S©MR

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ
Siberian Electronic Mathematical Reports
http://semr.math.nsc.ru
ISSN 1813-3304

Vol. 20, No. 2, pp. 144–145 (2023) УДК ???.??
https://doi.org/10.33048/semi.2023.20.??? MSC ??X??

DIFFERENTIABLE PRUNING BY CONVOLUTIONAL PROJECTION

D.S. CHUDAKOV AND **V.B. BERKOV**

Communicated by P.P. Petrov

Abstract: Structured pruning is a widely used technique for reducing the computational complexity of neural networks. The standard pruning process typically consists of two stages: first, identifying and removing the least significant blocks of parameters, and second, fine-tuning the network to restore its original accuracy. However, this separation of stages often results in suboptimal outcomes. In this paper, we propose a novel convolutional neural network pruning procedure that is fully end-to-end and differentiable. Our approach involves projecting convolutional kernels into a reduced number of channels through learnable linear transformations, implemented using 1x1 convolutional layers. Specifically, for each convolutional layer to be pruned, we insert learnable convolutions both before and after the layer. The pre-layer expands the number of channels to the original size, while the post-layer reduces the number of channels. Only these auxiliary convolutions are trained, after which the learned transformations are fused with the original layer to produce a pruned convolutional kernel. The proposed method demonstrates superior accuracy compared to traditional pruning techniques, particularly when removing a substantial number of parameters.

Keywords: Deep Learning, Computer Vision, Pruning.

The work of V. Berikov was supported by the State Contract of Sobolev Institute of Mathematics, project No. FWNF-2022-0015.

Received January, 1, 2023, Published December, 31, 2023.

1 Introduction

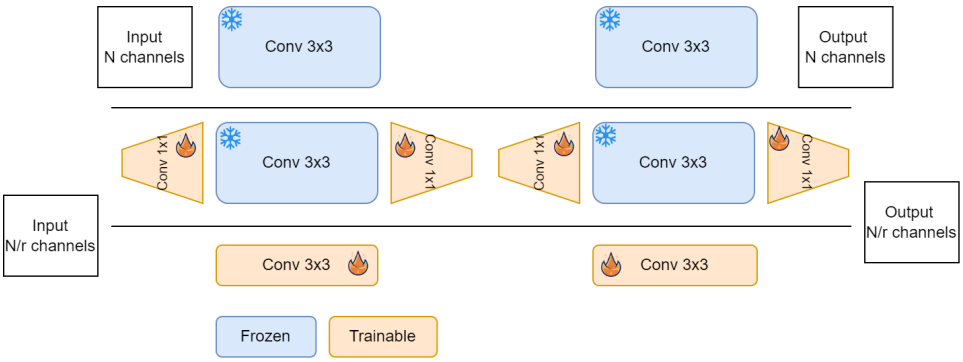

FIG. 1. The method is composed of two main phases. In the initial phase, projection convolution layers are inserted before and after each convolutional layer, and these layers are subsequently fine-tuned. During this stage, the number of channels kept after pruning is adjusted to be consistent between adjacent layers. In the subsequent phase, the convolutional layers are merged with the projection layers, after which the network may undergo additional fine-tuning.

Pruning [2] is an optimization method for neural networks [13] aimed at reducing their size and computational complexity by removing less significant parameters or layers. This approach is particularly important for deploying models on devices with limited computational resources, such as mobile phones or embedded systems, as well as for accelerating algorithm performance and reducing energy consumption, making them more efficient and environmentally friendly.

There are two primary approaches to pruning: structured and unstructured. Unstructured pruning arbitrarily removes parameters from the layers of a neural network. For this method to achieve real performance gains, it requires hardware that supports sparse computations and a high level of sparsity, often with some structural patterns. Structured pruning [1], on the other hand, removes parameters in connected blocks, enabling their complete elimination from the neural network (e.g., removing an entire convolutional filter). While this simplifies the deployment of pruned networks, it makes the pruning task more complex. This approach requires evaluating the significance of parameters collectively, as their influence can extend far beyond a single layer. In this paper, we focus on structured pruning.

A common pruning procedure involves two stages [5]. In the first stage, the importance of each block of parameters is evaluated, and the least significant blocks are removed. In the second stage, the neural network is retrained to recover the accuracy lost after parameter removal. Typically, this process is repeated iteratively, with each cycle removing a small portion of the

parameters to improve the final network's accuracy. However, since these two stages are separate, the solution is often suboptimal. Numerous heuristics are required to estimate the contribution of each parameter block to the final accuracy. Direct evaluation of this contribution is infeasible because it would involve testing all possible combinations of removed filters and retraining after each combination.

We propose a novel differentiable end-to-end approach (Fig. 1) where the stages of filter removal and layer retraining are not separated. We linearly project convolutional kernels onto convolutions with fewer channels, resulting in a fully differentiable method that eliminates the need for additional assumptions about the properties of neural network parameters and the application of heuristics. For example, the common assumption that removing parameters with minimal impact on accuracy will yield better final accuracy after retraining is no longer necessary. The method is straightforward to implement: each target layer is wrapped with 1x1 convolutions for projection, and only the new parameters are retrained. Before deploying the network, the projection convolutions are fused with the original layers to produce the final pruned network.

We evaluated the accuracy of our method on several popular neural network architectures and demonstrated its ability to improve accuracy. Additionally, we provided theoretical justification for the advantages of our approach.

The remainder of this paper is organized as follows: In section 2, we review the main approaches to pruning. In section 3, we formalize the pruning task by introducing the mathematical formulation that defines the objective and constraints for selecting a subset of weights while minimizing accuracy loss. Section 4 details our proposed approach based on differentiable convolutional projections. In section 5, we present our findings on the ImageNet classification task. Finally, section 6 analyzes the performance of our method, discusses its constraints, and outlines directions for future research.

2 Related Work

Pruning is an extensively studied method of model compression that has been researched since early works [2]. As neural network architectures became more complex, the necessity of structured pruning methods emerged [1][3], particularly for convolutional networks or transformer architectures [12]. In such architectures, independent parameter removal does not always lead to noticeable speedups for the neural network. Additionally, significant efforts have been made to theoretically justify pruning [5][6], especially in cases where extensive fine-tuning is required, with the costs of fine-tuning comparable to the initial training of the network. These studies often rely on the concept of "lottery tickets"in neural network weights, which disproportionately influence performance, allowing other parameters to be safely removed. Furthermore, these works explore when these lottery tickets can be identified and how iterative fine-tuning impacts the final results.

Pruning is frequently combined with other compression methods, such as distillation [7]. Distillation involves training a student network to mimic the outputs of a teacher network, transferring "dark knowledge,"which includes hidden inter-class distributions within the dataset, from teacher to student. This approach can significantly reduce training time, decrease the required amount of data, and improve the final model accuracy. In pruning tasks, the pruned network often serves as the student, while the original network acts as the teacher. Block-wise distillation methods [8] are also frequently employed to leverage intermediate activations, further enhancing results.

An essential step in pruning is identifying the least significant parameters for removal [1][3]. In the most basic approach, the importance of a weight is determined by the magnitude of its value, which is then aggregated across parameter blocks. Although this baseline method often yields acceptable quality, it relies on generic heuristics. The assumption that low-magnitude weights correspond to less significant parameters in deep neural networks is typically supported by regularization techniques. However, this approach does not consider other architectural features, such as the direct relationships between parameters throughout the network (e.g., until feature maps are downsampled) or the specific loss functions used.

More robust methods incorporate batch normalization statistics, such as variance, to more reliably estimate the contribution of parameter groups (e.g., filters) to the final result. One of the most popular and powerful techniques [14] evaluates parameter significance using Hessians. This evaluation involves expanding the network parameters into a Taylor series in the vicinity of a parameter's value. The second-order term is often ignored under the assumption that the network is already trained, and gradients are negligible, leaving the Hessian as the primary focus. However, calculating the Hessian is computationally prohibitive for commonly used industrial architectures due to its high complexity and memory requirements. To address this, approximations such as the squared first gradient are often used. This method is particularly convenient because, by adding gates (simple multipliers set to one), it is straightforward to collect the necessary gradients using standard deep learning frameworks[9]. Such methods also allow for the use of shared gates across connected parameters, explicitly indicating parameter dependencies across layers when gathering importance statistics.

Matrix decompositions [18] are also used in pruning tasks to compress convolutional operators [15]. Tucker decomposition [16] is one of the more common techniques, enabling parameter reduction without additional fine-tuning. Neural networks from the MobileNet family [4] can be seen as a decomposition of a convolution into three smaller convolutions with fewer parameters, inherently building a more parameter-efficient architecture.

Most existing approaches to differentiable pruning fundamentally rely on various relaxations of integer linear programming formulations [20]. In the classical pruning formulation, the use of soft masks [19] is a natural strategy. Although these methods frequently yield promising results, they demand

extensive hyperparameter tuning—particularly in selecting an appropriate coefficient schedule—to ensure that the soft mask ultimately converges to a hard mask.

3 Problem Statement

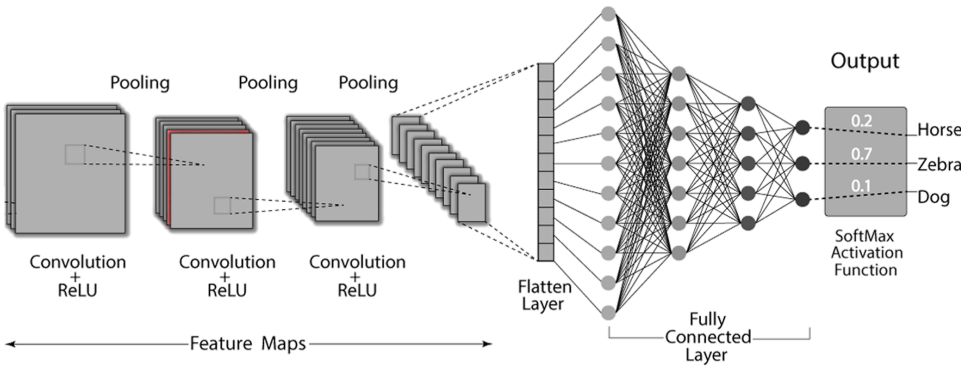

FIG. 2. In the illustration, a typical convolutional neural network architecture is depicted. The input feature map (i.e., the pixel color values of the image) is progressively transformed through successive applications of convolution with a learnable kernel, an activation function, and a pooling operation. The final prediction of class probabilities is generated using a fully connected layer. In structured pruning, individual filters in the convolution operator (and the resulting feature maps, highlighted in red) are removed.

We consider the pruning problem in the following setting. Our goal is to remove a fixed proportion of weights from each convolutional layer and subsequently fine-tune the model to achieve the highest possible accuracy. For consistency in comparisons and experiments, the fraction of pruned filters from each convolutional layer is fixed. Our experiments are conducted on the ImageNet classification task, where the goal is to minimize the risk of misclassifying an object belonging to one of 1000 classes.

We define the convolutional operator formally as follows. Let $X \in \mathbb{R}^{H \times W \times C_{\text{in}}}$ represent the input feature map, where H is the height, W is the width, and C_{in} is the number of input channels. Let $V \in \mathbb{R}^{k \times k \times C_{\text{in}} \times C_{\text{out}}}$ denote the convolutional kernel, where k is the kernel size and C_{out} is the number of output channels (filters). Let $b \in \mathbb{R}^{C_{\text{out}}}$ represent the bias vector. Given stride

s and padding p, the output feature map $Y \in \mathbb{R}^{H' \times W' \times C_{\text{out}}}$ is computed as

$$Y(i,j,c) = \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} \sum_{c'=1}^{C_{\text{in}}} V(u,v,c',c)\, X(i \cdot s + u - p,\ j \cdot s + v - p,\ c') + b(c),$$

where $i = 0, \ldots, H' - 1$, $j = 0, \ldots, W' - 1$, and $c = 1, \ldots, C_{\text{out}}$. The output dimensions H' and W' are given by $H' = \left\lfloor \frac{H+2p-k}{s} \right\rfloor + 1$ and $W' = \left\lfloor \frac{W+2p-k}{s} \right\rfloor + 1$. Square brackets $\lfloor \cdot \rfloor$ denote the floor function, which returns the greatest integer less than or equal to the argument.

Formally, let V denote the set of all trainable weights in the network (Fig. 2). We define a binary mask M of the same shape as V, which indicates retained weights $\big(M_{ij} \in \{0,1\}\big)$. To remove a fixed proportion ρ of weights in each convolutional layer ℓ, we enforce:

$$\frac{\|V^{(\ell)} \odot M^{(\ell)}\|_0}{\|V^{(\ell)}\|_0} = 1 - \rho,$$

where $V^{(\ell)}$ denotes the weights of layer ℓ, $M^{(\ell)}$ is the corresponding mask, and $\odot$ denotes element-wise multiplication.

This constraint ensures that exactly a fraction $1 - \rho$ of original weights in each convolutional layer remains active after pruning. Thus, if a layer initially contains a certain number of weights, only $1 - \rho$ of these weights remain active, while the remaining ρ fraction is pruned.

After pruning, we fine-tune the pruned model to minimize classification risk:

$$\min_{V,\,M} \quad \mathcal{L}\big(V \odot M\big) = \min_{V,\,M} \ \frac{1}{N} \sum_{n=1}^{N} \ell\big(f(x_n; V \odot M),\, y_n\big),$$

where ℓ is a suitable loss function (e.g., cross-entropy), $f(\cdot)$ is the neural network's forward pass, and $\{(x_n, y_n)\}_{n=1}^{N}$ are training samples, with x_n representing input objects and y_n their corresponding labels. Here, N is the total number of training samples.

The overall optimization objective is to identify both the weights V and binary mask M, such that after pruning the network (applying the mask), the average classification loss across the N training samples is minimized. This formulation captures the objective of retaining a fixed fraction of weights in each convolutional layer (as enforced by the constraint) while achieving optimal accuracy on the classification task.

4 Method

In this section, we explore the projection of convolutional operators into a lower-dimensional space. We begin by considering a convolution with an equal number of input and output channels, denoted as C (total number of channels), and a kernel size of k (spatial dimension of the convolutional kernel). Our goal is to obtain a convolution with $\lfloor C/r \rfloor$ input and output

channels, where $r > 0$ (compression ratio) represents the factor by which the number of channels is reduced. (Note that in all expressions, $\frac{C}{r}$ is to be interpreted as the rounded value to the nearest integer.)

To achieve this, two additional convolutions with a kernel size of 1×1 are placed on either side of the target convolution. Specifically:

- Before the target convolution, we insert a 1×1 convolution with $\lfloor C/r \rfloor$ input channels and C output channels.
- After the target convolution, another 1×1 convolution is added, with C input channels and $\lfloor C/r \rfloor$ output channels.

Figure 1 illustrates this arrangement. The group of these three convolutions can be replaced by a single convolution with a kernel size of k and $\lfloor C/r \rfloor$ input and output channels. The equivalent fused convolution weights W_{fused} (fused weight matrix) and bias b_{fused} (fused bias vector) are computed as follows:

$$W_{\text{fused}} = W_{\text{before}} \times \left(W_{\text{after}} \times W_{\text{target}}^{\top}\right)$$

$$b_{\text{fused}} = W_{\text{after}} \times b_{\text{target}}$$

where W_{before} and W_{after} are the weight matrices of the two 1×1 convolutions surrounding the target convolution, and W_{target} and b_{target} are the weight matrix and bias vector, respectively, of the target convolution.

This operation effectively combines the three convolutions into one while preserving functionality. To achieve this equivalence, the additional 1×1 convolutions are applied to the kernel of the target convolution. By surrounding each convolution in the neural network with such projecting convolutions, the number of channels in each layer is reduced by a factor of r.

The next step is to optimize these additional convolutions to minimize the neural network loss function on the target training dataset. During this optimization process, the most suitable projections for each filter are obtained. Unlike methods that directly seek projections for each convolution using principal component analysis (PCA), this approach makes no additional assumptions about the properties of neural network weights.

4.1. Method Complexity. In this subsection, we calculate the number of additional parameters that must be fine-tuned and show that they represent only a small fraction of the overall model parameters. The original number of parameters in a $k \times k$ convolution (with both input and output number channels equal to C) is given by $C^2k^2 + C$. To introduce two additional 1×1 layers with a compression ratio of r, we compute the overhead for each layer. The first 1×1 layer reduces the channel count from C to C/r, resulting in a cost of $C \cdot (C/r) + (C/r)$; the second 1×1 layer expands the channel count from C/r back to C, with a cost of $(C/r) \cdot C + C$. Therefore, the total number of additional parameters for these two layers becomes $2(C^2/r) + C + (C/r)$. The fraction of additional parameters relative to the original $k \times k$ convolution is

then

$$\text{Fraction of Additional parameters} = \frac{2(C^2/r) + C + (C/r)}{C^2k^2 + C}.$$

For large C, the bias terms (which are proportional to C) are relatively small ($o(C)$) compared to the weight terms (proportional to C^2). In the case of a 3×3 convolution (i.e., $k = 3$), the main term's ratio can be approximated as $(2C^2/r)/(9C^2)$, which simplifies to $2/(9r)$. For instance, this ratio is approximately 0.11 for $r = 2$ and 0.055 for $r = 4$, decreasing further for larger values of r.

For ResNet-34, most of the convolutional parameters come from 3×3 layers with channel counts of 64, 128, 256, or 512. In this scenario, the overhead is about 10% for $r = 2$, around 5% for $r = 4$, and roughly 2% for $r = 8$. In contrast, for ResNet-50, which features a bottleneck structure including a "reduce" 1×1 layer (converting from $4C$ to C), a "main" 3×3 layer (with C channels), and an "expand" 1×1 layer (converting from C back to $4C$), if both the 3×3 and the "reduce" 1×1 layers are wrapped with new 1×1 layers, the overhead becomes significant due to the large compression ratio: approximately 70% for $r = 2$, around 35% for $r = 4$, and roughly 15% for $r = 8$. If only the 3×3 layers (with channel counts of 64, 128, 256, or 512) are compressed, while the 1×1 layers are either left unchanged or replaced with newly initialized 1×1 layers to reduce overhead, the overall overhead is similar to that of ResNet-34's 3×3 layers, with about 10% for $r = 2$ and only a few percent for higher values of r.

For ResNet-34 and ResNet-50 projection training step takes only fraction of full parameters finetuning and takes roughly 10 minutes on modern consumer grade GPU.

5 Experimental Results

Таблица 1. Accuracy (%) of the ResNet18 neural network [3] on the ImageNet task [4] with different compression ratio.

Method	**Ratio, (%)**	**Pruning**	**Finetuning**
IENNP [1]	25	6.0	67.1
Projection pruning (ours)	25	55.0	68.4
IENNP [1]	50	1.9	60.9
Projection pruning (ours)	50	50.1	63.8
ResNet18 Original Top1 accuracy	–	69.758	69.758

We evaluated the effectiveness of our method on the ImageNet task using the ResNet18 (Table 1) and ResNet50 (Table 2) architectures. These architectures are among the most popular convolutional neural network architectures that do not utilize Depthwise Convolution layers, which cannot be linearly projected into a Depthwise Convolution layer with fewer filters. Our experiments demonstrate

Таблица 2. Accuracy (%) of the ResNet50 neural network [3] on the ImageNet task [4] with different compression ratio.

Method	**Ratio 25%**	**Ratio 50%**
IENNP	74.9	69.2
Projection pruning (Ours)	75.8	73.1
ResNet50 Original Top1 accuracy	76.13	76.13

an improvement in accuracy, with the performance gap increasing as the compression ratio grows.

In these experiments, our method demonstrates higher accuracy. Moreover, the higher the degree of compression, the more significant the improvement in performance achieved by our approach.

Table 1 further compares the accuracy before and after fine-tuning. Notably, the accuracy before fine-tuning is already sufficiently high and suitable for practical applications, whereas conventional pruning approaches result in lower pre-fine-tuning accuracy. The training phase for the projections is lightweight, as we only need to fine-tune a small number of parameters. In some cases, this phase can be used independently. For fine-tuning, we employed the standard scheduling and optimizers used to train the original models in the torchvision library.

Our experimental setup follows the default PyTorch training recipe for ImageNet training. The configuration details are as follows:

We utilize Stochastic Gradient Descent (SGD) as the optimizer with a momentum of 0.9. The initial learning rate is set to 0.5 and follows a cosine annealing schedule. A linear warm-up strategy is employed for the first 5 epochs, with an initial learning rate scaled by a decay factor of 0.01. To enhance generalization, several regularization techniques and data augmentation strategies are applied. These include label smoothing, mixup, cutmix, auto-augmentation, and random erasing. Additionally, a repeated augmentation sampler with multiple repetitions is used. For training and evaluation, images are resized using bilinear interpolation. The validation images are resized to 232×232 and then center-cropped to 224×224. The training images are randomly cropped to 176×176.

6 Discussion

Neural network pruning aims to select a subset of layers whose removal, followed by subsequent fine-tuning, causes the smallest possible accuracy degradation. Additional complications may arise from boundary conditions (e.g., specific accuracy or latency targets), but we will omit those for now. A commonly used criterion for selecting layers is:

$$I_m = \big(E(D, W) - E(D, W \mid w_m = 0)\big)^2,$$

where $E(\cdot)$ denotes some measure of the model's error, D denotes validation dataset, W model weights and m in $w_m = 0$ denote some index in weight. Applying this formula directly would require evaluating all possible combinations of filters to remove and then fine-tuning the network i.e. an approach that is computationally prohibitive. Therefore, researchers typically assume a "greedy selection" property [17] to reduce the complexity of pruning. Without this assumption, classical pruning procedures become nearly unfeasible. However, when employing classical independent methods of importance estimation, it is not difficult to construct a counterexample in which the greedy selection property fails (although how frequently this occurs in real networks trained on practical datasets remains an open question).

By contrast, gradient-based methods—such as *Importance Estimation for Neural Network Pruning*[14] tend to align more closely with the greedy selection property. In these approaches, the property is reinforced implicitly because filter-importance gradients are calculated using the same gate parameter values, which accounts for parameter dependencies across neighboring convolutional layers.

A second commonly assumed property is that the accuracy drop *before* fine-tuning is linearly related to the accuracy drop *after* fine-tuning. This assumption is more speculative, and violations frequently occur in practice. For example, if a filter is primarily responsible for rescaling activations, other filters may assume that role after fine-tuning; however, initially removing that filter can still cause a noticeable decline in accuracy.

Our proposed method does not require either of these assumptions. Because it is fully differentiable, it can naturally identify, during the training process itself, which layers will be most important for the already fine-tuned network. Furthermore, our method does not rely on an iterative prune-and-fine-tune cycle repeated many times. For example, *IENNP*[14] typically needs such an iterative routine because its Taylor series expansion becomes invalid if too many filters are removed at once (it assumes the network is fully trained and that the second-order term in the Taylor series is zero). While this iterative approach can indeed improve accuracy, it significantly slows down the pruning process.

The proposed method is particularly suitable for scenarios where high compression is required while maintaining a low computational budget during fine-tuning. A key limitation of methods such as *IENNP*[14] is their potential loss of accuracy. However, this can be mitigated through an iterative pruning strategy, where only a small proportion of weights is removed at each step until the target number of channels is reached. While this approach increases computational complexity, it is a well-established standard in model compression tasks.

Additionally, our method is not directly applicable to a widely used type of convolution, *DWC*[4]. Unlike standard convolutions, these operations do not allow for a differentiable linear transformation into convolutions with a reduced number of channels. As a result, our projection-based convolutions map DepthWise convolutions to standard convolutions, which, unexpectedly, increases the number of operations rather than reducing them.

Our approach has the potential to complement other differentiable pruning techniques, particularly those leveraging *relaxation-based integer programming methods*[19]. Such techniques could be applied to layers that cannot be linearly projected, such as *DWC*[4] or *MultiHead Self-Attention layers*[12]. Additionally, a more in-depth analysis of projection results could help develop a more reliable criterion for selecting the least important filters, further improving the efficiency and effectiveness of the pruning process.

7 Conclusion

In this paper, we introduced a novel pruning method for convolutional neural networks. Our approach is a fully differentiable, end-to-end technique, which eliminates the need for assumptions about network properties. As a result, it significantly improves the final accuracy of the pruned model.

References

[1] He, Y., Xiao, L.: Structured pruning for deep convolutional neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence (2023).
[2] LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. Advances in Neural Information Processing Systems **2** (1989).
[3] Blalock, D., Gonzalez Ortiz, J.J., Frankle, J., Guttag, J.: What is the state of neural network pruning? Proceedings of Machine Learning and Systems **2**, 129–146 (2020).
[4] Sinha, D., El-Sharkawy, M.: Thin MobileNet: An enhanced MobileNet architecture. In: 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), pp. 0280–0285. IEEE (2019).
[5] Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018).
[6] Zhou, H., Lan, J., Liu, R., Yosinski, J.: Deconstructing lottery tickets: Zeros, signs, and the supermask. Advances in Neural Information Processing Systems **32** (2019).
[7] Hinton, G.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015).
[8] Yu, C., Zhang, F., Chen, R., Wang, A., Liu, Z., Tan, S., Li, E.-P.: Decoupling dark knowledge via block-wise logit distillation for feature-level alignment. IEEE Transactions on Artificial Intelligence (2024).
[9] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems **32** (2019).
[10] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255. IEEE (2009).
[11] Targ, S., Almeida, D., Lyman, K.: ResNet in ResNet: Generalizing residual architectures. arXiv preprint arXiv:1603.08029 (2016).

[12] Vaswani, A.: Attention is all you need. Advances in Neural Information Processing Systems (2017).
[13] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems **25** (2012).
[14] Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11264–11272 (2019).
[15] Chen, S., Zhou, J., Sun, W., Huang, L.: Joint matrix decomposition for deep convolutional neural networks compression. Neurocomputing **516**, 11–26 (2023).
[16] Zhong, Z., Wei, F., Lin, Z., Zhang, C.: ADA-Tucker: Compressing deep neural networks via adaptive dimension adjustment Tucker decomposition. Neural Networks **110**, 104–115 (2019).
[17] Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions—I. Mathematical Programming **14**, 265–294 (1978).
[18] Oseledets, I.V.: Tensor-train decomposition. SIAM Journal on Scientific Computing **33**(5), 2295–2317 (2011).
[19] Cho, M., Adya, S., Naik, D.: PDP: parameter-free differentiable pruning is all you need. Advances in Neural Information Processing Systems **36** (2024).
[20] Fisher, M.L.: The Lagrangian relaxation method for solving integer programming problems. Management Science **50**(12_supplement), 1861–1871 (2004).

Dmitry Sergeevich Chudakov
Novosibirsk State University,
1, Pirogova str.,
630090, Novosibirsk, Russia
Email address: d.chudakovv@g.nsu.ru

Vladimir Borisovich Berikov
Sobolev Institute of Mathematics,
pr. Koptyuga, 4,
630090, Novosibirsk, Russia
Email address: berikov@math.nsc.ru