# LinDeps: A Fine-tuning Free Post-Pruning Method to Remove Layer-Wise Linear Dependencies with Guaranteed Performance Preservation

Maxim Henry, Adrien Deliège, Anthony Cioppa, and Marc Van Droogenbroeck
Montefiore Institute, University of Liège, Liège, Belgium
{Maxim.Henry,Adrien.Deliege,Anthony.Cioppa,M.VanDroogenbroeck}@uliege.be

## Abstract

*Convolutional Neural Networks (CNN) are widely used in many computer vision tasks. Yet, their increasing size and complexity pose significant challenges for efficient deployment on resource-constrained platforms. Hence, network pruning has emerged as an effective way of reducing the size and computational requirements of neural networks by removing redundant or unimportant parameters. However, a fundamental challenge with pruning consists in optimally removing redundancies without degrading performance. Most existing pruning techniques overlook structural dependencies across feature maps within a layer, resulting in suboptimal pruning decisions. In this work, we introduce* LinDeps*, a novel post-pruning method,* i.e.*, a pruning method that can be applied on top of any pruning technique, which systematically identifies and removes redundant filters via linear dependency analysis. Particularly, LinDeps applies pivoted QR decomposition to feature maps to detect and prune linearly dependent filters. Then, a novel signal recovery mechanism adjusts the next layer's kernels to preserve compatibility and performance without requiring any fine-tuning. Our experiments on CIFAR-10 and ImageNet with VGG and ResNet backbones demonstrate that LinDeps improves compression rates of existing pruning techniques while preserving performances, leading to a new state of the art in CNN pruning. We also benchmark LinDeps in low-resource setups where no retraining can be performed, which shows significant pruning improvements and inference speedups over a state-of-the-art method. LinDeps therefore constitutes an essential add-on for any current or future pruning technique.*

## 1. Introduction

Convolutional neural networks are used in many computer vision tasks. Yet, the increasing complexity and size of state-of-the-art architectures pose significant challenges in terms of training cost, inference latency, memory footprint,

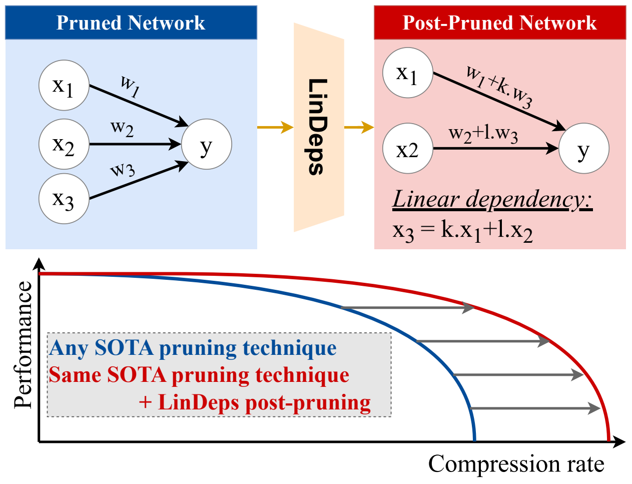

Figure 1. **Overview of our *LinDeps* post-pruning method.** We propose LinDeps, a post-pruning method applicable on top of any pruning method. LinDeps identifies and *removes linear dependencies* between input neurons $x_i$ and output neurons $y$ of a layer using a pivoted QR decomposition, and provides a signal recovery mechanism to preserve the network's performance *without fine-tuning* leading to *compression improvements with guaranteed performance preservation* of the model. Specifically, after pruning a neuron $x_i$ linearly dependent of the others, the weights $w_i$ are updated to compensate for the removal such that the same output $y$ is preserved.

and energy consumption. The need for efficient deployment of such networks on resource-constrained devices, such as mobile devices and edge computing platforms, has intensified research into network compression techniques, among which network pruning is a key solution.

Network pruning aims to reduce the size and computational requirements of neural networks by removing redundant or less important parameters while preserving as much as possible overall performance. Pruning techniques generally fall into two broad categories: unstructured pruning [4], which removes individual weights, often leading to irregular sparsity that requires specialized hardware for acceleration, and structured pruning [4, 18], which removes entire neurons, filters, or layers to yield smaller dense networks that benefit more from common GPU acceleration [15, 26]. While pruning has demonstrated effectiveness in reducing network size and inference latency, a fundamental challenge remains: how to optimally identify and remove redundancies without degrading network performance.

Traditional pruning techniques rely on heuristic criteria such as weight magnitude [15] or norm-based importance scores [26]. However, these approaches may overlook deeper structural dependencies within the network. Recent works have proposed alternative strategies, such as information bottleneck-based pruning [13] and redundancy-aware pruning techniques that leverage linear dependency analysis [34]. Nevertheless, existing techniques often fail to fully eliminate basic linear dependencies across multiple feature maps of the same network layer, leading to suboptimal pruning decisions.

In this work, we introduce *LinDeps*, a novel post-pruning method, *i.e*. a pruning applicable on top of any pruning technique. LinDeps systematically identifies and removes redundant filters by leveraging linear dependencies within feature maps, as illustrated in Figure 1. In complement to traditional low-importance pruning approaches, which focus on removing the least informative neurons, LinDeps further prunes layer-wise linear dependencies. By applying PQR decomposition on feature maps, our method identifies and removes linearly dependent filters. Importantly, we incorporate a novel signal recovery mechanism to compensate for removed parameters, thereby guaranteeing performance preservation while achieving higher compression rates, without needing to fine-tune the network. We experimentally demonstrate that LinDeps significantly enhances the compression rate of state-of-the-art pruning techniques, while preserving the performance, thus establishing a new state of the art for CNN pruning. Finally, we test our methods on different devices and measure the pruning time and inference speedup when the first compression method is applied without finetuning. This allows to evaluate the effectiveness of LinDeps in setups where a finetuning would be considered too costly, such as test-time adaptation or on very scarce resources devices.

**Contributions.** We summarize our contributions as follows: **(i)** We propose *LinDeps*, a post-pruning method based on PQR decomposition to identify and remove linear dependencies in convolutional neural networks, incorporating a novel signal recovery mechanism that compensates for the removed filters, preserving the network's accuracy without requiring any additional fine-tuning, that can be combined with any other existing state-of-the-art pruning technique. **(ii)** We demonstrate that *LinDeps* consistently improves the compression rates of existing pruning techniques while maintaining their performance, establishing new state-of-the-art pruning results. **(iii)** In addition, we apply *LinDeps* in setups with low resources, *e.g*., where retraining is considered too costly or where the model needs frequent updates, and notice significant inference speedups.

## 2. Related Work

Pruning networks to increase the speed and reduce the needs in memory size and computation resources has always been tied to research on neural networks, from the early stages [16, 25] to more recent advancements [4, 18, 28]. Many variants exist, the more relevant to our work being: unstructured vs. structured pruning, data-independent vs. data-dependent pruning, low-importance vs. similarity-based pruning, and pruning of pairwise vs. layer-wise linear relations. In the following, we position LinDeps with respect to these categories and highlight its benefits over the existing literature.

**Unstructured and structured pruning.** Network pruning can be divided into two main approaches: structured pruning that removes neurons, filters, or layers of a network, and unstructured pruning that removes individual weights. While unstructured pruning techniques [10, 14, 15] usually allow removing more weights, they cannot be fully leveraged by most pieces of hardware. Even though some recent devices allow for acceleration of specific sparsity ratio (*e.g*., Hu *et al*. use the NVIDIA Ampere architecture GPUs to accelerate networks with a 2:4 sparsity ratio [22]), this solution lacks flexibility and genericity. Conversely, structured pruning [13, 19, 20, 27, 29, 32, 34, 35] does not require using specific hardware configurations and reduces the memory size needs of the network, making it a more flexible and generic solution. LinDeps removes entire filters and feature maps, which thus falls within the structured pruning category and inherits its benefits over unstructured pruning.

**Data-independent and data-dependent methods.** Among structured pruning techniques, some are data independent [19, 20, 26, 32], meaning that they rely solely on the weights of the network for the pruning decision process, while other techniques use training data to aggregate information about the feature map distribution [21, 29, 34], about gradients [42], or retrain an auxiliary network [11] that will make the pruning decision. Recent data-dependent pruning techniques, despite a slight computational overhead, tend to provide more informed pruning decisions, as they allow a better retention of task-relevant structures. LinDeps follows this trend, as we use a batch of training data to find and prune linear dependencies between filters of the network.

**Low-importance and similarity pruning.** Some pruning techniques have been developed around an importance score used to rank channels within convolutional neural networks, such as HRank [29], and prune the network until a certain pruning objective is met. That importance score is usually used to represent how much information is contained by the associated filter so that filters carrying the least information can be pruned, yielding what is called a "low-importance pruning". A drawback of low-importance pruning techniques is that they cannot remove neurons that carry similar and significant information.

Complementary approaches, such as APIB [13] propose to use an information bottleneck principle [41], where they evaluate the mutual information between feature maps to identify filters with similar information. This technique computes a score that maximizes information used for the task while reducing similarity, yielding what is called a "similarity pruning". The downside of similarity-based pruning alone is that they cannot prune neurons that convey little to no significant information within the network.

To bridge both worlds, recent state-of-the-art techniques propose to combine low-importance and similarity-based pruning modules, such as the most complete variant of APIB [13], resulting in hybrid approaches. In a different fashion, NORTON [35], breaks down convolutional filters into small triplets of factorized components, before pruning the triplets deemed non informative enough or too similar to another according to a custom distance function. We present LinDeps as a post-pruning method, in the spirit of hybrid ones, in that we first leverage a base low-importance pruning technique, and then we apply our linear dependencies removal for maximized efficiency.

**Pruning pairwise and layer-wise linear dependencies.** APIB [13] and NORTON [35] compute mutual information or similarity in a pairwise manner, *i.e.* only between two filters at a time. This is limitative, as it cannot account for more complex linear relationships among multiple filters. By contrast, LinDeps considers each filter's relationship within a layer, allowing for a comprehensive assessment of layer-wise linear dependencies and a more relevant pruning of the filters.

Few pruning techniques have investigated structured data-dependent similarity pruning at the layer level (*i.e.* beyond pairwise correlations). Only LDFM [34] goes in that direction, by leveraging a QR decomposition of a feature-based matrix and pruning the filters, yielding features linearly independent of others. While we use the same decomposition principle in the first step of LinDeps, we additionally derive a novel recovery mechanism that allows us to prune the network while compensating for the pruned filters without needing fine-tuning, contrary to LDFM. Consequently, this allows LinDeps to be seamlessly combined, in a post-pruning fashion, with any state-of-the-art pruning technique that focuses on identifying and removing low-importance filters, pushing pruning levels to larger values without any accuracy drop.

**Note on transformer pruning.** We developed LinDeps on convolutional neural networks (CNNs), yet a separate and growing body of literature focuses on pruning transformer architectures [5, 23, 40]. Although pruned transformers are less frequently deployed on low-resource devices due to their larger memory and computation requirements, we will explore in future work whether LinDeps can be effectively extended to this distinct architectural paradigm on top of existing transformer pruning techniques.

## 3. Methodology

We first describe our pruning procedure by focusing on a single layer with C feature maps of size H×W produced by as many convolution kernels before explaining how we sequentially prune successive layers of the network. As shown in Fig. 2, we proceed in three main stages, described hereafter: (1) aggregation of feature maps into a matrix, (2) pruning via *PQR* decomposition, and (3) signal recovery to adapt subsequent layers.

**Step 1: Aggregation of feature maps.** Let us define a batch size B. Passing the entire batch through the network up to the layer $i$ of interest yields B sets of feature maps, each of shape C×H×W. We *flatten and concatenate* these feature maps across the batch and spatial dimensions to form a single matrix $\mathbf{A} \in \mathbb{R}^{C\times(B\cdot H\cdot W)}$. Specifically, the $j$-th row of $\mathbf{A}$ corresponds to the flattened feature map of channel $j$, concatenated over all B images.

**Step 2: Pruning via PQR decomposition.** Following [34], to identify linearly dependent feature maps, we perform a pivoted QR decomposition (*PQR*) [12] on the transposed aggregated matrix $\mathbf{A}^\top$. Concretely,

$$\mathbf{A}^\top\mathbf{P} \;=\; \mathbf{Q}\mathbf{R} \;=\; \mathbf{Q}\begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}, \tag{1}$$

where $\mathbf{Q}$ is orthonormal, $\mathbf{R} \in \mathbb{R}^{(B\cdot H\cdot W)\times C}$ with $\mathbf{R}_{11} \in \mathbb{R}^{C'\times C'}$ an upper-triangular matrix, $C'$ the effective rank of $\mathbf{A}$, and $\mathbf{P} \in \mathbb{R}^{C\times C}$ is a permutation matrix ensuring that the diagonal elements of $\mathbf{R}_{11}$ are sorted in decreasing order (in absolute value). The effective rank $C'$ of $\mathbf{A}$ measures how many rows (feature maps) are *linearly independent*.

We introduce a pruning threshold $\tau \in [0,1)$ that determines how aggressively we discard near-dependent channels. Conceptually, we set to zero all diagonal entries of $\mathbf{R}_{11}$ that are smaller than $\tau$ times the largest diagonal element, denoted as $(\mathbf{R}_{11})_{11}$, such that:

$$(\mathbf{R}_{11})_{kk} \leftarrow 0, \quad \forall k \text{ where } (\mathbf{R}_{11})_{kk} < \tau(\mathbf{R}_{11})_{11}. \tag{2}$$

This ensures that only the most significant independent components are retained while pruning weaker dependencies based on the threshold $\tau$. More precisely, if $\tau = 0$, we only prune channels that are *exactly* linearly dependent; if $\tau > 0$, we allow a mild approximation error to remove additional channels that exhibit near-dependence. In this case,

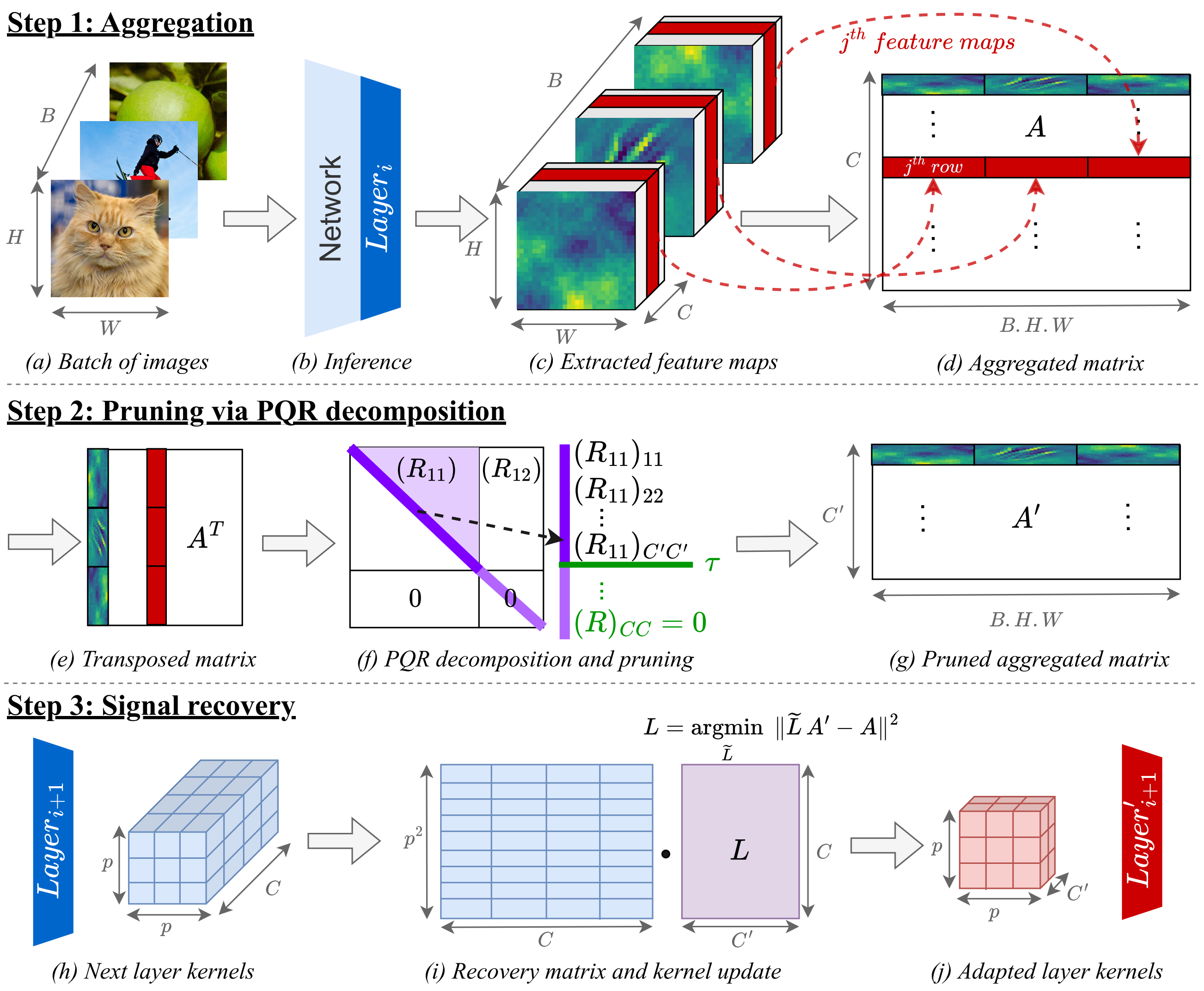

Figure 2. **Pipeline of *LinDeps* applied to a single layer.** Our method proceeds in three steps. **[Step 1] Aggregation:** (a) A batch of images of size $B$ is passed through (b) the network up to the layer to be pruned. (c) This produces $C$ feature maps per image, which are (d) flattened and concatenated into a matrix $\mathbf{A} \in \mathbb{R}^{C\times(B\cdot H\cdot W)}$, where each row corresponds to one channel, aggregated over all spatial locations and all images in the batch. **[Step 2] Pruning via PQR decomposition:** (e) The aggregated matrix is transposed, and (f) a pivoted QR decomposition (*PQR*) is applied to $\mathbf{A}^\top$, producing: $\mathbf{A}^\top \mathbf{P} = \mathbf{QR}$, where $\mathbf{P}$ permutes channels and $\mathbf{R}$ reveals linear dependencies on the diagonal of the upper-left matrix $\mathbf{R}_{11}$. A threshold $\tau$ prunes (near-)dependent channels by setting to 0 all values of $(R_{11})_{kk} < \tau(R_{11})_{11}$, resulting in (g) a pruned matrix $\mathbf{A}' \in \mathbb{R}^{C'\times(B\cdot H\cdot W)}$, keeping only $C'$ independent feature maps. **[Step 3] Signal recovery:** (h) To adapt the next layer's kernels, (i) a recovery matrix is computed by solving: $\mathbf{L}\,\mathbf{A}' \approx \mathbf{A}$, by leveraging a least-squares error approximation. Each kernel in (h) the next layer, originally of size $C \times p \times p$, is (j) updated to a new kernel of size $C' \times p \times p$ by multiplying its flattened version with $\mathbf{L}$.

we update $C'$ to be the resulting count of channels to keep. Then, we apply the inverse of $\mathbf{P}$ to map back to the original indexing of $\mathbf{A}$, thereby identifying $C'$ feature maps to *retain* and $C - C'$ feature maps to *prune*. Since each feature map is produced by a convolution kernel, we actually remove the whole kernel from the network, which incidentally prunes its corresponding feature map.

**Step 3: Signal recovery by weight adjustment.** To compensate for the information loss in the current layer of interest and incidentally enable pruning the next layer of the network independently of the current one, we develop a fine-tuning-free signal recovery mechanism, detailed hereafter. Having pruned channels at layer $i$, we must make the subsequent layer $(i + 1)$ compatible again with layer $i$. Originally, layer $(i + 1)$ expects an input of size $C \times H \times W$.

Now, we only provide $C'$ channels. To ensure that the next layer's output remains equivalent (or nearly so in the case of $\tau > 0$) to what it was before pruning, we compute a *recovery matrix* $\mathbf{L} \in \mathbb{R}^{C \times C'}$ such that

$$\mathbf{L}\,\mathbf{A}' \;\approx\; \mathbf{A}, \tag{3}$$

where $\mathbf{A}' \in \mathbb{R}^{C' \times (B \cdot H \cdot W)}$ is obtained by selecting the $C'$ retained rows of $A$, corresponding to the nonzero diagonal elements of $\mathbf{R}_{11}$ after thresholding. Mathematically, each pruned channel in $\mathbf{A}$ is (approximately if $\tau > 0$) a linear combination of the retained channels in $\mathbf{A}'$. If $\tau = 0$ and the pruned channels are strictly dependent, then $\mathbf{L}$ can be computed theoretically from the blocks of $\mathbf{R}$. Nevertheless, the exact solution requires inverting $\mathbf{R}_{11}$, which might be unstable numerically if its diagonal contains very small elements. Therefore, we rather solve for $\mathbf{L}$ in the least-squares sense of the Euclidean norm as follows:

$$\mathbf{L} \;=\; \underset{\widetilde{\mathbf{L}}}{\operatorname{argmin}}\, \|\widetilde{\mathbf{L}}\,\mathbf{A}' - \mathbf{A}\|^2, \tag{4}$$

which also allows us to handle the case of $\tau > 0$. Each element $(\mathbf{L})_{ij}$ represents the contribution of the $j$-th retained channel in $\mathbf{A}'$ to reconstruct the $i$-th original channel in $\mathbf{A}$. Finally, to accommodate the pruning performed in layer $i$, we *adapt* the convolution kernels of layer $(i+1)$ thanks to $\mathbf{L}$. Specifically, each kernel in layer $(i+1)$ is initially of shape $C \times p \times p$ (typically $p = 3$). We flatten it to a $p^2 \times C$ matrix, multiply it by $\mathbf{L}$, and reshape it into a $C' \times p \times p$ kernel. Thus, the new kernel can ingest $C'$ channels instead of $C$, becoming compatible with the pruned layer $i$, and produces equivalent outputs under the no-approximation ($\tau = 0$) scenario. Once the kernels producing feature maps of layer $(i+1)$ are updated this way, we can prune this layer *independently* using the procedure described previously, ensuring that each layer's pruning decisions do not conflict with preceding modifications.

Eventually, to prune a whole network, we iteratively prune a layer, compensate for the information loss with our novel fine-tuning-free signal recovery mechanism, then prune the next layer, and so on. Importantly, our recovery mechanism does not need the network to be retrained or fine-tuned to compensate for a performance drop due to the pruning. This is a major advantage over other pruning techniques, and it allows LinDeps to be applied efficiently after any pruning technique, providing a guaranteed extra pruning at little ($\tau > 0$) to no ($\tau = 0$) performance cost.

**Implementation details.** Hereafter, we provide a few generic details about our practical implementation:

- *Batch size $B$.* We typically use $B = 256$, following the experimental protocols presented in [13], although this can be adjusted if memory is constrained. Let us note that the equations governing LinDeps hold when $B \cdot H \cdot W > C$, which is easily satisfied in current network architectures, even with moderate batch sizes.
- *Batch Normalization (BN).* If a layer is immediately followed by a batch normalization module, we prune the corresponding BN parameters together with the channel.
- *Practical pruning threshold.* In practice, using $\tau = 0$ retains even the smallest, numerically unstable values in the diagonal of $\mathbf{R}_{11}$. Therefore, we call "lossless pruning" the results obtained with the application of LinDeps with an actual value of $\tau = 10^{-6}$ instead of $\tau = 0$ .

## 4. Experiments

We first describe the experimental settings in Sec. 4.1, including the dataset and network architectures, the evaluation scores, the base pruning techniques, and the LinDeps settings used. We then present the main quantitative results in Sec. 4.2 on various datasets and network architectures, as well as a study where LinDeps is used as a standalone pruning technique, *i.e.*, not as a post-pruning method. Finally, in Sec. 4.3, we present results in low-resource setups, first by computing the pruning gain brought by LinDeps over two methods when compute or time resources do not allow any retraining, second by computing the time needed to apply LinDeps and the inference speedup gain on various devices, from consumer laptops to edge devices.

### 4.1. Experimental settings

**Datasets and network architectures.** We assess the pruning capabilities of our post-pruning LinDeps method in combination with existing pruning techniques on two datasets: CIFAR-10 [24] and ImageNet [7]. On CIFAR-10, we compare three architectures: VGG-16 [37], ResNet-56 [17], and ResNet-110 [17]. On ImageNet, we use the ResNet-50 architecture, following common practice in pruning benchmarks [35].

**Evaluation scores.** We assess the compression induced by LinDeps in comparison to existing pruning techniques using the reduction ratio of FLOPs (floating-point operations) and the reduction ratio of the number of network parameters. We measure the amount of FLOPs and parameters using the *ptflop* [38] library. To measure the pruned network performance, we report the corresponding top-1 accuracy, in line with standard benchmarks commonly used in the pruning literature. Furthermore, it is the most neutral choice of ranking score in terms of application preferences, as demonstrated in Piérard *et al.* [36].

**Base pruning techniques.** We study the effect of our proposed post-pruning LinDeps method on top of two of the latest state-of-the-art pruning techniques: APIB [13] released in 2023, and NORTON [35] published in 2025. For completeness, we also provide the results of LinDeps without a base pruning technique. To combine the APIB pruning technique with LinDeps, we first reproduced the results of the pruned network by using the code released by the authors on GitHub[1]. After several trainings with different seeds, we selected the baseline VGG-16 as the one whose accuracy before pruning is the closest to the one reported in the original paper [13]. We kept all training and post-pruning fine-tuning settings similar to the ones reported by the authors, *i.e.*, all networks are trained using the Stochastic Gradient Descent (SGD) optimizer with a momentum of 0.9 and a weight decay of $2 \times 10^{-4}$. The initial learning rate is set to 0.1 and decayed using the cosine annealing scheduling [33]. The batch size is set to 256 and the number of epochs to 350. As APIB allows targeting any amount of pruning, we pruned several networks with pruning ratios between 65% and 80%. For each amount of pruning, we fine-tuned the network obtained as described in [13]. To avoid the seed cherry-picking reproducibility issue, we repeated the fine-tuning process eight times from the start and reported the average accuracy over the eight trials. Regarding LinDeps, the eight fine-tuned networks were post-pruned without additional fine-tuning, and the performances in terms of top-1 accuracies, FLOPs, and parameters were averaged, also to ensure reproducibility. Similarly, we combined the current state-of-the-art NORTON pruning technique [35] with LinDeps. To do so, we also first reproduced the results of the paper through the code provided on the authors' GitHub[2]. As a base, we used the checkpoints of the pruned and retrained networks available from the GitHub page, without retraining the networks ourselves. We then applied LinDeps without additional fine-tuning and reported the accuracy, FLOPs, and parameters.

## 4.2. Main quantitative results

In this section, we present the main quantitative results of our post-pruning LinDeps method in the performance-preserving setting applied on top of APIB and NORTON on various datasets and with different network architectures.

**VGG-16 on CIFAR-10.** The results of post-pruning using LinDeps in combination with APIB and NORTON on CIFAR-10 are reported in Tab. 1. For all compression levels, LinDeps increases the reduction ratios while keeping the original top-1 accuracy. Particularly, with APIB, LinDeps increases the number of FLOPs pruned by an additional 1.23%, 1.26%, 1.25%, 1.01%, 0.91%, 0.96%, 0.71%, and 0.54% for initial reduction ratios ranging from 65.78% to 79.85%. Combined with NORTON, LinDeps increases the reduction ratios by 1.39%, 0.52%, 5.7%, and 0.54% for initial reduction ratios ranging between 59.58% and 95.58%. Furthermore, we provide a complete benchmark of pruning techniques in Fig. 3. As illustrated, LinDeps combined with NORTON leads to a new state-of-the-art performance, outperforming previously published techniques.

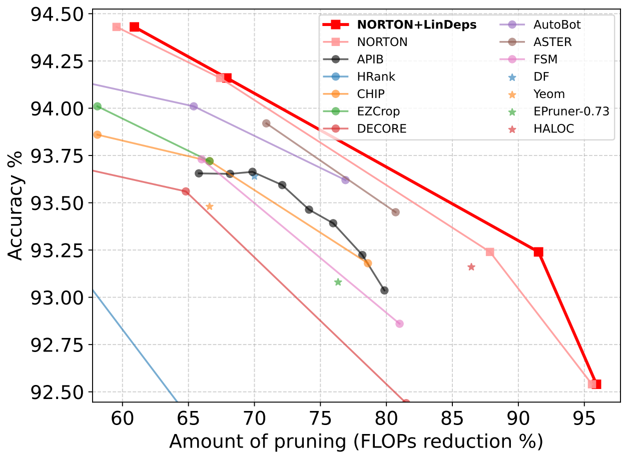

Figure 3. **Benchmark of pruning techniques with VGG-16 trained on CIFAR-10.** LinDeps, applied on top of NORTON, achieves a new state of the art, surpassing all previously published pruning techniques by a significant margin in terms of compression level while maintaining the top-1 accuracy.

**ResNet-56/110 on CIFAR-10.** Tables 2 and 3 respectively, show the results of LinDeps combined with NORTON with a ResNet-56 and ResNet-110 network architecture on CIFAR-10. In the performance-preserving regime, LinDeps similarly pushes the pruning ratio of FLOPs by an additional 1.44%, 0.16%, 2.57%, and 1.1% on ResNet-56 for reduction ratios ranging from 26.69% to 89.20% and 2.82%, 0.48%, and 0.21% on ResNet-110 for ratios ranging from 36.87% to 81.26%. This shows that LinDeps works on both small and large network architectures.

**ResNet-50 on ImageNet.** The results for LinDeps applied on top of NORTON with a ResNet-50 architecture trained on ImageNet are presented in Tab. 4. We still observe an improvement for each compression level, up to 0.17% on the pruning ratio while keeping the same top-1 accuracy. Hence, this shows that even when trained on large datasets, our post-pruning LinDeps method still allows pushing the compression level at no performance cost. In this case, the improvement is smaller than the one obtained on CIFAR-10. This could be because ResNet-50 under-performs too much on the ImageNet dataset (only around 75% accuracy), and thus the feature maps may contain a lot of noise, making it harder to find linear relationships.

**LinDeps as a standalone pruning technique.** We also evaluated LinDeps as a standalone pruning technique, applied directly to VGG-16 trained on CIFAR-10, without relying on any base pruning technique. The results, reported for different threshold values $\tau$, show that LinDeps alone achieves the following pruning ratios and corresponding top-1 accuracies: for $\tau = 0$, LinDeps reaches a prun-

[1] https://github.com/sunggo/APIB
[2] https://github.com/pvti/NORTON

| Pruning techniques | Baseline Top-1 Acc. | Pruned Top-1 Acc. | FLOPs reduc. ↑ | Params. reduc. ↑ |
|---|---|---|---|---|
| HRank [29] | 93.96% | 93.43% | 53.50% | 82.90% |
| CHIP [39] | 93.96% | 93.86% | 58.10% | 81.60% |
| EZCrop [31] | 93.96% | 94.01% | 58.10% | 81.60% |
| DECORE-500 [1] | 93.96% | 94.02% | 35.30% | 63.00% |
| AutoBot [2] | 93.96% | 94.19% | 53.70% | 49.80% |
| APIB [13] | 93.68% | 94.08% | 60.00% | 76.00% |
| NORTON [35] △ | 93.96% | 94.43% | 59.58% | 82.72% |
| NORTON+LinDeps (ours) | 93.96% | 94.43% | 60.97% | 83.02% |
| HRank [29] | 93.96% | 92.34% | 65.30% | 82.10% |
| CHIP [39] | 93.96% | 93.72% | 66.60% | 83.30% |
| EZCrop [31] | 93.96% | 93.72% | 66.60% | 83.30% |
| DECORE-200 [1] | 93.96% | 93.56% | 64.80% | 89.00% |
| AutoBot [2] | 93.96% | 94.01% | 65.40% | 57.00% |
| Yeom *et al.* [44] | 93.96% | 93.48% | 66.60% | 80.90% |
| DF [9] | 94.14% | 93.64% | 70.00% | 90.10% |
| FSM [8] | 93.96% | 93.73% | 66.00% | 86.30% |
| APIB [13] | 93.68% | 94.00% | 66.00% | 78.00% |
| APIB ∗ | 93.69% | 93.65% | 65.78% | 89.59% |
| APIB+LinDeps (ours) | 93.69% | 93.65% | 67.01% | 90.17% |
| NORTON [35] △ | 93.96% | 94.16% | 67.43% | 84.30% |
| NORTON+LinDeps (ours) | 93.96% | 94.16% | 67.95% | 84.38% |
| CHIP [39] | 93.96% | 93.18% | 78.60% | 87.30% |
| DECORE-100 [1] | 93.96% | 92.44% | 81.50% | 96.60% |
| FSM [8] | 93.96% | 92.86% | 81.00% | 90.60% |
| EPruner-0.73 [30] | 93.02% | 93.08% | 76.34% | 88.80% |
| HALOC [43] | 92.78% | 93.16% | 86.44% | 98.56% |
| ASTER [45] | 93.90% | 93.92% | 70.90% | N/A |
| ASTER [45] | 92.90% | 93.45% | 80.70% | N/A |
| AutoBot [2] | 93.96% | 93.62% | 76.90% | 63.24% |
| APIB ∗ | 93.69% | 93.65% | 68.15% | 91.00% |
| APIB+LinDeps (ours) | 93.69% | 93.65% | 69.41% | 91.49% |
| APIB ∗ | 93.69% | 93.66% | 69.85% | 92.00% |
| APIB+LinDeps (ours) | 93.69% | 93.66% | 71.10% | 92.41% |
| APIB ∗ | 93.69% | 93.59% | 72.10% | 92.58% |
| APIB+LinDeps (ours) | 93.69% | 93.59% | 73.11% | 92.99% |
| APIB ∗ | 93.69% | 93.46% | 74.13% | 93.39% |
| APIB+LinDeps (ours) | 93.69% | 93.46% | 75.04% | 93.72% |
| APIB ∗ | 93.69% | 93.39% | 75.96% | 94.29% |
| APIB+LinDeps (ours) | 93.69% | 93.39% | 76.92% | 94.61% |
| APIB ∗ | 93.69% | 93.22% | 78.17% | 95.14% |
| APIB+LinDeps (ours) | 93.69% | 93.22% | 78.88% | 95.42% |
| APIB ∗ | 93.69% | 93.03% | 79.85% | 95.80% |
| APIB+LinDeps (ours) | 93.69% | 93.03% | 80.39% | 96.07% |
| NORTON [35] △ | 93.96% | 93.24% | 87.86% | 87.04% |
| NORTON+LinDeps (ours) | 93.96% | 93.24% | 93.56% | 96.28% |
| HRank [29] | 93.96% | 91.23% | 76.50% | 92.00% |
| RGP [3] | 93.14% | 91.45% | 90.49% | 90.62% |
| DECORE-50 [1] | 93.96% | 91.68% | 88.30% | 98.30% |
| NORTON [35] △ | 93.96% | 92.54% | 95.58% | 98.33% |
| NORTON+LinDeps (ours) | 93.96% | 92.54% | 96.12% | 98.66% |

Table 1. **Performance comparison of pruning techniques for VGG-16 on CIFAR-10** roughly grouped by FLOPs reduction as in [35]. The following symbols apply: no symbol means that the results are directly imported from the original paper, $*$ refers to results reproduced using the code from the official GitHub, and △ means that the results are obtained from the available code and checkpoints from the official GitHub. Applying LinDeps consistently increases FLOPs reduction and parameters reduction.

ing ratio of 24.9% with no accuracy loss (93.68% top-1 accuracy), demonstrating that nearly 25% of the parameters can be safely removed at no performance cost by eliminating strictly linearly dependent filters. However, as the pruning threshold $\tau$ increases between 0.05 and 0.5, the pruning ratios increase to 34.9%, 40.8%, 43.2%, and 64.2%, but at the cost of significant performance degradation, with top-1 accuracies dropping to 92.65%, 91.08%,

| Pruning techniques | Baseline Top-1 Acc. | Pruned Top-1 Acc. | FLOPs reduc. ↑ | Params. reduc. ↑ |
|---|---|---|---|---|
| HRank [29] | 93.26% | 93.52% | 29.30% | 16.80% |
| DECORE-450 [1] | 93.26% | 93.34% | 26.30% | 24.20% |
| NORTON [35] △ | 93.26% | 94.48% | 25.25% | 30.77% |
| NORTON+LinDeps (ours) | 93.26% | 94.48% | 26.69% | 31.06% |
| HRank [29] | 93.26% | 93.17% | 50.00% | 42.4% |
| DECORE-200 [1] | 93.26% | 93.26% | 49.90% | 49.00% |
| FSM [8] | 93.26% | 93.63% | 51.20% | 43.60% |
| AutoBot [2] | 93.27% | 93.76% | 55.9% | 45.9% |
| EZCrop [31] | 93.26% | 93.80% | 47.40% | 42.80% |
| NORTON [35] △ | 93.26% | 93.99% | 41.33% | 47.31% |
| NORTON+LinDeps (ours) | 93.26% | 93.99% | 41.49% | 47.34% |
| CHIP [39] | 93.26% | 92.05% | 71.80% | 72.30% |
| APIB [13] | 93.26% | 93.29% | 67.00% | 66.00% |
| NORTON [35] △ | 93.26% | 93.81% | 69.67% | 74.39% |
| NORTON+LinDeps (ours) | 93.26% | 93.81% | 72.24% | 76.72% |
| DECORE-55 [1] | 93.26% | 90.85% | 81.50% | 85.30% |
| APIB [13] | 93.26% | 91.53% | 81.00% | 83.00% |
| NORTON [35] △ | 93.26% | 91.61% | 88.10% | 89.69% |
| NORTON+LinDeps (ours) | 93.26% | 91.61% | 89.20% | 91.23% |

Table 2. **Pruning results with ResNet-56 trained on CIFAR-10**, comparing NORTON with and without LinDeps. In the performance-preserving regime, LinDeps consistently increases the pruning ratio of FLOPs across all compression levels, with gains ranging from 0.16% to 2.57%. The symbol conventions from Tab. 1 apply.

| Pruning techniques | Baseline Top-1 Acc. | Pruned Top-1 Acc. | FLOPs reduc. ↑ | Params. reduc. ↑ |
|---|---|---|---|---|
| DECORE-500 [1] | 93.50% | 93.88% | 35.40% | 35.70% |
| NORTON [35] △ | 93.50% | 94.85% | 34.05% | 37.16% |
| NORTON+LinDeps (ours) | 93.50% | 94.85% | 36.87% | 37.72% |
| DECORE-300 [1] | 93.50% | 93.50% | 61.80% | 64.80% |
| NORTON [35] △ | 93.50% | 94.05% | 63.00% | 65.13% |
| NORTON+LinDeps (ours) | 93.50% | 94.05% | 63.48% | 65.22% |
| DECORE-175 [1] | 93.50% | 92.71% | 76.90% | 79.60% |
| NORTON [35] △ | 93.50% | 92.77% | 81.05% | 82.41% |
| NORTON+LinDeps (ours) | 93.50% | 92.77% | 81.26% | 82.48% |

Table 3. **Pruning results with ResNet-110 trained on CIFAR-10**, comparing NORTON with and without LinDeps. Similarly to ResNet-56, LinDeps enhances the pruning ratio of FLOPs across all compression levels, with improvements ranging from 0.21% to 2.82%. The symbol conventions from Tab. 1 apply.

| Pruning techniques | Baseline Top-1 Acc. | Pruned Top-1 Acc. | FLOPs reduc. ↑ | Params. reduc. ↑ |
|---|---|---|---|---|
| NORTON [35] △ | 76.15% | 76.90% | 42.91% | 43.06% |
| NORTON+LinDeps (ours) | 76.15% | 76.90% | 43.08% | 43.10% |
| NORTON [35] △ | 76.15% | 76.59% | 48.82% | 47.03% |
| NORTON+LinDeps (ours) | 76.15% | 76.59% | 48.85% | 47.05% |
| NORTON [35] △ | 76.15% | 75.95% | 63.16% | 58.72% |
| NORTON+LinDeps (ours) | 76.15% | 75.95% | 63.24% | 58.73% |
| NORTON [35] △ | 76.15% | 74.00% | 75.80% | 68.78% |
| NORTON+LinDeps (ours) | 76.15% | 74.00% | 75.85% | 68.79% |
| NORTON [35] △ | 76.15% | 73.65% | 77.22% | 76.91% |
| NORTON+LinDeps (ours) | 76.15% | 73.65% | 77.33% | 76.93% |

Table 4. **Pruning results with ResNet-50 trained on ImageNet**, comparing NORTON with and without LinDeps. Similarly, LinDeps further improves the pruning ratio by up to 0.17% at identical top-1 accuracy. The symbol conventions from Table 1 apply.

78.60%, and 23.2%, respectively. This confirms that LinDeps alone does not achieve competitive performance compared to modern pruning techniques (see Tab. 1), as it was designed to complement existing pruning techniques rather than replace them. The lack of re-fine-tuning after pruning further exacerbates this drop, as no training step compensates for the removed filters. Additionally, the purely linear dependency-based approach of LinDeps only captures correlations between highly similar feature maps, but fails to target low-importance neurons that could be pruned independently regarding other neurons, which are typically removed by conventional importance-based pruning techniques. This highlights why LinDeps performs best when combined with other pruning techniques, leveraging complementary pruning criteria to maximize both compression and performance preservation.

## 4.3. LinDeps in a low-resource setup

In this section, we consider the practical case of low-resource environments where a model can be pruned but the resources to retrain it are not available. This can be the case when the retraining process is long and/or computationally expensive, or when regular and fast updates need to be made on the model, *e.g.*, in the case of test-time adaptation [6].

In that regard, Fig. 4 shows the pruning gain obtained by applying LinDeps on top of a VGG model trained on CIFAR-10 then compressed with NORTON (using their low-rank method with a rank of 6), but not retrained after pruning. We used $\tau = 0.1$ which empirically gave the best trade-off between pruning gain and accuracy loss when pruning without retraining. We achieve gains of up to 30% of pruning with losses in accuracy lower than 0.3%, highlighting the benefits of LinDeps in this setup.

Furthermore, Tab. 5 compares the time needed to perform NORTON and LinDeps on several devices, from consumer laptops to resource-limited systems, as well as the gain in inference time for a batch of 256 images before and after applying LinDeps (after the NORTON pruning). We can observe that, comparatively, running LinDeps takes significantly less time than running NORTON, and that LinDeps provides a consistent and valuable speedup in inference, which is critical in real-time applications[3].

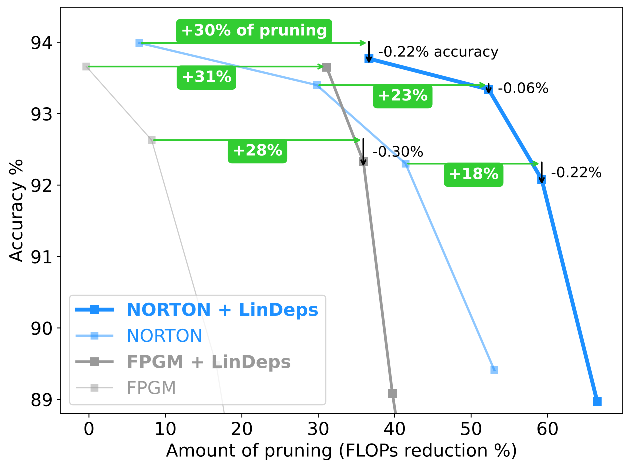

Figure 4. **LinDeps pruning improvement without retraining.** In low-resource environments where post-pruning retraining cannot be afforded, LinDeps gives a significant pruning boost for a comparatively negligible decrease in performance.

| Device | NORTON time (s) | LinDeps time (s) | Inference latency |
|---|---|---|---|
| Asus GPU RTX 3070 | 607 | 13 | -23% |
| Asus CPU Intel Core i7 | 460 | 33 | -21% |
| MacBook M1 MPS | 8,872 | 30 | -29% |
| MacBook M1 CPU | 288 | 111 | -33% |
| Orin AGX GPU | 3,718 | 17 | -25% |
| Orin AGX CPU | 1,903 | 403 | -23% |
| Raspberry Pi CPU | 1,738 | 247 | -28% |

Table 5. **Time benchmark of LinDeps.** Comparison of NORTON and LinDeps runtimes across consumer laptops (Asus laptop with a NVIDIA GeForce RTX 3070, MacBook Pro M1) and low-resource devices (Orin AGX, Raspberry Pi 5), and inference latency gain by applying LinDeps on batches of 256 images.

## 5. Conclusion

In this paper, we introduced *LinDeps*, a novel post-pruning method designed to systematically identify and eliminate redundant filters in convolutional networks through the analysis of linear dependencies within feature maps. By leveraging pivoted QR decomposition, LinDeps effectively detects dependent filters and removes them, while a novel signal recovery mechanism adjusts the subsequent layer to preserve the network's compatibility and performance. A key advantage of LinDeps is its flexibility: it can be applied on top of any existing pruning technique, boosting its compression rate with either zero or controlled accuracy degradation, depending on the desired compression rate. This versatility makes it a valuable complement to state-of-the-art pruning techniques across various architectures and datasets. Our experiments demonstrated that LinDeps consistently improves the compression rates of multiple pruning baselines on CIFAR-10 and ImageNet, achieving new state-of-the-art results. Furthermore, we highlighted that LinDeps performs well on both small and large networks. Overall, LinDeps is an efficient and theoretically grounded approach to post-pruning, contributing to the broader goal of enabling lightweight yet high-performing deep learning networks for low-resource devices.

[3]We also note that some operations in NORTON were not supported by the MPS chip of the MacBook, defaulting to the CPU whenever needed, which might explain the large corresponding time in Tab. 5

**Acknowledgments.** The research of M. Henry was funded by the PIT MecaTech under grant No. C8650 (ReconnAIssance). The research of A. Deliège was funded by the F.R.S.-FNRS under project grant T.0065.22 and by the SPW EER, Wallonia, Belgium, under grant n°2010235 (ARIAC by DIGITALWALLONIA4.AI).

## References

[1] Manoj Alwani, Yang Wang, and Vashisht Madhavan. DECORE: Deep compression with reinforcement learning. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 12339–12349, New Orleans, LA, USA, 2022. 7

[2] Thibault Castells and Seul-Ki Yeom. Automatic neural network pruning that efficiently preserves the model accuracy. *arXiv*, abs/2111.09635, 2021. 7

[3] Zhuangzhi Chen, Jingyang Xiang, Yao Lu, Qi Xuan, Zhen Wang, Guanrong Chen, and Xiaoniu Yang. RGP: Neural network pruning through regular graph with edges swapping. *IEEE Trans. Neural Networks Learn. Syst.*, 35(10):14671–14683, 2024. 7

[4] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(12):10558–10578, 2024. 1, 2

[5] Krishna Teja Chitty-Venkata, Sparsh Mittal, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. A survey of techniques for optimizing transformer inference. *J. Syst. Arch.*, 144, 2023. 3

[6] Anthony Cioppa, Adrien Deliege, Maxime Istasse, Christophe De Vleeschouwer, and Marc Van Droogenbroeck. ARTHuS: Adaptive real-time human segmentation in sports through online distillation. In *IEEE Int. Conf. Comput. Vis. Pattern Recognit. Work. (CVPRW), CVsports*, pages 2505–2514, Long Beach, CA, USA, 2019. 8

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 248–255, Miami, FL, USA, 2009. 5

[8] Yuanzhi Duan, Yue Zhou, Peng He, Qiang Liu, Shukai Duan, and Xiaofang Hu. Network pruning via feature shift minimization. In *Asian Conf. Comput. Vis. (ACCV)*, pages 618–634. 2023. 7

[9] Moonjung Eo, Suhyun Kang, and Wonjong Rhee. A differentiable framework for end-to-end learning of hybrid structured compression. *arXiv*, abs/2309.13077, 2023. 7

[10] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Int. Conf. Learn. Represent. (ICLR)*, pages 1–42, New Orleans, LA, USA, 2019. 2

[11] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Jointly training and pruning CNNs via learnable agent guidance and alignment. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 16058–16069, Seattle, WA, USA, 2024. 2

[12] Gene H. Golub. Numerical methods for solving linear least squares problems. *Numer. Math.*, 7(3):206–216, 1965. 3

[13] Song Guo, Lei Zhang, Xiawu Zheng, Yan Wang, Yuchao Li, Fei Chao, Chenglin Wu, Shengchuan Zhang, and Rongrong Ji. Automatic network pruning via Hilbert-Schmidt independence criterion lasso under information bottleneck principle. In *IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, pages 17412–17423, Paris, Fr., 2023. 2, 3, 5, 6, 7

[14] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 1387–1395, Barcelona, Spain, 2016. 2

[15] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 1135–1143, Montréal, Can., 2015. 2

[16] Stephen José Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 177–185, Denver, CO, USA, 1988. 2

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 770–778, Las Vegas, NV, USA, 2016. 5

[18] Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(5):2900–2919, 2024. 2

[19] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Int. Jt. Conf. Artif. Intell. (IJCAI)*, pages 2234–2240, Stockholm, Sweden, 2018. 2

[20] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 4335–4344, Long Beach, CA, USA, 2019. 2

[21] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv*, abs/1607.03250, 2016. 2

[22] Yuezhou Hu, Kang Zhao, Weiyu Huang, Jianfei Chen, and Jun Zhu. Accelerating transformer pre-training with 2:4 sparsity. In *Int. Conf. Mach. Learn. (ICML)*, pages 19531–19543, 2024. 2

[23] Beom Jin Kang, Hae In Lee, Seok Kyu Yoon, Young Chan Kim, Sang Beom Jeong, Seong Jun O, and Hyun Kim. A survey of FPGA and ASIC designs for transformer inference acceleration and optimization. *J. Syst. Arch.*, 155, 2024. 3

[24] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009. Technical report, University of Toronto. 5

[25] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 598–605, Denver, CO, USA, 1989. 2

[26] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *Int. Conf. Learn. Represent. (ICLR)*, pages 1–13, 2017. 2

[27] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In

*Int. Conf. Learn. Represent. (ICLR)*, pages 1–13, Toulon, France, 2017. 2

[28] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021. 2

[29] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. HRank: Filter pruning using high-rank feature map. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 1526–1535, Seattle, WA, USA, 2020. 2, 7

[30] Mingbao Lin, Rongrong Ji, Shaojie Li, Yan Wang, Yongjian Wu, Feiyue Huang, and Qixiang Ye. Network pruning using adaptive exemplar filters. *IEEE Trans. Neural Networks Learn. Syst.*, 33(12):7357–7366, 2022. 7

[31] Rui Lin, Jie Ran, Dongpeng Wang, King Hung Chiu, and Ngai Wong. EZCrop: Energy-zoned channels for robust output pruning. In *IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, pages 3595–3604, Waikoloa, HI, USA, 2022. 7

[32] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *IEEE Int. Conf. Comput. Vis. (ICCV)*, pages 2755–2763, Venice, Italy, 2017. 2

[33] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Int. Conf. Learn. Represent. (ICLR)*, pages 1–16, Toulon, France, 2017. 6

[34] Hao Pan, Zhongdi Chao, Jiang Qian, Bojin Zhuang, Shaojun Wang, and Jing Xiao. Network pruning using linear dependency analysis on feature maps. In *IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, pages 1720–1724, Toronto, Ontario, Can., 2021. 2, 3

[35] Van Tien Pham, Yassine Zniyed, and Thanh Phuong Nguyen. Enhanced network compression through tensor decompositions and pruning. *IEEE Trans. Neural Networks Learn. Syst.*, 36(3):4358–4370, 2025. 2, 3, 5, 6, 7

[36] Sébastien Piérard, Anaïs Halin, Anthony Cioppa, Adrien Deliège, and Marc Van Droogenbroeck. Foundations of the theory of performance-based ranking. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 14293–14302, Nashville, TN, USA, 2025. 5

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, abs/1409.1556, 2014. 5

[38] Vladislav Sovrasov. ptflops: a flops counting tool for neural networks in pytorch framework. `https://github.com/sovrasov/flops-counter.pytorch`, 2024. 5

[39] Yang Sui, Miao Yin, Yi Xie, Huy Phan, Saman Zonouz, and Bo Yuan. CHIP: channel independence-based pruning for compact neural networks. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 24604–24616, 2021. 7

[40] Yehui Tang, Yunhe Wang, Jianyuan Guo, Zhijun Tu, Kai Han, Hailin Hu, and Dacheng Tao. A survey on transformer compression. *arXiv*, abs/2402.05964, 2024. 3

[41] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Inf. Theory Work. (ITW)*, pages 1–5, Jerusalem, Israel, 2015. 3

[42] Joost van Amersfoort, Milad Alizadeh, Sebastian Farquhar, Nicholas Lane, and Yarin Gal. Single shot structured pruning before training. *arXiv*, abs/2007.00389, 2020. 2

[43] Jinqi Xiao, Chengming Zhang, Yu Gong, Miao Yin, Yang Sui, Lizhi Xiang, Dingwen Tao, and Bo Yuan. HALOC: Hardware-aware automatic low-rank compression for compact neural networks. In *AAAI Conf. Artif. Intell.*, pages 10464–10472, Washington DC, USA, 2023. 7

[44] Seul-Ki Yeom, Kyung-Hwan Shim, and Jee-Hyun Hwang. Toward compact deep neural networks via energy-aware pruning. *arXiv*, abs/2103.10858, 2021. 7

[45] Yuyao Zhang and Nikolaos M. Freris. Adaptive filter pruning via sensitivity feedback. *IEEE Trans. Neural Networks Learn. Syst.*, 35(8):10996–11008, 2024. 7