SVD-NAS: Coupling Low-Rank Approximation and Neural Architecture Search

Zhewen Yu, Christos-Savvas Bouganis
Imperial College London
London, UK
{zhewen.yu18, christos-savvas.bouganis}@imperial.ac.uk

Abstract

The task of compressing pre-trained Deep Neural Networks has attracted wide interest of the research community due to its great benefits in freeing practitioners from data access requirements. In this domain, low-rank approximation is a promising method, but existing solutions considered a restricted number of design choices and failed to efficiently explore the design space, which lead to severe accuracy degradation and limited compression ratio achieved. To address the above limitations, this work proposes the SVD-NAS framework that couples the domains of low-rank approximation and neural architecture search. SVD-NAS generalises and expands the design choices of previous works by introducing the Low-Rank architecture space, LR-space, which is a more fine-grained design space of low-rank approximation. Afterwards, this work proposes a gradient-descent-based search for efficiently traversing the LR-space. This finer and more thorough exploration of the possible design choices results in improved accuracy as well as reduction in parameters, FLOPS, and latency of a CNN model. Results demonstrate that the SVD-NAS achieves 2.06-12.85pp higher accuracy on ImageNet than state-of-the-art methods under the data-limited problem setting. SVD-NAS is open-sourced at `https://github.com/Yu-Zhewen/SVD-NAS`.

1. Introduction

Deep Neural Networks (DNNs) have attracted the interest of practitioners and researchers due to their impressive performance on a number of tasks, pushing the state-of-the-art beyond of what was thought to be achievable through classical Machine Learning methods. However, the high computational and memory storage cost of DNN models impede their deployment on resource-constrained edge devices. In order to produce lightweight models, the following techniques are often considered:

- compression of a pre-trained model followed by optional fine-tuning [14, 15].
- compression-aware training, where the computational costs are integrated into the training objective as a regulariser [8].
- design and train a lightweight model by construction using domain knowledge or Automated Machine Learning (AutoML) [23, 25].

However, in the real-world scenario, the access to the original training dataset might not be easily granted, especially when the training dataset is of value or contains sensitive information. In this situation, compressing a pre-trained model has attracted wide interest of the research community, as the task of compression has the minimal requirement of data access.

Among the model compression methods, pruning [12] and quantisation [1] have been well researched and deliver good results. However, the low-rank approximation approaches still remain a challenge on their application due to the severe accuracy degradation and limited compression ratio achieved [18]. The value of low-rank approximation originates from their potential impact on computational savings as well as their ability to result in structured computations, key element of today's computing devices.

This work considers the low-rank approximation problem of a Convolutional Neural Network (CNN). Let's consider a CNN that contains L convolutional layers. Let's denote the weight tensor of the i^{th} convolutional layer by $\boldsymbol{W_i}$, where $i \in [0, L-1]$, and having dimensions (f_i, c_i, k_i, k_i), denoting f_i filters, c_i input channels and $k_i \times k_i$ kernel size. The low-rank approximation problem can be expressed as finding a set of low-rank tensors $\hat{\mathbb{W}}_i = \{\boldsymbol{\hat{W}_i^0}, \boldsymbol{\hat{W}_i^1}, ...\}$, and a function $F(\hat{\mathbb{W}}_i)$ that approximate $\boldsymbol{W_i}$, in some metric space. Therefore, the low-rank approximation problem has two parts; to identify the decomposition scheme, i.e. the function F, and the rank kept to construct the low-rank tensors, i.e. $r_i = \{r_i^0, r_i^1, ...\}$, such that the metrics of interest are optimised.

The above problem defines a large design space to be explored but existing approaches restrict themselves to only

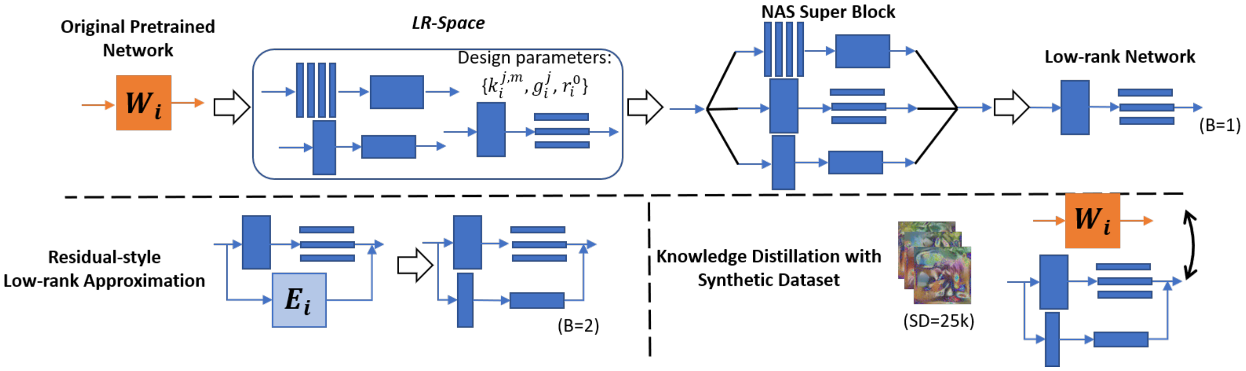
Original Pretrained Network
W_i
LR-Space
Design parameters:
$\{k_i^{j,m}, g_i^j, r_i^0\}$
NAS Super Block
Low-rank Network
(B=1)
Residual-style Low-rank Approximation
E_i
(B=2)
Knowledge Distillation with Synthetic Dataset
(SD=25k)
W_i

Figure 1: Main contributions of SVD-NAS. **Upper:** Given a pre-trained model, construct *LR-space* and utilise NAS to identify optimal approximations. **Left Lower:** Extend *LR-space* with the residual-style building block. **Right Lower:** Create a synthetic dataset and fine-tune the low-rank model by knowledge distillation.

consider a small fraction of this space, by forcing the weight tensors in a network to adopt the same or similar decomposition schemes across all the layers in a network [8, 17]. Moreover, even within their small sub-space, their design space exploration was slow and sub-optimum, either requires extensive hand-craft effort [34] or is based on heuristics that employ the Mean Squared Error (MSE) of weight tensors approximations as a proxy of the network's overall accuracy degradation [35, 32].

In this work, we offer a new perspective in applying low-rank approximation, by converting it to a Neural Architecture Search (NAS) problem. The key novel aspects of this work are:

Firstly, we describe the process of low-rank approximation as a per-layer substitution of the original pre-trained network. For every layer, we introduce a Low-Rank architecture design space, *LR-space*, which is defined by a set of parameterisable building blocks. We demonstrate searching the design parameters of these building blocks is equivalent to exploring different decomposition schemes and ranks. Afterwards, we utilise the gradient-descent NAS to navigate the *LR-space*, and jointly optimise the accuracy and the computational requirements (e.g. FLOPs) of the compressed model.

Secondly, a residual-style low-rank approximation is proposed to further refine the accuracy-FLOPs trade-off, based on a divide-and-conquer approach. We convert every convolutional layer in the original pre-trained network into the residual-style computational structure containing multiple branches, where each branch can have a distinct decomposition scheme and rank. Such a residual structure expands the design space, leading to a more fine-grained but still structured low-rank approximation solution.

Finally, motivated by previous work in model quantisation [5], where the authors generated synthetic dataset to deal with the data-limited problem setting, we applied a similar approach for low-rank approximation. The synthetic dataset is fed into both the original pre-trained model and the compressed low-rank model, enabling the tuning of the weights of the compressed model through knowledge distillation, improving further the framework's performance without accessing the actual training data.

A comparison of the proposed framework to the state-of-the-art approach [18] demonstrates that our framework is able to achieve 2.06-12.85pp higher accuracy on ResNet-18, MobileNetV2 and EfficientNet-B0 while requiring similar or even lower FLOPs under the data-limited problem setting.

2. Related Work

2.1. Low-rank Approximation

Previous work on low-rank approximation of CNNs can be classified broadly into two categories depending on the underlying methodology applied; Singular Value Decomposition (SVD) and Higher-Order Decomposition (HOD). In the case of SVD, the authors of [34, 28, 21] approximated $\boldsymbol{W_i}$ with two low-rank tensors where the latter tensor corresponds to a point-wise convolution. Their approaches differ in whether the first low-rank tensor corresponds to a grouped convolution and on the number of groups that it employs. Instead of having a point-wise convolution, Tai *et al.* [24] implemented the low-rank tensors with two spatial-separable convolutions. Our framework uses the SVD algorithm for decomposing the weight matrix mainly because of its low complexity compared to HOD methods, such as Tucker [11] and CP [13], that use the more expensive Alternate Least Squares algorithm.

2.2. Neural Architecture Search

NAS considers the neural network design process through the automatic search of the optimal combination of high-performance building blocks. The search can be performed using a top-down approach [3], where a super network is initially trained and then pruned, or bottom-up [19] approach, where optimal building blocks are firstly identified and put together to form the larger network. Popular searching algorithms include reinforcement learning [9], evolutionary [3] and gradient descent [29]. In this work, we adopt a gradient-descent search through a top-down approach to solve the low-rank approximation problem, and unlike the common problem setting of NAS which assumes the availability of large amount of training data, we focus on the data-limited scenario instead.

3. Design Space

The objective of the proposed approach is to approximate each convolutional layer in a given CNN through a low-rank approximation such as the computation cost of the CNN is minimised, while a minimum penalty in the accuracy of the network is observed. Towards this, a design space, *LR-space*, is firstly defined in this section and a searching methodology to traverse that space will be introduced in section 4.

3.1. Low-rank Architecture Space (*LR-space*)

In the rest of the paper, the convolutional layers in the pre-trained model are referred to as *original layers*. Each *original layer* is substituted with a parameterisable building block, as Fig. 2 Left shows. The building block has the same input and output feature map dimensions as the *original layer*, but it contains two consecutive convolutional layers which are referred to as *low-rank layers*.

The proposed building block is characterised through three design parameters: the low-rank kernel size $k_i^{j,m}$, the low-rank group number g_i^j and the rank r_i^0, where $j \in \{0, 1\}$, denoting the first and the second *low-rank layer* respectively, and $m \in \{0, 1\}$, denoting two spatial dimensions. In order to derive the weights of *low-rank layers* from the *original layer* with SVD-based decomposition, which will be elaborated in section 3.2, additional constraints on the design parameters are introduced as follows:

- $\prod\limits_j k_i^{j,m} = k_i$, which equivalently forces the kernel size of the *low-rank layers* to be one of $\{1{\times}1, k_i{\times}k_i, k_i{\times}1, 1 \times k_i\}$, since k_i is a prime number for most CNNs.

- $\underset{j}{min}(g_i^j) = 1$. This ensures that two *low-rank layers* cannot be grouped convolutions at the same time.

- $r_i^0 \underset{j}{max}(g_i^j)(\frac{c_i}{g_i^0}\prod\limits_m k_i^{0,m} + \frac{f_i}{g_i^1}\prod\limits_m k_i^{1,m}) < c_i f_i k_i k_i$, the total number of weights inside two *low-rank layers* should be less than the *original layer*.

The proposed *LR-space* generalises previous works which only took a subset of the space into account. Specifically, [34, 28, 6] considered the corner case of low-rank group number that $g_i^0 \in \{1, f_i, c_i\}$. Although [21, 17] introduced a design parameter to control group number, they did not explore different possibilities of kernel sizes as well as not attempt to put the grouped convolution in the second layer.

3.2. Data-free Weight Derivation

In this section, we demonstrate how to use SVD to derive the weights of *low-rank layers* from the *original layers* in a data-free manner. Same as before, we denote the weight tensor of the *original layer* by $\boldsymbol{W_i}$, while the weight tensors of the corresponding two *low-rank layers* are denoted by $\boldsymbol{\hat{W}_i^0}$ and $\boldsymbol{\hat{W}_i^1}$ respectively.

$\boldsymbol{W_i}$, as a 4-d tensor, has the dimensions of (f_i, c_i, k_i, k_i). As (1) shows, if we slice and split $\boldsymbol{W_i}$ on its first and second dimension into g_i^1 and g_i^0 groups respectively, we will obtain $g_i^0 g_i^1$ tensors in total, where the dimensions of each tensor are $(\frac{f_i}{g_i^1}, \frac{c_i}{g_i^0}, k_i, k_i)$.

$$\boldsymbol{W_{i,q,p}} = \boldsymbol{W_i}[q\frac{f_i}{g_i^1} : (q+1)\frac{f_i}{g_i^1},\ p\frac{c_i}{g_i^0} : (p+1)\frac{c_i}{g_i^0},\ :,\ :],$$
$$p \in [0, g_i^0 - 1],\ q \in [0, g_i^1 - 1] \quad (1)$$

Due to the previous constraint on design parameters, we can substitute k_i with the low-rank kernel size $k_i^{j,m}$. Therefore, the dimensions of each sliced tensor can also be expressed as $(\frac{f_i}{g_i^1}, \frac{c_i}{g_i^0}, k_i^{0,0}k_i^{1,0}, k_i^{0,1}k_i^{1,1})$. If we now reshape each sliced tensor $\boldsymbol{W_{i,q,p}}$ from 4-d to 2-d, we obtain the tensors $\mathcal{W}_{i,q,p}$, each having the dimensions of $(\frac{f_i}{g_i^1}k_i^{1,0}k_i^{1,1}, \frac{c_i}{g_i^0}k_i^{0,0}k_i^{0,1})$.

Applying SVD to $\mathcal{W}_{i,q,p}$ and keeping only the top-r_i^0 singular values, we obtain the following approximation,

$$\mathcal{W}_{i,q,p} = USV \approx U_{r_i^0} S_{r_i^0} V_{r_i^0} = \hat{\mathcal{W}}_{i,q,p}^1 \hat{\mathcal{W}}_{i,q,p}^0 \quad (2)$$

where $\hat{\mathcal{W}}_{i,q,p}^0$ and $\hat{\mathcal{W}}_{i,q,p}^1$ are 2-d low-rank tensors after absorbing the truncated diagonal matrix $S_{r_i^0}$ into $V_{r_i^0}$ and $U_{r_i^0}$.

The obtained 2-d low-rank tensors are reshaped back into the 4-d weight tensors, and they are concatenated together on their first and second dimension, which reverts the slice operation in (1). Eventually, two 4-d low-rank weight tensors are generated, denoted by $\boldsymbol{\hat{W}_i^0}$ and $\boldsymbol{\hat{W}_i^1}$, and have the dimensions of $(r_i^0, \frac{c_i}{g_i^0}, k_i^{0,0}, k_i^{0,1})$ and $(\frac{f_i}{g_i^1}, r_i^0, k_i^{1,0}, k_i^{1,1})$ respectively.

Recall that the SVD-based low-rank approximation problem is to identify optimal $F(\boldsymbol{\hat{W}_i^1}, \boldsymbol{\hat{W}_i^0})$ that approximates $\boldsymbol{W_i}$, involving choosing both the decomposition

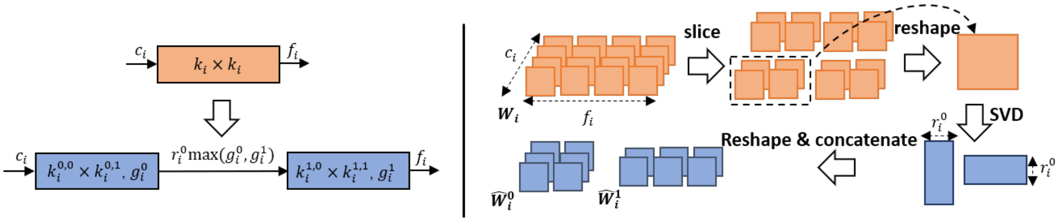

Figure 2: **Left:** Each building block contains two convolutional layers which are characterised by the design parameters: $k_i^{j,m}$, g_i^j and r_i^0. **Right:** The process of deriving the low-rank weight tensors.

scheme and the decomposition rank. Among the design parameters of *LR-space*, $k_i^{j,m}$ and the g_i^j determine how the slicing and reshaping are performed, which correspond to the decomposition scheme F, while r_i^0 represents the decomposition rank.

3.3. Residual Extension of *LR-space*

We also propose a residual-style building block as an extension of the *LR-space* in order to further refine metrics trade-off. Continuing the previous analysis on the weight tensors, the process of low-rank approximation replaces $\boldsymbol{W_i}$ with $\boldsymbol{\hat{W}_i^1}$, $\boldsymbol{\hat{W}_i^0}$, and injects the error $\boldsymbol{E_i}$ at the same time.

$$\boldsymbol{W_i} = F(\boldsymbol{\hat{W}_i^1}, \boldsymbol{\hat{W}_i^0}) + \boldsymbol{E_i} \tag{3}$$

So far, $\boldsymbol{E_i}$ is completely ignored and pruned. Alternatively, we can choose to keep part of $\boldsymbol{E_i}$ by further applying low-rank approximation to $\boldsymbol{E_i}$. Therefore,

$$\boldsymbol{W_i} = \sum_{b=0}^{1} F_b(\boldsymbol{\hat{W}_i^{1,b}}, \boldsymbol{\hat{W}_i^{0,b}}) \tag{4}$$

which corresponds to a residual-style building block with 2 branches whose outputs are element-wise summed, each branch containing two unique *low-rank layers*. The superscript b is to distinguish these two branches. The computation in the first branch is to approximate $\boldsymbol{W_i}$ while the second branch is to approximate $\boldsymbol{E_i}$. Although both branches have been low-rank approximated, their decomposition schemes and decomposition ranks can differ with each other, which makes the low-rank approximation more fine-grained compared with merely having one branch and increasing its rank.

4. Searching Algorithm

Having defined the design space, the proposed framework considers the following multi-objective optimisation problem. The formulation aims to minimise the required number of computations per layer and at the same time to maximise the achieved accuracy of the network, subject to the form of decomposition.

$$\min_{k_{i,b}^{j,m}, g_{i,b}^{j}, r_{i,b}^{0}} FLOP(\boldsymbol{\hat{W}_{i,b}^{j}}), \max_{k_{i,b}^{j,m}, g_{i,b}^{j}, r_{i,b}^{0}} ACC(\boldsymbol{\hat{W}_{i,b}^{j}}),$$
$$i \in [0, L-1], j \in \{0,1\}, m \in \{0,1\}, b \in \{0,1\} \tag{5}$$

$FLOP$ and ACC represents the total operations and validation accuracy of the low-rank model respectively.

4.1. Gradient-descent NAS

The framework uses a standard gradient-descent NAS approach [29] to solve the above optimisation problem. As Fig. 1 shows, each convolutional layer in the pre-trained model is replaced with a super block during the search. The per-layer super block is constructed by exhaustively traversing through all the combinations of the design parameters of the *LR-space* and instantiating the corresponding building blocks. Notice that the *original layer* is also included as a candidate inside the super block, which provides the option to not compress this layer at all.

The super block provides a Gumbel-Softmax weighted sum of the candidate building blocks drawn from *LR-space*. Within this weighted sum, the weight of each candidate is given by $\boldsymbol{\theta_i}$, which is known as the sampling parameter in the previous literature to be distinguished from $\hat{\mathbb{W}}_i$, the actual weight tensors of convolution. During the search, the sampling parameter $\boldsymbol{\theta_i}$ gets updated with gradient descent by minimising the following multi-objective loss function.

$$l_{nas}(\boldsymbol{\theta}) = l_{ce} \cdot [log(FLOP_{\hat{\mathbb{N}}})/log(FLOP_{\mathbb{N}})]^{\beta} \tag{6}$$

where l_{ce} is the cross-entropy loss, while $FLOP_{\hat{\mathbb{N}}}$ and $FLOP_{\mathbb{N}}$ represents the FLOPs of the compressed model and the original model respectively. β is a hyperparameter which implicitly controls the compression ratio. When calculating the FLOPs of the super block, we also take the weighted sum of each candidate by the sampling parameter. At the end of the search, the candidate with the largest value of the sampling parameter is finally selected to replace the *original layer*.

4.2. Reduce Searching Cost

It is well-known that NAS can be very time-consuming and GPU-demanding given the huge size of the design space to be explored. For example, considering the *LR-space* of a single convolutional layer where (f_i, c_i, k_i, k_i) is $(64, 64, 3, 3)$, there are 74902 candidates to be compared for approximating that layer. In this section, we introduce some techniques to help explore the design space more efficiently, but at the same time, we can still keep the design choices of our framework more fine-grained than the previous work.

Before the searching starts, we prune the *LR-space* to reduce the number of candidate configurations considered by the framework. The following strategies are considered:

- prune by FLOPs, we perform a grid search across the FLOPs. For example, we are only interested in those candidates whose FLOPs is close to $\{95\%, 90\%, 85\%, ...\}$ of the *original layer*.
- prune by accuracy, we use a proxy task where only one layer from the original network is compressed using the candidate configuration, while all other layers are left uncompressed, and we evaluate the corresponding accuracy degradation. The candidate will be pruned from the design space if this degradation is larger than a pre-defined threshold τ_{proxy}.

During the searching, we explore the space by an iterative searching method, which only searches for the configuration of one branch each time rather than simultaneously. We start with the case that there is only one branch and no residual structure inside the building block. With the help of NAS, we find the optimal configuration of design parameters belonging to that branch and fix that configuration. Afterwards, we add the residual branch into the building block and we start the searching again.

Moreover, during every forward pass of searching, we sample and only compute the weighted sum of two candidates rather than all of them. The probabilities of each candidate getting sampled are the softmaxed $\boldsymbol{\theta_i}$. This technique was proposed by [4] to reduce GPU memory.

Algorithm 1 Iterative Searching

```
$\boldsymbol{E_{i,0}} = \boldsymbol{W_i}$
for $b \in \{0, 1\}$ do
    identify optimal $F_b(\boldsymbol{\hat{W}_i^{1,b}}, \boldsymbol{\hat{W}_i^{0,b}})$ to approximate $\boldsymbol{E_{i,b}}$
    $\boldsymbol{E_{i,b+1}} = \boldsymbol{E_{i,b}} - F_b(\boldsymbol{\hat{W}_i^{1,b}}, \boldsymbol{\hat{W}_i^{0,b}})$
end for
```

5. Experiments

The proposed SVD-NAS framework is evaluated on the ImageNet dataset using pre-trained ResNet-18, MobileNetV2 and EfficientNet-B0 coming from torchvision[1]. Thanks to techniques discussed in 4.2, we can perform the searching on a single NVIDIA GeForce RTX 2080 Ti or a GTX 1080 Ti. Details of hyperparameters set-up can be found in this paper's supplementary material.

5.1. Performance Comparison

According to the previous work [1, 20, 16], the data-limited problem setting can be interpreted as two kinds of experiment set-up: post-training, where no training data is allowed for fine-tuning, and few-sample training, only a tiny subset of training data can be used for fine-tuning.

For both set-ups, the proposed SVD-NAS framework's performance was evaluated and compared against existing works on CNN compression. The metrics of interest include, the reduction of FLOPs and parameters, in percentage (%), as well as the degradation of Top-1 and Top-5 accuracy, in percentage point (pp).

5.1.1 Post-training without tuning

We firstly report the performance of the compressed model without any fine-tuning. Table 1 presents the obtained results of the proposed framework for a number of networks and contrasts them to current state-of-the-art approaches.

ALDS [18] and LR-S2 [8] are two automatic algorithms based on the MSE heuristic, while F-Group [21] is a handcrafted design. The results show that SVD-NAS outperforms all existing works when no fine-tuning is applied. In more details, on ResNet-18 and EfficientNet-B0, our work produced designs that achieve the highest compression ratio in terms of FLOPs as well as parameters, while maintaining a higher accuracy than other works. In terms of MobileNetV2, we achieve the best accuracy-FLOPs trade-off but not the best parameters reduction, as we do not include the number of parameters as an objective in (6).

5.1.2 Post-training, but tuning with synthetic data

Even though our framework outperforms the state-of-the-art approaches, we still observe a significant amount of accuracy degradation when no fine-tuning is applied. As training data are not available in the post-training experiment set-up, the proposed framework considers the generation of an unlabelled synthetic dataset and then uses knowledge distillation to guide the tuning of the parameters of the obtained model.

Inspired by the previous work on post-training quantisation [5], the synthetic data are generated by optimising the following loss function on the randomly initialised image $\boldsymbol{I}$:

[1] https://github.com/pytorch/vision

$$
\begin{aligned}
l_{bn}(\boldsymbol{I}) &= \alpha[(\mu'_{\boldsymbol{I}})^2 + (\sigma'_{\boldsymbol{I}} - 1)^2] \\
&+ \sum_{i=0}^{L-1} \frac{1}{f_i} \sum_{f=0}^{f_i-1} [(\mu'_f - \mu_f)^2 + (\sigma'_f - \sigma_f)^2] \quad (7)
\end{aligned}
$$

μ_f and σ_f is the running mean and running standard deviation stored in the batch normalisation layers from the pre-trained model. μ'_f and σ'_f represents the corresponding statistics recorded when the current image is fed into the original pre-trained network. In addition, $\mu'_{\boldsymbol{I}}$ and $\sigma'_{\boldsymbol{I}}$ represents the mean and standard deviation of the current image itself, while α is a hyperparameter that balances these two terms.

Once the synthetic dataset is generated, we treat the original pre-trained model as the teacher and the compressed low-rank model as the student. Since the synthetic dataset is unlabelled, the knowledge distillation focuses on minimising the MSE of per-layer outputs. According to the results in Table 1, the synthetic dataset can improve the top-1 accuracy by 2.44pp-7.50pp on three targeted models, which enlarges our advantage over the state-of-the-art methods.

5.1.3 Few-Sample Training

Few-sample training differs from the previous post-training in that now a small proportion of training data are available for the fine-tuning purpose. Specifically, for evaluation, we randomly select 1k images from the ImageNet training set as a subset and fix it throughout the experiment. During the fine-tuning, we use the following knowledge distillation method,

$$
\begin{aligned}
l_{kd}(\hat{\mathbb{W}}_i) &= \sum_{i=0}^{L-1} MSE(\hat{y}_i, y_i) \\
&+ \alpha_{kd} \cdot T_{kd}^2 \cdot l_{KLdiv} + (1 - \alpha_{kd}) \cdot l_{ce} \quad (8)
\end{aligned}
$$

where $MSE(\hat{y}_i, y_i)$ stands for the Mean Square Error on the outputs of convolutional layers, while l_{KLdiv} is the KL divergence on logits which are softened by temperature T_{kd} (set as 6). l_{ce} is the cross-entropy loss on the compressed model. Hyperparameter α_{kd} is set as 0.95.

As none previous work has reported any result on few-sample low-rank approximation, we compare our framework with existing works on few-sample pruning instead. From Table 2, our SVD-NAS framework provides a competitive accuracy-FLOPs trade-off on ResNet-18, especially when we are interested in those structured compression methods. We also observe that the compression ratio of MobileNetV2 achieved by our method is relatively less profound than the pruning methods, as that network contains a lot of depth-wise and point-wise convolutions which are less redundant in terms of the ranks of weight tensors.

Table 1: Post-training results of low-rank approximation. * no fine-tuning. ** fine-tuning with 25k synthetic images

Model	Method	Δ FLOPs (%)	Δ Params (%)	Δ Top-1 (pp)	Δ Top-5 (pp)
ResNet-18	**SVD-NAS**	**-58.60**	**-68.05**	-13.35^* $\mathbf{-5.85}^{**}$	-9.14^* $\mathbf{-3.34}^{**}$
	ALDS [18]	-42.31	-65.14	-18.70	-13.38
	LR-S2 [8]	-56.49	-57.91	-38.13	-33.93
	F-Group[21]	-42.31	-10.66	-69.34	-87.63
MobileNetV2	**SVD-NAS**	**-12.54**	-9.00	-15.09^* $\mathbf{-9.99}^{**}$	-7.79^* $\mathbf{-6.11}^{**}$
	ALDS [18]	-2.62	**-37.61**	-16.95	-10.91
	LR-S2 [8]	-3.81	-6.24	-17.46	-10.34
EfficientNet-B0	**SVD-NAS**	**-22.17**	**-16.41**	-10.11^* $\mathbf{-7.67}^{**}$	-5.49^* $\mathbf{-4.06}^{**}$
	ALDS [18]	-7.65	-10.02	-16.88	-9.96
	LR-S2 [8]	-18.73	-14.56	-22.08	-14.15

Table 2: Comparison with few-sample pruning.

Model	Method	Struct.	Δ FLOPs (%)	Δ Params (%)	Δ Top-1 (pp)	Δ Top-5 (pp)
ResNet-18	**SVD-NAS**	yes	**-59.17**	**-66.77**	**-3.95**	-2.36
	FSKD [16]	yes	-59.01	-64.64	-6.01	-
	Reborn [26]	yes	-33.33	-	-	-4.24
	POT [12]	no	-	-50.00	-	**-1.48**
MobileNetV2	**SVD-NAS**	yes	**-14.17**	-10.66	-6.63	-3.61
	MiR [27]	yes	-13.30	-7.70	**-1.80**	-
	POT [12]	no	-	**-40.00**	-	**-2.87**

5.1.4 Full Training

Although we are mainly interested in the data-limited scenarios, it is also interesting to remove the constraint of data availability and check the results when the full training set is available. Under this setting, we totally abandon knowledge distillation and only keep the cross-entropy term in (8) for fine-tuning. All other experiment settings remain the same as before.

Table 3 presents the obtained results. In the case of ResNet-18, SVD-NAS reduces 59.17% of the FLOPs and 66.77% of parameters without any penalty on accuracy. In the case of MobileNetv2, the proposed framework produces competitive results as the other state-of-the-art works.

To summarise, we observe that the advantage of our framework over SOTA is correlated with the problem setting on data availability, as the advantage is more prominent in post-training and few-sample training, but is less evident in full training. This finding suggests that when the data access is limited, the design choices of low-rank approximation should be more carefully considered, while when a

Table 3: Fine-tune low-rank network on the full training set.

Model	Method	Δ FLOPs (%)	Δ Params (%)	Δ Top-1 (pp)	Δ Top-5 (pp)
ResNet-18	**SVD-NAS**	-59.17	-66.77	**+0.03**	**+0.10**
	ALDS [18]	-43.51	-66.70	-0.40	-0.05
	S-Conv [2]	-51.23	-52.18	-0.63	-
	MUSCO [7]	-58.67	-	-0.47	-0.30
	ADMM-TT [31]	-59.51	-	-	0.00
	CPD-EPC [22]	-67.64	**-73.82**	-0.69	-0.15
	TRP [30]	**-68.55**	-3.76	-2.33	
MobileNetV2	**SVD-NAS**	-14.17	-10.66	-1.66	-1.90
	Shared [10]	0.00	-7.43	**+0.39**	**+0.37**
	ALDS [18]	-11.01	**-32.97**	-1.53	-0.73
	S-Conv [2]	**-19.67**	-25.14	-0.90	-

lot of training data are available, the performance gap between different design choices can be compensated through fine-tuning.

5.2. Ablation Study

In this section, we analyse the individual contribution of each part of our framework.

5.2.1 Design Space

As motioned earlier, although the low-rank approximation problem involves choosing both decomposition scheme and decomposition rank, many existing works [17, 8] focused on proposing a single decomposition scheme and lowering the rank of the approximation that would minimise the required number of FLOPs with minimum impact on accuracy.

In our framework, we construct the *LR-space* which expands the space of exploring different decomposition schemes and ranks on a per-layer basis. In Fig. 3 Left, the accuracy vs. FLOPs trade-off is plotted for the possible configurations when considering a single layer. As the figure demonstrates, the optimal decomposition scheme and rank depend on the FLOPs allocated to each layer, and the Pareto front is populated with a number of different schemes. These results confirm that previous work which overlooked the choice of decomposition scheme would lead to sub-optimal performance.

5.2.2 Searching

Many previous works [24, 8, 18] exploit the MSE heuristic to automate the design space exploration of low-rank approximation. Although their methods would result in a faster exploration, that would penalise the quality on estimating the accuracy degradation. Fig. 3 Right confirms that MSE of the weight tensor is a poor proxy of the network accuracy degradation. We observed that some configurations of design parameters have similar MSE, but they lead to distinct accuracy results. Therefore, it demonstrates the necessity of using NAS to explore the diverse *LR-space*, which directly optimises accuracy versus FLOPs.

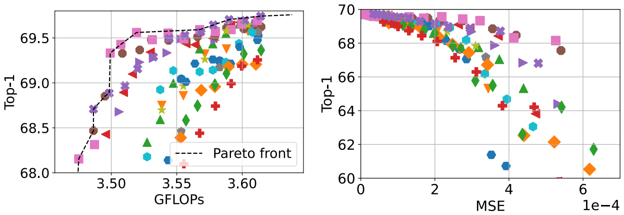

Figure 3: Explore the *LR-space* of the second convolutional layer in ResNet-18. All other layers are not compressed. Each type of marker corresponds to a specific decomposition scheme. **Left:** Accuracy versus FLOPs **Right:** Accuracy versus MSE

5.2.3 Synthetic Dataset

To investigate the proper quantity of the synthetic images, Fig. 4 Left demonstrates the top-1 accuracy vs number of FLOPs for 1k, 5k and 25k synthetic images. To distinguish the different experiment configurations that we carried on, they are denoted in the form of Bx-SDy, where x represents the number of branches in the building block (the residual-style building block is disabled when x=1), and y represents the number of synthetic images. The results demonstrate that the accuracy improvement from 5k to 25k images is below 0.5pp.

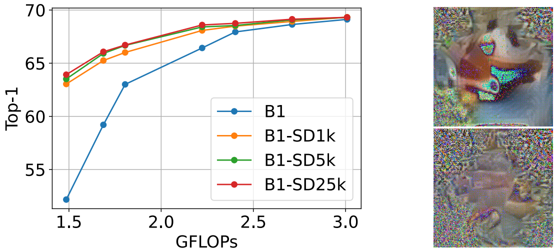

Figure 4: **Left:** ResNet-18 results of varying the size of synthetic dataset. **Right Upper and Lower:** Synthetic images of MobileNetV2, taken from our approach and ZeroQ respectively.

In terms of the quality of the synthetic dataset, although our method is inspired by ZeroQ [5], we found their algorithm is not directly applicable to our problem. Compared with ZeroQ, we scale the loss of batch normalisation layers by the number of channels, as (7). Fig. 4 Right shows two sample images taken from our method and the original ZeroQ implementation respectively on MobileNet-V2.

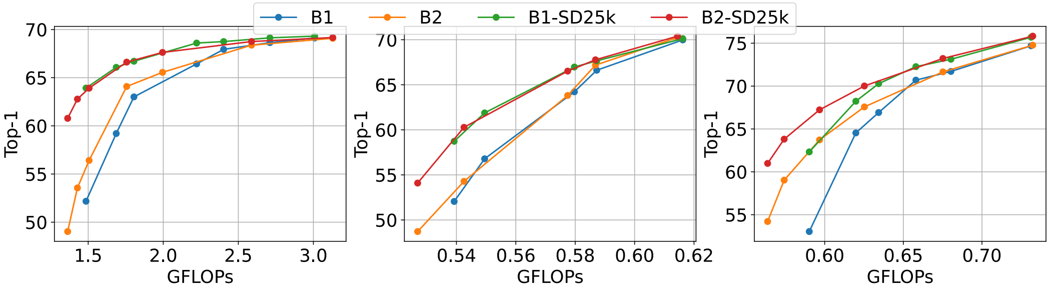

Figure 5: Ablation study of top-1 accuracy-FLOPs trade-off for different configurations of the SVD-NAS approach. **Left:** ResNet-18. **Centre:** MobileNetV2. **Right:** EfficientNet-B0

Without the introduced scaling term, the synthetic image becomes noisy, which we found, is more likely to cause overfitting during the fine-tuning.

5.2.4 Residual Block

An investigation was performed in Fig. 5 to assess the impact of multiple branches in the building block of the proposed framework. The system's performance was assessed under considering building blocks with one ($B1$) and two branches ($B2$). In the case of ResNet-18 and EfficientNet-B0, moving from $B1$ to $B2$, improvement in accuracy is obtained, which increases with the compression ratio.

However, we observed that this gap shrinks when the synthetic dataset is applied, suggesting that the multiple branch flexibility is a more attractive option when we have no training data at all and cannot generate synthetic dataset; in the later case, for example, when the pre-trained model has no batch normalisation layer at all or the batch normalisation layer has already been fused into the convolutional layer.

5.3. Latency-driven Search

Till now, the framework has focused on reducing the FLOPs without considering the actual impact on the execution time reduction on a hardware device. We extended the framework by integrating the open-source tool nn-Meter [33] to provide a CNN latency lookup table targeting a Cortex-A76 CPU. The lookup table is then used to replace the FLOPs estimate in (6). Having exposing the latency in the execution of a layer to the framework, we optimised the targeted networks for execution on a Pixel 4 mobile phone. We used a single thread and fixed the batch size to 1. Table 4 presents the obtained results measured on device, showing that FLOPs can be used as a proxy for performance, especially for ResNet-18 and MobileNetV2. EfficientNet contains SiLU and squeeze-and-excitation operations [25], that currently are not well optimised on CPU and lead to larger discrepancy between latency and FLOPs as a measure of performance.

Table 4: Latency-driven search results on Pixel 4

Model	Objective	Δ Top-1 (pp)	Δ FLOPs (%)	Δ Latency (%)	Latency (ms)
ResNet-18	FLOPs	-5.83	**-59.17**	-44.52	76.70
	Latency	-5.67	-54.78	**-49.46**	69.87
MobileNetV2	FLOPs	-9.99	**-12.54**	-1.03	30.66
	Latency	-8.22	-9.55	**-4.75**	29.51
EfficientNet-B0	FLOPs	-9.45	**-22.85**	-1.92	67.08
	Latency	-10.49	-21.39	**-6.46**	63.97

6. Conclusion

This paper presents SVD-NAS, a framework that significantly optimises the trade-off between accuracy and FLOPs in data-limited scenarios by fusing the domain of low-rank approximation and NAS. Regarding future work, we will look into further expanding the *LR-space* by including non-SVD based decomposition methods.

Acknowledgement

For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

References

[1] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*, 2018.

[2] Yash Bhalgat, Yizhe Zhang, Jamie Menjay Lin, and Fatih Porikli. Structured convolutions for efficient neural network design. *Advances in Neural Information Processing Systems*, 33:5553–5564, 2020.

[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[5] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

[6] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *arXiv preprint arXiv:1404.0736*, 2014.

[7] Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Philip Blagoveschensky, Andrzej Cichocki, and Ivan Oseledets. Automated multi-stage compression of neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[8] Yerlan Idelbayev and Miguel A Carreira-Perpinán. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059, 2020.

[9] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[10] Woochul Kang and Daeyeon Kim. Deeply shared filter bases for parameter-efficient convolutional neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[11] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[12] Ivan Lazarevich, Alexander Kozlov, and Nikita Malinin. Post-training deep neural network pruning via layer-wise calibration. *arXiv preprint arXiv:2104.15023*, 2021.

[13] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[14] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[15] Chong Li and CJ Shi. Constrained optimization based low-rank approximation of deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 732–747, 2018.

[16] Tianhong Li, Jianguo Li, Zhuang Liu, and Changshui Zhang. Few sample knowledge distillation for efficient network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14639–14647, 2020.

[17] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5623–5632, 2019.

[18] Lucas Liebenwein, Alaa Maalouf, Dan Feldman, and Daniela Rus. Compressing neural networks: Towards determining the optimal layer-wise decomposition. *Advances in Neural Information Processing Systems*, 34, 2021.

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[20] Szymon Migacz. 8-bit inference with tensorrt. In *GPU technology conference*, volume 2, page 7, 2017.

[21] Bo Peng, Wenming Tan, Zheyang Li, Shun Zhang, Di Xie, and Shiliang Pu. Extreme network compression via filter group approximation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 300–316, 2018.

[22] Anh-Huy Phan, Konstantin Sobolev, Konstantin Sozykin, Dmitry Ermilov, Julia Gusak, Petr Tichavskỳ, Valeriy Glukhov, Ivan Oseledets, and Andrzej Cichocki. Stable low-rank tensor decomposition for compression of convolutional neural network. In *European Conference on Computer Vision*, pages 522–539. Springer, 2020.

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[24] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[25] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[26] Yehui Tang, Shan You, Chang Xu, Jin Han, Chen Qian, Boxin Shi, Chao Xu, and Changshui Zhang. Reborn filters: Pruning convolutional neural networks with limited data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5972–5980, 2020.

[27] Huanyu Wang, Junjie Liu, Xin Ma, Yang Yong, Zhenhua Chai, and Jianxin Wu. Compressing models with few samples: Mimicking then replacing. *arXiv preprint arXiv:2201.02620*, 2022.

[28] Min Wang, Baoyuan Liu, and Hassan Foroosh. Factorized convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 545–553, 2017.

[29] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[30] Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Wenrui Dai, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. Trained rank pruning for efficient deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 14–17. IEEE, 2019.

[31] Miao Yin, Yang Sui, Siyu Liao, and Bo 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*, pages 10674–10683, 2021.

[32] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[33] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 81–93, 2021.

[34] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015.

[35] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and pattern Recognition*, pages 1984–1992, 2015.