor \frac{H}{stride} \right\rfloor \times \left\lfloor \frac{W}{stride} \right\rfloor$*,*
  - *F(Conv2D):* $th^2c_{in}c_{out}$
  - *F(DeCEF):* $tc_{in}r\left(h^2 + c_{out}\right)$

  Example.Given $H = W = 100$*,* $c_{in} = 128$*,* $c_{out} = 128$ *and* $h = 3$ *with* $stride = 1$*, we have* $F(\text{Conv2D}) = 1.47$ *GFLOPs. For* $r = 8$*,* $F(\text{DeCEF}) = 1.40$ *GFLOPs. For* $r = 4$*,* $F(\text{DeCEF}) = 0.70$ *GFLOPs.*

Property 2. *Robustness*

Lemma 1. *Let* $\Delta\mathbf{I}_i$ *be an additive perturbation matrix and* $\mathbf{w}_j^{(i)} \in \mathbb{R}^{h\times h}$ *be a filter parameterized by Eq. (4), which is learned from some training process. Let*

$$\bar{\mathbf{U}}^{(i)} = \left[\bar{\mathbf{u}}_0^{(i)}, \ldots, \bar{\mathbf{u}}_r^{(i)} \right]. \tag{5}$$

If $\bar{\mathbf{U}}^{(i)T}\bar{\mathbf{U}}^{(i)} = \mathbf{I}$ *and* $\left\|\mathbf{a}_j^{(i)}\right\|_2 \le \epsilon$*,* $\forall i, j$*,*

$$\left\| \sum_i \Delta\mathbf{I}_i * \mathbf{w}_j^{(i)} \right\|_\infty \le \epsilon hr \sum_i \left\|\Delta\mathbf{I}_i\right\|_2. \tag{6}$$

Proof. See Appendix A. □

Robustness in this context is indicated by the propagation of the additive perturbation between input and output feature maps. Lemma 1 shows that when (1) $\bar{\mathbf{U}}^{(i)\text{T}}\bar{\mathbf{U}}^{(i)} = \mathbf{I}$, i.e. the vectorized filters are orthonormal, and (2) $\left\|\mathbf{a}_j^{(i)}\right\|_2 \le \epsilon$, i.e. the coefficients are bounded by ϵ, the effect of the perturbation on the output is bounded by Eq. (6).

The rank r of the eigen-filters is a hyperparameter that yields a trade-off between the robustness and the representational power of a DeCEF layer. In this work, we use a rule based approach for choosing this hyperparameter.

2.4. Training algorithms

In this section, we show how to construct and train a network composed of DeCEF layers.

2.4.1. The optimization problem

Given a network architecture with a set of layers $\mathcal{N} = \{\mathcal{W}_1 \ \cdots \ \mathcal{W}_L\}$. Denote the index set of the network $\mathcal{N}$ using $\mathcal{I}_\mathcal{N} = \{1, \ldots, L\}$. Let $\mathcal{G} = \{\Theta_{m_1} \ \cdots \ \Theta_{m_S}\} \subseteq \mathcal{N}$ be a set of DeCEF layers with index set $\mathcal{I}_\mathcal{G} = \{m_1, \ldots, m_S\}$. Let $\tilde{\mathcal{G}} = \mathcal{N} \setminus \mathcal{G}$ be the rest of the layers in the network. Let $f(\mathcal{N})$ be an objective function and $\lambda\Phi(\tilde{\mathcal{G}})$ be a regularization term applied to the set $\tilde{\mathcal{G}}$, where $\lambda > 0$ is the multiplier. The optimization problem is formulated as:

$$\begin{aligned} \min \quad & f(\mathcal{N}) + \lambda\Phi(\tilde{\mathcal{G}}) \\ \text{subject to} \quad & \bar{\mathbf{U}}_l^{(i)\text{T}}\bar{\mathbf{U}}_l^{(i)} = \mathbf{I} \\ & \left\|\mathbf{a}_{l,j}^{(i)}\right\|_2 \le \epsilon, \ \forall l \in \mathcal{I}_\mathcal{G} \end{aligned} \tag{7}$$

2.4.2. Relaxed regularization

Finding an exact optimal DeCEF layer is an NP-hard problem due to the orthonormality constraint. Therefore, we approximate the constraint by the following regularizations. For a given DeCEF layer l, we have:

$$\Phi_1 : \lambda_1 \left\| \bar{\mathbf{U}}^{(i)\text{T}}\bar{\mathbf{U}}^{(i)} - \mathbf{I} \right\|_2 \tag{8}$$

$$\Phi_2 : \lambda_2 \left\|\mathbf{a}_j^{(i)}\right\|_2, \quad \mathbf{a}_j^{(i)} = \left[a_{1,j}^{(i)}, \ldots, a_{r,j}^{(i)}\right] \tag{9}$$

Note that the layer index l is neglected.

The loss function of the whole network is then written as:

$$f(\mathcal{N}) + \lambda\Phi(\tilde{\mathcal{G}}) + \lambda_1\Phi_1(\mathcal{G}) + \lambda_2\Phi_2(\mathcal{G}) \tag{10}$$

where $\Phi_i(\mathcal{G}) = \sum_{l\in\mathcal{I}_\mathcal{G}} \Phi_i^l$, $i = 1, 2$.

2.4.3. Deterministic rule-based hyperparameters

Hyperparameters are chosen based on deterministic rules to avoid complex hyperparameter tuning and to increase reproducibility. These rules are determined using a transfer learning approach. First, we find the hyperparameters in DeCEF using cross-validation on a small dataset CIFAR-10, where cross-validation is affordable. Then we establish a deterministic rule for each hyperparameter. These rules are then directly applied to the larger dataset ImageNet without tuning. There are three sets of hyperparameters h1 ∼ h3:

h1: Ranks r (Algorithm 1): The observation of singular values from several networks shows that the effective ranks typically have a decreasing trend with respect to the depth, i.e., layers at the beginning of the network often have higher rank, and vice versa. The idea of choosing the rank before training a network is to find a monotonically decreasing function given the increasing depth (cf. Fig. 2). In this paper, we adopt two alternative routines for choosing the rank in each layer: linear decay (simple) and logarithmic decay (aggressive). Let l be the depth index of a layer and $K = h^2$. Denote $l_{\max} = \max(l)$ and $l_{\min} = \min(l)$.

- Linear decay: $\hat{r}_i = \left\lfloor K - \frac{l(K-1)}{l_{\max} - l_{\min}} \right\rfloor$.
- Logarithmic decay: $\hat{r}_i = \left\lfloor \frac{K-1}{log2(l+1)} \right\rfloor$.

h2: Regularization coefficients (Algorithm 1, cf. Eqs. (8), (9)): $\lambda_1 = 10^{-4} r$ and $\lambda_2 = 10^{-4}$.

h3: Singular value threshold to determine the effective rank (cf. Eqs. (2), (3)): $\gamma = 0.3$.

A note on the rank function. Our observation indicates that the effective ranks of Conv2D layers have a decreasing tendency, which motivates us to choose a rank function that is decreasing over the depth of the network to approximate this behavior and enable the possibility of training a DeCEF network from scratch. Interestingly, this observation seems to be related to recent research on layer convergence bias [5], which states that shallower layers are carrying lower frequency information and they tend to converge sooner than deeper layers. A possible explanation for the decreasing rank function is that allowing a broader range of frequency components to pass from earlier layers to deeper layers grants the latter more freedom to learn high-frequency information. Since the most significant eigen-filters can be interpreted as low frequency components, if the ranks of shallower layers were set too low, high frequency information will be filtered out by the low rank approximation, and therefore will not be passed onto deeper layers for learning.

2.4.4. Training algorithm

The training algorithm is summarized in Algorithm 1.

Algorithm 1 (*DeCEF Training Strategy*).

- **Step 1:** Choose a network topology fully or partially composed of DeCEF layers. For example, one can replace all Conv2D layers with DeCEF layers.
- **Step 2:** Choose hyperparameters r, λ_1, λ_2.
- **Step 3:** Initialization for each DeCEF layer ($k = 1, \cdots, r$):
  - Eigen-filters $\mathbf{u}_k^{(i)}$:
    - Generate random matrices: $\mathbf{A}^{(i)} \in \mathbb{R}^{K \times r}$.
    - Compute the truncated SVD: $\mathbf{A}^{(i)} = \bar{\mathbf{U}}^{(i)} \bar{\mathbf{S}}^{(i)} \bar{\mathbf{V}}^{(i)\mathrm{T}}$.
    - Reshape each column in $\bar{\mathbf{U}}^{(i)}$ into matrix $\mathbf{u}_k^{(i)} \in \mathbb{R}^K$.
  - Coefficients $a_{k,j}^{(i)}$: randomly initialized from a normal distribution.

  Step 4: Forward and backward paths:
  - Forward $\mathbf{I}_l \to \mathbf{I}_{l+1}$: for each out channel j,

$$\mathbf{I}_{l+1}^j = \sum_{i=1}^{c_{\text{in}}} \sum_{k=1}^{r} a_{k,j}^{(i),l} \mathbf{u}_k^{(i),l} * \mathbf{I}_l^i$$

  - Backward: backpropagation with the loss function described in Eq. (10).

2.5. Refactor a Conv2D network into DeCEF

There are such use cases where a pre-trained CNN is available and one needs to reduce the runtime complexity of the network. This is not the focus of this work but we also propose a compression algorithm presented in Appendix B.

3. Related work

To compare to the state-of-the-art techniques, in this section, we list the following existing approaches.

Subspace techniques: The first category is the Low-Rank Approximation (LRA) technique. There are mainly two different approaches in the existing literature: (1) *Separable bases:* Jaderberg et al. [6] decomposes the $d \times d$ filters into $1 \times d$ and $d \times 1$ filters to construct rank-1 bases in the spatial domain. In later work, Tai et al. [7], Lin et al. [8], presents closed form solutions that significantly improves the efficiency over previous iterative optimization solvers are proposed. Ioannou et al. [9] introduces a novel weight initialization that allows small basis filters to be trained from scratch, which has achieved similar or higher accuracy than the conventional CNNs. Yu et al. [10] proposes a SVD-free algorithm that uses the idea that filters usually share smooth components in a low-rank subspace. Alvarez and Salzmann [11] introduces a regularizer that encourages the weights of the layers to have low rank during the training. More recently, Yang et al. [12] introduced SVD training, a method to achieve low-rank Deep Neural Networks (DNNs) that avoids costly singular value decomposition at every step. The Generalized Depthwise-Separable convolution [13] is an efficient post-training approximation for 2D convolutions in CNNs, improving throughput while preserving robustness. Yin et al. [14] proposes an ADMM-based framework for tensor decomposition in model compression, formulating tensor train decomposition as an optimization problem with tensor rank constraints and iteratively solving it to obtain high-accuracy tensor train-format DNN models for CNNs and Recurrent Neural Networks (RNNs). Li et al. [15] introduces a two-phase progressive genetic algorithm, PSTRN, which leverages the discovery of interest regions in rank elements to efficiently determine optimal ranks in tensor ring networks. Recently, Chen et al. [16] proposes joint matrix decomposition for CNN compression, leveraging shared structures to project weights into the same subspace, and offers three decomposition schemes with SVD-based optimization for improved compression results.

(2) *Filter vectorization:* Some existing work implements the low rank approximation by vectorizing the filters. For instance, Denton et al. [17] stacks all filters for each output channel into a high dimensional vector space and approximates the trained filters using SVD. Wen et al. [18] presents a regularization to enforce filters to coordinate into lower-rank space, where the subspaces are constructed from all the input channels for each given output channel. Later, Peng et al. [19] proposed a decomposition focusing on exploiting the filter group structure for each layer.

Pruning: Pruning refers to techniques that aim at reducing the number of parameters in a pre-trained network by identifying and removing redundant weights. This is a very invested topic in the attempt to reduce the model complexity. Although being different from our use case, we list the state-of-the-art pruning techniques in this section to have a more complete view on model reduction techniques. In Optimal Brain Damage by LeCun et al. [20], and later in Optimal Brain Surgeon by Hassibi et al. [21], redundant weights are defined by their impact on the objective function, which are identified using the Hessian of the loss function. Other definitions of redundancy have been proposed in subsequent work. For instance, Anwar et al. [22] applies pruning on the filter-level of CNNs by using particle filters to propose pruning candidates. Han et al. [23] introduces a simpler pruning method using a strong L2 regularization term, where weights under a certain threshold

are removed. Molchanov et al. [24] uses Taylor expansion to approximate the influence in the loss function by removing each filter. Hu et al. [25] iteratively optimizes the network by pruning unimportant neurons based on analysis of their outputs on a large dataset. Li et al. [26] identifies and removes filters having a small effect on the accuracy. Aghasi et al. [27] prunes a trained network layer-wise by solving a convex optimization program. Liu et al. [28] takes wide and large networks as input models, but during training insignificant channels are automatically identified and pruned afterwards. More recently, Luo and Wu [29], Luo et al. [30] analyzes the redundancy of filters in a trained network by looking at statistics computed from its next layer. He et al. [31] proposes an iterative LASSO regression based channel selection algorithm. Huang et al. [32] removes filters by training a pruning agent to make decisions for a given reward function. Yu et al. [33] poses the pruning problem as a binary integer optimization and derives a closed-form solution based on final response importance. Lin et al. [34] prunes filters across all layers by proposing a global discriminative function based on prior knowledge of each filter. Tung and Mori [35] combines network pruning and weight quantization in a single learning framework that performs pruning and quantization jointly. Zhang et al. [36] first formulate the weight pruning problem a nonconvex optimization problem constraints specifying the sparsity requirements and optimize using the alternating direction method of multipliers. Other work, such as Zhuang et al. [37], uses discrimination-aware losses into the network to increase the discriminative power of intermediate layers. Huang and Wang [38] adds a scaling factor to the outputs and then add sparsity regularizations on these factors. He et al. [39] compresses CNN models by pruning filters with redundancy, rather than those with "relatively less" importance. Lin et al. [40] proposes a scheme that incorporates two different regularizers which fully coordinates the global output and local pruning operations to adaptively prune filters. Later, Lin et al. [41], proposed an effective structured pruning approach that jointly prunes filters as well as other structures in an end-to-end manner by defining a new objective function with sparsity regularization which is solved by generative adversarial learning. Ding et al. [42] proposes a novel optimization method, which can train several filters to collapse into a single point in the parameter hyperspace which can be trimmed with no performance loss. Liu et al. [43] proposes a meta network, which is able to generate weight parameters for any pruned structure given the target network, which can be used to search for good-performing pruned networks. You et al. [44] introduce gate decorators to identify unimportant filters to prune. Molchanov et al. [45] prunes filters by using Taylor expansions to approximate a filter's contribution. Ding et al. [46] finds the least important filters to prune by a binary search. Luo and Wu [47] proposes an efficient channel selection layer to find less important filters automatically in a joint training manner. Lin et al. [48] proposes a method that is mathematically formulated to prune filters with low-rank feature maps. He et al. [49] introduces a differentiable pruning criteria sampler. Ding et al. [50] proposes a re-parameterization of CNNs to a remembering part and a forgetting part. The former learns to maintain the performance and the latter learns for efficiency. Liu et al. [51] proposes a layer grouping algorithm to find coupled channels automatically. Shi et al. [52] uses an effective estimation of each filter, i.e., saliency, to measured filters from two aspects: the importance for prediction performance and the consumed computational resources. This can be used to preserve the prediction performance while zeroing out more computation-heavy filters.

Architectural design: Effort has been put into designing a smaller network architecture without loss of the generalization ability. For instance, He et al. [53] achieves a higher accuracy in [53,54] compared to other more complex networks by introducing the residual building block. The residual building blocks adds an identity mapping that allows the signals to be directly propagated between the layers. Iandola et al. [55] introduces SqueezeNet and the Fire module, which is designed to reduce the number of parameters in a network by introducing 1×1 filters. By utilizing dense connections pattern between blocks, Huang et al. [56] manages to reduce the number of required parameters. Xie et al. [57] proposed a multi-branch architecture which exposes a new hyperparameter for each block to control the capacity of the network. Other work, like MobileNet [58,59] and EfficientNet [60] specifically focus on builing architectures suitable for devices with low compute capacity, such as mobile phones. By a design that maintains a high-resolution representation throughout the whole network, Wang et al. [61] achieves good accuracy and performance in HRNet.

Compression: Deep Compression, by Han et al. [62], reduces the storage size of the model using quantization and Huffman encoding to compress the weights in the network. Other work on reducing the memory size of models is done by binarization. In XNOR-Net by Rastegari et al. [63], the weights are reduced to a binary representation and convolutions are replaced by XNOR operations. More recently, Suau et al. [64] proposed to analyze filter responses to automatically select compression methods for each layer.

Weight sharing: Another approach to reduce the number of parameters in a network is to share weights between the filters and layers. Boulch [65] share weights between the layers in a residual network operating on the same scale.

Depthwise separable convolutions: introduced by Chollet [66], have shown to be a more efficient use of parameters compared to regular Conv2Ds Inception like architectures. Depthwise separable convolutions have also been used in other work, e.g., [31], where it was used to gain a computational speed-up of ResNet networks.

Our focus: We observe and analyze the Conv2D from a different perspective compared to the previous subspace techniques. More specifically, (i) we vectorize the filters instead of using separable basis in the original vector space [6,7,9,10]; (ii) we do not concatenate these vectorized filters into a large vector space [17–19], which achieves a better modularity compared to the concatenated vectors. Our perspective is motivated by the empirical evidence from our experiments. This opens up new opportunities and provides new analytical tools for understanding the design of convolutional networks with respect to their subspace redundancies. In our experiments, we choose a popular base network (ResNet) and compare our experimental results to various modifications of the same base network. We also conduct tests on other more recent network architectures such as HRNet-W18-C and DenseNet-121 for further comparison and validation.

4. Experiments and results

4.1. Hardware

For training and experiments, Nvidia Tesla V100 SXM2 with 32 GB of GPU memory are used.

4.2. Dataset CIFAR-10: Ablation study

Dataset: To empirically study the behavior of DeCEF, we conduct various experiments on the standard image recognition dataset CIFAR-10 by Krizhevsky and Hinton [67]. **Benchmark:** We use ResNet-32 as the base net for comparison. ResNet-32 has three **blocks**, where the last block (block-3) in ResNet-32 has the most filters. Since our goal is to reduce the amount of trainable parameters and FLOPs, we mainly vary the structure in block-3 in our experiments.

Experiments: We design four experiments as follows.

Experiment 1. varying rank r and c_{out}. For a layer with input channels $i = 1, \ldots, c_{\text{in}}$ and output channels $j = 1, \ldots, c_{\text{out}}$, the filters in the DeCEF layer is expressed as $\mathbf{w}_j^{(i)} = \sum_{k=1}^{r} a_{k,j}^{(i)} \mathbf{u}_k^{(i)}$. We empirically show that DeCEF layers achieve higher accuracy with significantly lower number of parameters. In this experiment, we vary two hyperparameters: (1) the rank r of each filter in the DeCEF layer, and (2) the number of output channels c_{out}. We compare the accuracy versus the number of parameters in different types of layers (Conv2D and DeCEF with

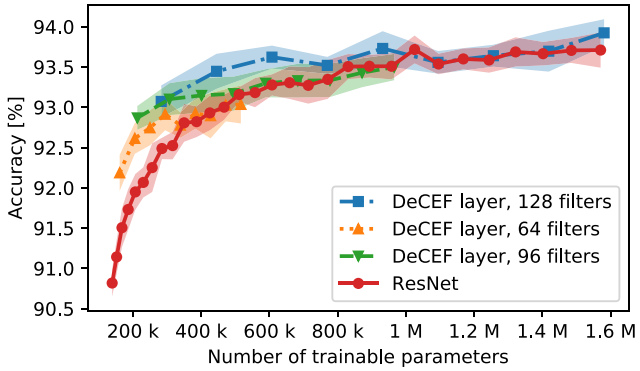


Fig. 3. Accuracy versus number of parameters on CIFAR-10.

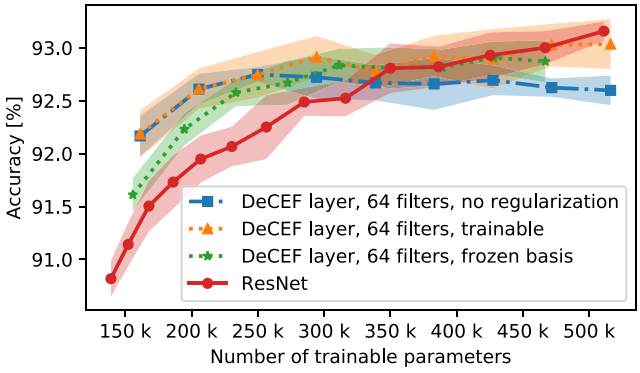


Fig. 4. DeCEF layer with trainable vs. frozen bases on CIFAR-10.

different hyperparameters). As shown in Fig. 3, with a lower number of parameters, DeCEF achieves a better accuracy with low rank techniques. Moreover, when we increase the number of output channels, DeCEF shows a even more promising result with fewer parameters in total.

Experiment 2. trainable vs. frozen eigen-filters. In Algorithm 1, the eigen-filters in DeCEF layers are trained simultaneously using backpropagation. In this experiment, we investigate the impact of this training process and try to understand if it is sufficient to use *random basis vectors* as eigen-filters. We initialize the eigen-filters according to Algorithm 1 and freeze them during training. The comparison between the accuracies achieved by frozen and trainable eigen-filters can be found in Fig. 4. By using frozen eigen-filters, the network has a fewer number of trainable parameters for the same rank. With a low rank ($r < 5$), the accuracy is degraded without training.

Experiment 3. with or without Φ_1 regularization. To study the effect of Φ_1 introduced in Eq. (8), some experiments can be found in Fig. 4. We can see that with a high rank, the regularization needs to be applied. In our experiment, we use $\lambda_1 = 10^{-4}r$ and $\lambda_2 = 10^{-4}$, where λ_1 is the multiplier of the constraint on the eigen-filters and λ_2 is on the subspace coefficients. The reason for having the multiplier r in λ_1 is to suppress the growth of the cost when r becomes large.

Experiment 4. comparison to related work. In this experiment, we implement Algorithm 1 (DeCEF-ResNet-32) to compared to the state-of-the-art techniques. We vary the number of output channels c_{out} in the last ResNet block for comparison, where we see that having fewer eigen-filters with more output channels yields a better result.

Results: The results are presented in terms of the estimated mean and the standard deviation of the classification accuracy on the testing set with 10 runs for each experimental setup, which are shown in Figs. 3 and 4. The accuracy is then presented with respect to the number of trainable parameters for each network structure. For DeCEF layers, there are nine data points in each presented result, which correspond to different layer ranks in block-3 $r_3 \in \{1, \dots, 9\}$. In addition, the number of trainable parameters in DeCEF layers is also varied by using different numbers of output channels in block-3, i.e., $c_{\text{out}} \in \{64, 96, 128\}$. We then vary c_{out} in ResNet-32 block-3 ($c_{\text{out}} \in \{16, 20, 24, \dots, 128\}$) to have a comparable result. We compare the accuracy achieved by DeCEF-ResNet-32 Fig. 5.

4.3. Dataset ImageNet (ILSVRC-2012)

To further compare our algorithms to the state-of-the-art, we use the standard dataset ImageNet (ILSVRC-2012) by Deng et al. [69]. ImageNet has 1.2 M training images and 50 k validation images of 1000 object classes, commonly evaluated by Top-1 and Top-5 accuracy. We use the networks ResNet-50 v2 [54], DenseNet-121 [56] and HRNet-W18-C [61] as the base networks. The results are visualized in Figs. 6 and 7 for Top-1 and Top-5 accuracy, respectively.

The hyperparameters used in DeCEF-ResNet-50 are determined by the deterministic rules presented in **h1**, **h2** and **h3**. For each setup, we have five runs and report the average accuracy and its standard deviation. From the experiments, we see the trade-off between the two rank decay mechanisms: linear decay is less aggressive, which yield to a better accuracy, whereas logarithmic decay reduce a greater number of FLOPs while still having a decent accuracy. To further validate DeCEF, we run the same experiments on three commonly used base networks. The results are reported in Tables 2 and 3 to compare with the corresponding base network and state-of-the-art model reduction techniques.

4.4. Comparison to related work

Various configurations of the DeCEF method are compared to related work on two different datasets, CIFAR-10 and ImageNet.

For CIFAR-10, the DeCEF-ResNet-32 (32, 64, 128) configuration achieves competitive accuracy while having fewer parameters and lower computational complexity than comparable competitive methods, such as HRank ResNet-110 [48] and SASL ResNet-110 [52].

The DeCEF-ResNet-32 (24, 48, 96) configuration also demonstrates comparable accuracy compared to SASL ResNet-56 and HRank ResNet-56 with significantly fewer parameters and lower computational requirements, and superior accuracy compared to GBN-40 [44] with similar complexities. Among pruning and subspace approximation techniques, ResRep ResNet-110 [50] outperforms DeCEF in terms of accuracy and complexity (94.19%, 108 MFLOPS vs. 94.62%, 105.68 MFLOPS). However, as ReRep is a channel pruning technique, it can be easily combined with DeCEF to achieve better efficiency.

On the ImageNet dataset, the DeCEF-ResNet-50 (log decay) configuration showcases competitive accuracy while maintaining fewer parameters and lower computational complexities compared to alternative methods such as Taylor-FO-BN-91% [45] fewer parameters than ShaResNet-101 and ShaResNet-152 (FLOPs not reported), and lower computational complexities than GFP ResNet-50 1 [51] (parameters not reported). The model has similar complexities as GBN-60 while achieving a higher accuracy. Additionally, the DeCEF-HRNet-W18-C (log decay) configuration achieves competitive accuracy with significantly fewer parameters and superior computational efficiency compared to ResRep ResNet-50 1, ResRep ResNet-50 2, GBN-50, and SSS-ResNetXt-38 [38].

Based on these observations, the DeCEF method shows promise in enhancing model efficiency, making it a valuable approach to consider in deep learning model development.

Combining DeCEF with state-of-the-art deep learning models that achieve high accuracy with relatively low computational requirements (GFLOPs) such as the EfficientNet family [60], InceptionResNetV2 [71], Xception [66], and Inception V3 [72] (by, for example, replacing the depthwise convolutional layers with DeCEF layers) could potentially

Table 1
Comparison to state-of-the-art model reduction techniques on CIFAR-10.

Network	Acc.	Std.	No. param.	MFLOPs
(a) DeCEF vs. baseline network				
DeCEF-ResNet-32 (32, 64, 128)[0]	94.19%	(0.18%)	533.00 k	108.00
DeCEF-ResNet-32 (24, 48, 96)[1]	93.64%	(0.16%)	311.00 k	64.72
ResNet-110[2] [53]	93.57%		1.72 M	252.89
ResNet-56[3] [53]	93.03%		850.00 k	125.49
ResNet-32[4] [53]	92.49%		467.00 k	69.00
DeCEF-ResNet-32 (16, 32, 64)[5]	92.45%	(0.17%)	148.00 k	32.42
(b) Related work				
ResRep ResNet-110[6] [50]	94.62%			105.68
C-SGD-5/8 ResNet-110[7] [42]	94.44%			98.91
HRank ResNet-110 1[8] [48]	94.23%		1.04 M	148.70
SASL ResNet-110[9] [52]	93.99%		1.17 M	122.15
SFP ResNet-110 20%[10] [68]	93.93%			182.00
SFP ResNet-56 10%[11] [68]	93.89%			107.00
SASL ResNet-56[12] [52]	93.88%		689.35 k	80.44
SFP ResNet-110 30%[13] [68]	93.86%			150.00
Bi-JSVD0.7 ResNet-16 11.99[14] [16]	93.84%	(0.09%)	930.78 k	373.00
SFP ResNet-110 10%[15] [68]	93.83%			216.00
SASL* ResNet-110[16] [52]	93.80%		786.04 k	75.36
ShaResNet-164[17] [65]	93.80%		930.00 k	
LFPC ResNet-110[18] [49]	93.79%			101.00
SFP ResNet-56 30%[19] [68]	93.78%			74.00
FPGM-only 40% ResNet-110[20] [39]	93.74%			121.00
ResRep ResNet-56 1[21] [50]	93.73%			59.09
LFPC ResNet-56 1[22] [49]	93.72%			66.40
GAL-0.1 ResNet-110[23] [41]	93.59%		1.65 M	205.70
SASL* ResNet-56[24] [52]	93.58%		538.90 k	53.84
ResNet-110-pruned-A[25] [26]	93.55%		1.68 M	213.00
HRank ResNet-56 1[26] [48]	93.52%		710.00 k	88.72
FPGM-only 40% ResNet-56[27] [39]	93.49%			59.40
SFP ResNet-56 20%[28] [68]	93.47%			89.80
C-SGD-5/8 ResNet-56[29] [42]	93.44%			49.13
GBN-40[30] [44]	93.43%		395.25 k	50.07
GAL-0.6 ResNet-56[31] [41]	93.38%		750.00 k	78.30
NISP-110[32] [33]	93.38%		976.10 k	
HRank ResNet-110 2[33] [48]	93.36%		700.00 k	105.70
SFP ResNet-56 40%[34] [68]	93.35%			59.40
LFPC ResNet-56 2[35] [49]	93.34%			59.10
SFP ResNet-32 10%[36] [68]	93.22%			58.60
RJSVD-1 ResNet-16 17.76[37] [16]	93.19%	(0.04 %)	628.38 k	350.00
HRank ResNet-56 2[38] [48]	93.17%		490.00 k	62.72
ResNet-56-pruned-A[39] [26]	93.10%		770.10 k	112.00
GBN-30[40] [44]	93.07%		283.05 k	37.27
ResNet-56-pruned-B[41] [26]	93.06%		733.55 k	90.90
ResNet-110-pruned-B[42] [26]	93.00%		1.16 M	115.00
NISP-56[43] [33]	92.99%		487.90 k	81.00
ADMM TT ResNet-32[44] [14]	92.87%		97.29 k	
FPGM-mix 40% ResNet-32[45] [39]	92.82%			32.30
GAL-0.5 ResNet-110[46] [41]	92.74%		950.00 k	130.20
ResRep ResNet-56 2[47] [50]	92.67%			27.82
SVD ResNet-32 Spatial Hoyer[48] [12]	92.66%			26.72
HRank ResNet-110 3[49] [48]	92.65%		530.00 k	79.30
LFPC ResNet-32[50] [49]	92.12%			32.70
nin-c3-lr[51] [9]	91.78%		438.00 k	104.00
GAL-0.8 ResNet-56[52] [41]	91.58%		290.00 k	49.99
PSTRN-S ResNet-32[53] [15]	91.44%		180.00 k	
HRank ResNet-56 3[54] [48]	90.72%		270.00 k	32.52
SFP ResNet-32 20%[55] [68]	90.63%			49.00
SFP ResNet-32 30%[56] [68]	90.08%			40.30

Table 2
Comparison to the base networks on ImageNet.

	Layers	Rank decay	Top-1	Top-5	params	(G)FLOPs
ResNet-50	Conv2D	None	76.47%	93.21%	25.56M	3.80
	DeCEF	Linear	**76.61%**	93.22%	17.27M	2.90
	DeCEF	Logarithmic	76.46%	**93.24%**	**16.64M**	**2.50**
DenseNet-121	Conv2D	None	74.81%	92.32%	79.79M	2.83
	DeCEF	Linear	**74.85%**	**92.61%**	72.10M	2.81
	DeCEF	Logarithmic	74.40%	91.89%	**62.92M**	**2.11**
HRNet-W18-C	Conv2D	None	**77.00%**	**93.50%**	21.30M	3.99
	DeCEF	Linear	76.17%	92.99%	9.490M	2.55
	DeCEF	Logarithmic	75.11%	92.47%	**7.05M**	**1.27**

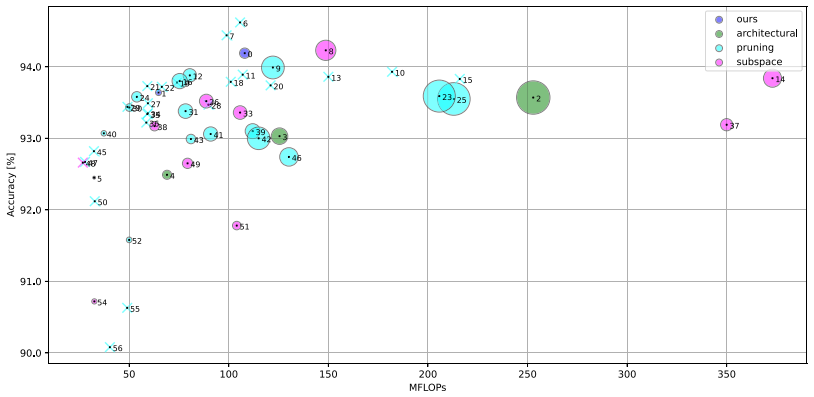


Fig. 5. Ball chart for CIFAR-10, where the size of the ball indicates the number of trainable parameters. For papers that have not reported the FLOPs, we use a cross instead of a ball to represent them. The exact values are reported in Table 1. The number in each ball is the network ID, which is indicated as the superscript of each entry in Table 1.

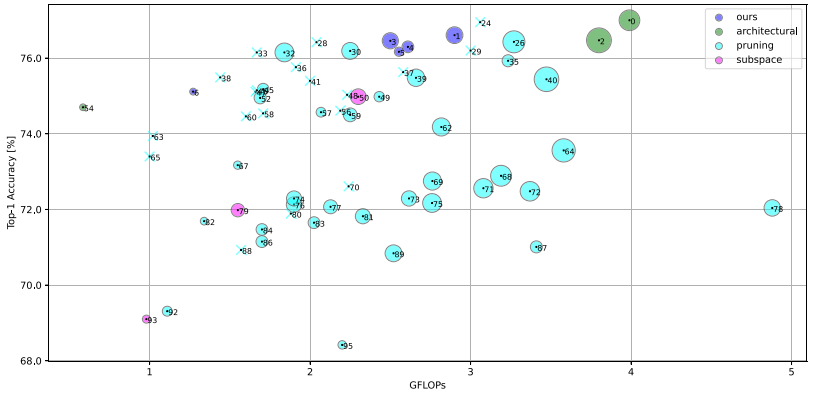


Fig. 6. Ball chart for ImageNet Top-1 accuracy with the same set up as Fig. 5. The corresponding values can be found in Table 3.

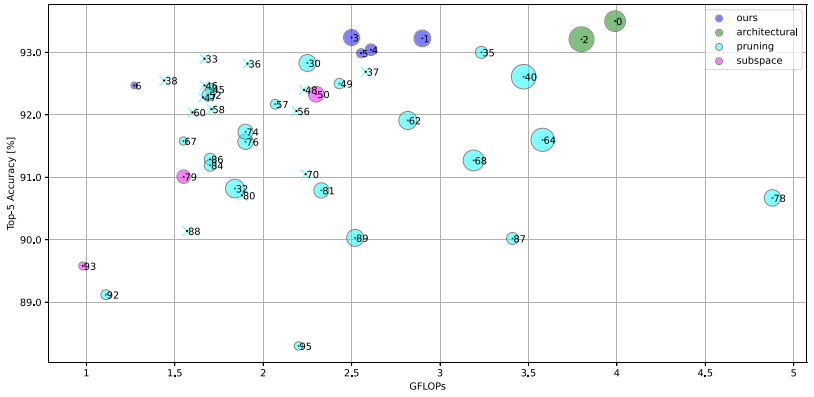


Fig. 7. Ball chart for ImageNet Top-5 accuracy with the same set up as Fig. 5. The corresponding values can be found in Table 3.

improve both performance and efficiency, as these models primarily focus on architectural designs.

Moreover, exploring the integration of channel pruning techniques like those employed in ResRep with DeCEF could serve as a potential future direction, further boosting the efficiency and accuracy of these configurations.

5. Conclusion and future work

In this paper, we propose a new methodology to observe and analyze the redundancy in a CNN. Motivated by our observations of the low rank behaviors in vectorized Conv2D filters, we present a layer structure DeCEF as an alternative parameterization to Conv2D filters for the purpose of reducing their complexity in terms of trainable parameters and FLOPs. Our experiments have shown that in a convolutional layer with filter size $h \times h$, it is not necessary to have more than h^2 eigen-filters given the training strategy in Section 2.4.

In terms of the accuracy-to-complexity ratio, it is beneficial to use more coefficients (i.e. output channels) with fewer eigen-filters in DeCEF layers. The DeCEF layer is simple to implement in most deep learning frameworks using depthwise separable convolutions with a new training strategy. With the deterministic rules for choosing the effective ranks, it is easy to design and reproduce the results. From our observations, the underlying subspace structure is a commonly shared property among different network architectures and topologies, which provides insights into the design and analysis of CNNs.

For future work, we plan to delve deeper into this low-rank structure to optimize the selection of effective ranks by exploring more advanced

Table 3
Comparison to state-of-the-art model reduction techniques on ImageNet.

Network	Top-5 Acc.	Std.	Top-1 Acc.	Std.	No. param.	GFLOPs
(a) DeCEF vs. baseline network						
HRNet-W18-C[0] [61]	93.50%		77.00%		21.30 M	3.99
DeCEF-ResNet-50 (lin decay)[1]	93.22%	(0.07%)	76.61%	(0.06%)	17.27 M	2.90
ResNet-50[2] [54]	93.21%		76.47%		25.56 M	3.80
DeCEF-ResNet-50 (log decay)[3]	93.24%	(0.05%)	76.46%	(0.05%)	16.64 M	2.50
DeCEF-HRNet-W18-C (inferred rank)[4]	93.04%		76.30%		12.06 M	2.61
DeCEF-HRNet-W18-C (lin decay)[5]	92.99%		76.17%		9.49 M	2.55
DeCEF-HRNet-W18-C (log decay)[6]	92.47%		75.11%		7.05 M	1.27
(b) Related work						
EfficientNet-B7[7] [60]	96.84%		84.43%		64.10 M	
EfficientNet-B6[8] [60]	96.90%		84.08%		41.00 M	
EfficientNet-B5[9] [60]	96.71%		83.70%		28.50 M	
EfficientNet-B4[10] [60]	96.26%		82.96%		17.70 M	
NASNetLarge[11] [70]	96.00%		82.50%		84.90 M	
EfficientNet-B3[12] [60]	95.68%		81.58%		10.80 M	
InceptionResNetV2[13] [71]	95.25%		80.26%		54.30 M	
EfficientNet-B2[14] [60]	94.95%		80.18%		7.80 M	
EfficientNet-B1[15] [60]	94.45%		79.13%		6.60 M	
Xception[16] [66]	94.50%		79.00%		22.86 M	
ResNet152V2[17] [54]	94.16%		78.03%		58.30 M	
InceptionV3[18] [72]	93.72%		77.90%		21.80 M	
ShaResNet-152[19] [65]	93.86%		77.77%		36.80 M	
DenseNet201[20] [56]	93.62%		77.32%		18.30 M	
ResNet101V2[21] [54]	93.82%		77.23%		42.60 M	
EfficientNet-B0[22] [60]	93.49%		77.19%		4.00 M	
ShaResNet-101[23] [65]	93.45%		77.09%		29.40 M	
GFP ResNet-50 1[24] [51]			76.95%			3.06
ResNet152[25] [53]	93.12%		76.60%		58.40 M	
Taylor-FO-BN-91%[26] [45]			76.43%		22.60 M	3.27
ResNet101[27] [53]	92.79%		76.42%		42.70 M	
GFP ResNet-50 2[28] [51]			76.42%			2.04
MetaPruning 0.85 ResNet-50[29] [43]			76.20%			3.00
GBN-60[30] [44]	92.83%		76.19%		17.42 M	2.25
DenseNet169[31] [56]	93.18%		76.18%		12.60 M	
ResNet-50 GAL-0.5-joint[32] [41]	90.82%		76.15%		19.31 M	1.84
ResRep ResNet-50 1[33] [50]	92.90%		76.15%			1.67
ResNet50V2[34] [54]	93.03%		75.96%		23.60 M	
SSS-ResNetXt-41[35] [38]	93.00%		75.93%		12.40 M	3.23
SASL[36] [52]	92.82%		75.76%			1.91
AOFP-C1[37] [46]	92.69%		75.63%			2.58
ResRep ResNet-50 2[38] [50]	92.55%		75.49%			1.44
Taylor-FO-BN-81%[39] [45]			75.48%		17.90 M	2.66
SSS-ResNet-41[40] [38]	92.61%		75.44%		25.30 M	3.47
MetaPruning 0.75 ResNet-50[41] [43]			75.40%			2.00
ShaResNet-50[42] [65]	92.59%		75.39%		20.50 M	
MobileNetV2(alpha = 1.4)[43] [59]	92.42%		75.23%		4.40 M	
ResNet-50 Variational[44] [73]	92.10%		75.20%		15.30 M	
GBN-50[45] [44]	92.41%		75.18%		11.91 M	1.71
SASL*[46] [52]	92.47%		75.15%			1.67
AOFP-C2[47] [46]	92.28%		75.11%			1.66
ResNet-50 FPGM-only 30%[48] [39]	92.40%		75.03%			2.23
SSS-ResNetXt-38[49] [38]	92.50%		74.98%		10.70 M	2.43
ResNet-50 HRank 1[50] [48]	92.33%		74.98%		16.15 M	2.30
DenseNet121[51] [56]	92.26%		74.97%		7.00 M	
DCP[52] [37]	92.32%		74.95%		12.41 M	1.69
ResNet50[53] [53]	92.06%		74.93%		23.60 M	
MobilenetV2[54] [59]			74.70%		6.90 M	0.58
MobileNetV2(alpha = 1.3)[55] [59]	92.12%		74.68%		3.80 M	
SFP[56] [68]	92.06%		74.61%			2.19
SSS-ResNetXt-35-A[57] [38]	92.17%		74.57%		10.00 M	2.07
C-SGD-50[58] [42]	92.09%		74.54%			1.71
Taylor-FO-BN-72%[59] [45]			74.50%		14.20 M	2.25
LFPC[60] [49]	92.04%		74.46%			1.60
NASNetMobile[61] [70]	91.85%		74.37%		4.30 M	
SSS-ResNet-32[62] [38]	91.91%		74.18%		18.60 M	2.82
GFP ResNet-50 3[63] [51]			73.94%			1.02
Pruned-90[64] [29]	91.60%		73.56%		23.89 M	3.58
MetaPruning 0.5 ResNet-50[65] [43]			73.40%			1.00
ShaResNet-34[66] [65]	90.58%		73.27%		13.60 M	
SSS-ResNetXt-35-B[67] [38]	91.58%		73.17%		8.50 M	1.55
Pruned-75[68] [29]	91.27%		72.89%		21.47 M	3.19
NISP-50-A[69] [33]			72.75%		18.63 M	2.76

(continued on next page)

Table 3 (*continued*).

GDP 0.7[70] [34]	91.05%	72.61%		2.24
ResNet-34-pruned-A[71] [26]		72.56%	19.90 M	3.08
ResNet-34-pruned-C[72] [26]		72.48%	20.10 M	3.37
NISP-34-A[73] [33]		72.29%	15.74 M	2.62
ResNet-50 SSR-L2,0 A[74] [40]	91.73%	72.29%	15.50 M	1.90
ResNet-34-pruned-B[75] [26]		72.17%	19.30 M	2.76
ResNet-50 SSR-L2,1 A[76] [40]	91.57%	72.13%	15.90 M	1.90
NISP-50-B[77] [33]		72.07%	14.36 M	2.13
ThiNet-70[78] [30]	90.67%	72.04%	16.94 M	4.88
ResNet-50 HRank 2[79] [48]	91.01%	71.98%	13.77 M	1.55
GDP 0.6[80] [34]	90.71%	71.89%		1.88
SSS-ResNet-26[81] [38]	90.79%	71.82%	15.60 M	2.33
Taylor-FO-BN-56%[82] [45]		71.69%	7.90 M	1.34
NISP-34-B[83] [33]		71.65%	12.17 M	2.02
ResNet-50 SSR-L2,0 B[84] [40]	91.19%	71.47%	12.00 M	1.70
MobileNetV2(alpha = 1.0)[85] [59]	90.14%	71.34%	2.30 M	
ResNet-50 SSR-L2,1 B[86] [40]	91.29%	71.15%	12.20 M	1.70
ThiNet-50[87] [30]	90.02%	71.01%	12.38 M	3.41
GDP 0.5[88] [34]	90.14%	70.93%		1.57
Pruned-50[89] [29]	90.03%	70.84%	17.38 M	2.52
MobileNet(alpha = 1.0)[90] [58]	89.50%	70.42%	3.20 M	
MobileNetV2(alpha = 0.75)[91] [59]	89.18%	69.53%	1.40 M	
ResNet-50 GAL-1-joint[92] [41]	89.12%	69.31%	10.21 M	1.11
ResNet-50 HRank 3[93] [48]	89.58%	69.10%	8.27 M	0.98
GreBdec (VGG-16)[94] [10]	89.06%	68.75%	9.70 M	
ThiNet-30[95] [30]	88.30%	68.42%	8.66 M	2.20
MobileNet(alpha = 0.75)[96] [58]	88.24%	68.41%	1.80 M	
GreBdec (GoogLeNet)[97] [10]	88.11%	68.30%	1.50 M	

strategies, such as integrating learnable rank functions. For instance, in certain network architectures, the effective rank shows an initial increase followed by a rapid decrease – a phenomenon we aim to account for in our approach. Moreover, since the DeCEF layer can be implemented by the depthwise separable convolutions with a new training strategy, a second future direction is to modify and train the traditional depthwise separable convolutional layers in well-known networks using DeCEF to reduce the model complexity. Finally, during the experiments, we have come up with several hypotheses regarding the low rank behaviors in deep neural networks that we plan to explore. In particular, we will investigate the convergence properties, such as speed and stability with respect to initialization for the subspaces in different layers to better understand and interpret a CNN from this perspective.

CRediT authorship contribution statement

Yinan Yu: Methodology development, Software implementation, Experimental design and analysis, Writing. **Samuel Scheidegger:** Methodology development, Software implementation, Experimental design and analysis, Writing. **Tomas McKelvey:** Methodology development, Results analysis, Review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgment

This work is partially funded by the Chalmers Artificial Intelligence Research Centre (CHAIR) through the Vermillion project.

Appendix A. Proof of Lemma 1

Proof. For each input channel i, given an additive perturbation matrix $\Delta\mathbf{I}_i$, let $\tilde{\mathbf{I}}_i = \mathbf{I}_i + \Delta\mathbf{I}_i$. Given optimal parameters of kernel j expressed as $\mathbf{w}_j^{(i)} = \sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k^{(i)}$, which are learned from the training data, the output of the convolutional layer is

$$\begin{aligned}\mathbf{I}_j &= \sum_i \left(\mathbf{I}_i + \Delta\mathbf{I}_i\right) * \mathbf{w}_j^{(i)} = \sum_i \left(\mathbf{I}_i + \Delta\mathbf{I}_i\right) * \sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k^{(i)} \\ &= \sum_i \mathbf{I}_i * \sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k + \sum_i \Delta\mathbf{I}_i * \sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k^{(i)} \\ &= \mathbf{I}_j^* + \underbrace{\sum_i \Delta\mathbf{I}_i * \sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k^{(i)}}_{\text{perturbation term}}\end{aligned}$$

where $\mathbf{I}_j^*$ denotes the optimal feature map. By using the infinity norm to characterize the effect of the perturbation, we have:

$$\left\|\mathbf{I}_j^* - \mathbf{I}_j\right\|_\infty = \left\|\sum_i \Delta\mathbf{I}_i * \sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k^{(i)}\right\|_\infty \tag{A.1}$$

$$\leq \sum_i \left\|\Delta\mathbf{I}_i * \sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k^{(i)}\right\|_\infty \tag{A.2}$$

From Young's inequality:

$$\begin{aligned}\text{(A.1)} &\leq \sum_i \left\|\Delta\mathbf{I}_i\right\|_2 \left\|\sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k^{(i)}\right\|_2 \\ &\leq \sum_i \left\|\Delta\mathbf{I}_i\right\|_2 \left\|\sum_{k=1}^{r} a_{k,j}^{(i)}\mathbf{u}_k^{(i)}\right\|_2 \\ &\leq \sum_i \left\|\Delta\mathbf{I}_i\right\|_2 \sum_{k=1}^{r} \left|a_{k,j}^{(i)}\right| \left\|\mathbf{u}_k^{(i)}\right\|_2\end{aligned}$$

$$\leq \sum_i \left\|\Delta\mathbf{I}_i\right\|_2 \sum_{k=1}^{r} \left|a_{k,j}^{(i)}\right| \left\|\mathbf{u}_k^{(i)}\right\|_F \tag{A.3}$$

$$\leq \sum_i \left\|\Delta\mathbf{I}_i\right\|_2 \left\|\mathbf{a}_j^{(i)}\right\|_1 \sum_{k=1}^{r} \left\|\mathbf{u}_k^{(i)}\right\|_F$$

where $\mathbf{a}_j^{(i)} = \begin{bmatrix} a_{1,j}^{(i)} & \cdots & a_{r,j}^{(i)} \end{bmatrix}^{\mathrm{T}}$ and $\|\cdot\|_F$ denotes the Frobenius norm. Let $\bar{\mathbf{u}}_k^{(i)} = \mathrm{vect}(\mathbf{u}_k^{(i)})$, where $\mathrm{vect}(\cdot)$ denotes the vectorization of a matrix. If $\bar{\mathbf{U}}^{(i)\mathrm{T}}\bar{\mathbf{U}}^{(i)} = \mathbf{I}$, we have $\left\|\mathbf{u}_k^{(i)}\right\|_F = 1$ and hence

$$(A.3) \le \sum_i r \left\|\mathbf{a}_j^{(i)}\right\|_1 \|\Delta\mathbf{I}_i\|_2 \le \sum_i rh \left\|\mathbf{a}_j^{(i)}\right\|_2 \|\Delta\mathbf{I}_i\|_2$$

where h is the kernel size. If $\left\|\mathbf{a}_j^{(i)}\right\|_2 \le \epsilon$, $\forall i, j$, then

$$\left\|\sum_i \Delta\mathbf{I}_i * \mathbf{w}_j^{(i)}\right\|_\infty \le \epsilon hr \sum_i \|\Delta\mathbf{I}_i\|_2 \quad \square$$

Appendix B. Network compression using DeCEF layers

One application of DeCEF is to use it as a model reduction technique for a pre-trained network. As discussed in the paper, this is not the main focus of DeCEF. Nevertheless, we propose an algorithm as follows for this type of applications.

Algorithm 2 (*DeCEFC-basenet*)**.**

> **Step 1:** Analysis described in Procedure 1.
> **Step 2:** For each layer, let $\bar{\mathbf{u}}_k^{(i)}$ be the columns of $\bar{\mathbf{U}}^{(i)}$. Approximate $\mathbf{w}_j^{(i)}$ by $\mathbf{w}_j^{(i)} \approx \sum_{k=1}^{r_i} a_{k,j}^{(i)} \mathbf{u}_k^{(i)}$, where $\mathbf{u}_k^{(i)}$ is obtained by reshaping $\bar{\mathbf{u}}_k^{(i)}$ into a $h \times h$ matrix.
> **Step 3 (optional):** Network fine-tuning by freezing the eigen-filters and training the other trainable parameters.

Appendix C. Convergence video

In this supplementary material, we include videos (in the file called "effective_rank_converge_video.zip") to show some examples of how the effective ranks (cf. Eq. (2) in the paper) in each layer converge over training epochs. The title of each video indicates the layer index, i.e. the larger the index, the deeper the layer is. The network and data used here are the DenseNet-121 and ImageNet, respectively.

In the video, the leftmost rectangular box shows the singular values computed from all the output channels (filters) for each input channel. Each row in this figure contains the singular values for one input channel. The image in the middle is the histogram density of the effective rank. Each frame in this video shows the singular values and effective ranks computed from one epoch. Finally, the image on the right shows the convergence of this effective rank over raining epochs. Note that due to the limit on the file size, we only show the convergence for every fourth layer.

Appendix D. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.neucom.2024.128461.

References

[1] S. Luccioni, Y. Jernite, E. Strubell, Power Hungry Processing: Watts Driving the Cost of AI Deployment?, Association for Computing Machinery, New York, NY, USA, 2024, http://dx.doi.org/10.1145/3630106.3658542.

[2] P. Belhumeur, J. Hespanha, D. Kriegman, Eigenfaces vs fisher faces recognition using class specific linear projection, IEEE Trans. Pattern Anal. Mach. Intell. 19 (1997) 711–720.

[3] I.T. Jolliffe, Principal Component Analysis, Springer-Verlag, 1986.

[4] G. Golub, C. van Loan, Matrix Computations, third ed., Johns Hopkins Press, 1996.

[5] Y. Chen, A. Yuille, Z. Zhou, Which layer is learning faster? A systematic exploration of layer-wise convergence rate for deep neural networks, in: The Eleventh International Conference on Learning Representations, 2023, URL: https://openreview.net/forum?id=wlMDF1jQF86.

[6] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, in: Proceedings of the British Machine Vision Conference, BMVA Press, 2014, http://dx.doi.org/10.5244/C.28.88.

[7] C. Tai, T. Xiao, Y. Zhang, X. Wang, et al., Convolutional neural networks with low-rank regularization, 2015, arXiv preprint arXiv:1511.06067.

[8] S. Lin, R. Ji, C. Chen, D. Tao, J. Luo, Holistic cnn compression via low-rank decomposition with knowledge transfer, IEEE Trans. Pattern Anal. Mach. Intell. 41 (12) (2018) 2889–2905.

[9] Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, A. Criminisi, Training cnns with low-rank filters for efficient image classification, 2015, arXiv preprint arXiv:1511.06744.

[10] X. Yu, T. Liu, X. Wang, D. Tao, On compressing deep models by low rank and sparse decomposition, in: The IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017.

[11] J.M. Alvarez, M. Salzmann, Compression-aware training of deep networks, in: Advances in Neural Information Processing Systems, 2017, pp. 856–867.

[12] H. Yang, M. Tang, W. Wen, F. Yan, D. Hu, A. Li, H. Li, Y. Chen, Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2020.

[13] H. Dbouk, N. Shanbhag, Generalized depthwise-separable convolutions for adversarially robust and efficient neural networks, in: A. Beygelzimer, Y. Dauphin, P. Liang, J.W. Vaughan (Eds.), Advances in Neural Information Processing Systems, 2021.

[14] M. Yin, Y. Sui, S. Liao, B. Yuan, Towards efficient tensor decomposition-based DNN model compression with optimization framework, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2021, pp. 10674–10683.

[15] N. Li, Y. Pan, Y. Chen, Z. Ding, D. Zhao, Z. Xu, Heuristic rank selection with progressively searching tensor ring network, Complex Intell. Syst. (2021) 1–15.

[16] S. Chen, J. Zhou, W. Sun, L. Huang, Joint matrix decomposition for deep convolutional neural networks compression, Neurocomputing 516 (2023) 11–26, http://dx.doi.org/10.1016/j.neucom.2022.10.021.

[17] E.L. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 1269–1277.

[18] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, H. Li, Coordinating filters for faster deep neural networks, in: The IEEE International Conference on Computer Vision, ICCV, 2017.

[19] B. Peng, W. Tan, Z. Li, S. Zhang, D. Xie, S. Pu, Extreme network compression via filter group approximation, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 300–316.

[20] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing Systems 2, Morgan-Kaufmann, 1990, pp. 598–605.

[21] B. Hassibi, D.G. Stork, G.J. Wolff, Optimal Brain Surgeon and general network pruning, in: IEEE International Conference on Neural Networks, Vol. 1, 1993, pp. 293–299, http://dx.doi.org/10.1109/ICNN.1993.298572.

[22] S. Anwar, K. Hwang, W. Sung, Structured pruning of deep convolutional neural networks, J. Emerg. Technol. Comput. Syst. 13 (3) (2017) 32:1–32:18, http://dx.doi.org/10.1145/3005348.

[23] S. Han, J. Pool, J. Tran, W.J. Dally, Learning both weights and connections for efficient neural networks, in: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS '15, MIT Press, Cambridge, MA, USA, 2015, pp. 1135–1143.

[24] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, 2016, arXiv preprint arXiv:1611.06440.

[25] H. Hu, R. Peng, Y.-W. Tai, C.-K. Tang, Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, 2016, arXiv preprint arXiv:1607.03250.

[26] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient convnets, 2016, arXiv preprint arXiv:1608.08710.

[27] A. Aghasi, A. Abdi, N. Nguyen, J. Romberg, Net-trim: Convex pruning of deep neural networks with performance guarantee, in: Advances in Neural Information Processing Systems, 2017, pp. 3177–3186.

[28] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2736–2744.

[29] J.-H. Luo, J. Wu, An entropy-based pruning method for cnn compression, 2017, arXiv preprint arXiv:1706.05791.

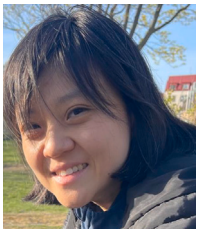
[30] J.-H. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5058–5066.

[31] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, Vol. 2017-Octob, 2017, pp. 1398–1406, http://dx.doi.org/10.1109/ICCV.2017.155, arXiv:arXiv:1707.06168v2.

[32] Q. Huang, K. Zhou, S. You, U. Neumann, Learning to prune filters in convolutional neural networks, 2018, arXiv preprint arXiv:1801.07365.

[33] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V.I. Morariu, X. Han, M. Gao, C.-Y. Lin, L.S. Davis, NISP: Pruning networks using neuron importance score propagation, in: The IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2018.

[34] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, B. Zhang, Accelerating convolutional networks via global & dynamic filter pruning, in: IJCAI, 2018, pp. 2425–2432.
[35] F. Tung, G. Mori, Clip-q: Deep network compression learning by in-parallel pruning-quantization, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7873–7882.
[36] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, Y. Wang, A systematic dnn weight pruning framework using alternating direction method of multipliers, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 184–199.
[37] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, J. Zhu, Discrimination-aware channel pruning for deep neural networks, in: Advances in Neural Information Processing Systems, 2018, pp. 875–886.
[38] Z. Huang, N. Wang, Data-driven sparse structure selection for deep neural networks, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 304–320.
[39] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4340–4349.
[40] S. Lin, R. Ji, Y. Li, C. Deng, X. Li, Toward compact convnets via structure-sparsity regularized filter pruning, IEEE Trans. Neural Netw. Learn. Syst. 31 (2) (2019) 574–588.
[41] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, D. Doermann, Towards optimal structured cnn pruning via generative adversarial learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2790–2799.
[42] X. Ding, G. Ding, Y. Guo, J. Han, Centripetal sgd for pruning very deep convolutional networks with complicated structure, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4943–4953.
[43] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, J. Sun, Metapruning: Meta learning for automatic neural network channel pruning, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 3296–3305.
[44] Z. You, K. Yan, J. Ye, M. Ma, P. Wang, Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks, 2019, arXiv:1909.08174.
[45] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, J. Kautz, Importance estimation for neural network pruning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2019.
[46] X. Ding, G. Ding, Y. Guo, J. Han, C. Yan, Approximated oracle filter pruning for destructive CNN width optimization, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 1607–1616.
[47] J.-H. Luo, J. Wu, Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference, Pattern Recognit. (2020) 107461.
[48] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, L. Shao, HRank: Filter pruning using high-rank feature map, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1529–1538.
[49] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, Y. Yang, Learning filter pruning criteria for deep convolutional neural networks acceleration, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2020.
[50] X. Ding, T. Hao, J. Liu, J. Han, Y. Guo, G. Ding, Lossless cnn channel pruning via gradient resetting and convolutional re-parameterization, 2020, arXiv preprint arXiv:2007.03260. 1.
[51] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J.-H. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, W. Zhang, Group fisher pruning for practical network compression, in: M. Meila, T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 7021–7032.
[52] J. Shi, J. Xu, K. Tasaka, Z. Chen, SASL: Saliency-adaptive sparsity learning for neural network acceleration, IEEE Trans. Circuits Syst. Video Technol. 31 (5) (2021) 2008–2019, http://dx.doi.org/10.1109/TCSVT.2020.3013170.
[53] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 770–778, http://dx.doi.org/10.1109/CVPR.2016.90.
[54] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European Conference on Computer Vision, Springer, 2016, pp. 630–645.
[55] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, Squeezenet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size, 2016, arXiv preprint arXiv:1602.07360.
[56] G. Huang, Z. Liu, L. van der Maaten, K.Q. Weinberger, Densely connected convolutional networks, 2016, arXiv:1608.06993.
[57] S. Xie, R. Girshick, P. Dollár, Z. Tu, K. He, Aggregated residual transformations for deep neural networks, in: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Vol. 2017-January, 2017, pp. 5987–5995, http://dx.doi.org/10.1109/CVPR.2017.634, arXiv:arXiv:1611.05431v2.
[58] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017, arXiv preprint arXiv:1704.04861.
[59] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.C. Chen, MobileNetV2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520, http://dx.doi.org/10.1109/CVPR.2018.00474, arXiv:arXiv:1801.04381v4.
[60] M. Tan, Q.V. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, 2019, arXiv preprint arXiv:1905.11946.
[61] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, et al., Deep high-resolution representation learning for visual recognition, IEEE Trans. Pattern Anal. Mach. Intell. (2020).
[62] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015, arXiv preprint arXiv:1510.00149.
[63] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, XNOR-Net: ImageNet classification using binary convolutional neural networks, in: B. Leibe, J. Matas, N. Sebe, M. Welling (Eds.), Computer Vision – ECCV 2016, Springer International Publishing, Cham, 2016, pp. 525–542.
[64] X. Suau, L. Zappella, N. Apostoloff, Network compression using correlation analysis of layer responses, 2018, arXiv:1807.10585.
[65] A. Boulch, Reducing parameter number in residual networks by sharing weights, Pattern Recognit. Lett. 103 (2018) 53–59, http://dx.doi.org/10.1016/j.patrec.2018.01.006.
[66] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: The IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017.
[67] A. Krizhevsky, G. Hinton, Learning Multiple Layers of Features from Tiny Images, Technical Report, Citeseer, 2009.
[68] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, 2018, arXiv preprint arXiv:1808.06866.
[69] J. Deng, W. Dong, R. Socher, L. Li, K. Li, L. Fei-Fei, ImageNet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255, http://dx.doi.org/10.1109/CVPR.2009.5206848.
[70] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710.
[71] S. Christian, I. Sergey, V. Vincent, A. Alexander, Inception-v4, inception-resnet and the impact of residual connections on learning, 2017.
[72] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
[73] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, Q. Tian, Variational convolutional neural network pruning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2780–2789.



Dr. Yinan Yu is an Assistant Professor in the Computer Science and Engineering Department at Chalmers University of Technology, Gothenburg, Sweden. Dr. Yu holds a Master's degree in Communication Engineering and a Ph.D. in Machine Learning and Signal Processing. Her current research focuses on automated machine learning for efficient and interpretable training and human-in-the-loop natural language processing, applied primarily to the industries of automotive, smart manufacturing and healthcare.

Samuel Scheidegger received his B.Sc. degree in Mechatronics and his M.Sc. degree in Systems, Control, and Mechatronics from Chalmers University of Technology in Gothenburg, Sweden, in 2013 and 2015, respectively. Since then, he has worked in the automotive industry and co-founded two AI start-up companies, including his current position as CEO of Asymptotic AI and Lumilogic. He has been actively conducting research in the fields of deep learning, computer vision, robotics, and autonomous systems. His research interests include optimizing deep learning models for computer vision tasks, with a focus on network optimization, model compression, and alternative parameterizations, as well as advancing technologies in various domains of autonomous systems to enhance their performance.

Tomas McKelvey (Senior Member, IEEE) received the M.Sc. degree in electrical engineering from Lund University, Lund, Sweden, in 1991, and the Ph.D. degree in automatic control from Linköping University, Linköping, Sweden, in 1995.,He held research and teaching positions with Linköping University, from 1995 to 1999, where he became a Docent, in 1999. From 1999 and 2000, he was a Visiting Researcher with the University of Newcastle, Newcastle, NSW,

Australia. Since 2000, he has been with the Chalmers University of Technology, Gothenburg, Sweden, where he was a Full Professor and has been the head of the Signal Processing Group, since 2006 and 2011, respectively. His research interests include model-based and statistical signal processing, system identification, and optimal control with applications to radar systems, electrical power systems, and vehicle propulsion systems.