Neural Processing Letters (2024) 56:8
https://doi.org/10.1007/s11063-024-11445-y

Merging of Neural Networks

Martin Pašen[1,2] · Vladimír Boža[1]

Accepted: 25 November 2023 / Published online: 6 February 2024

Abstract
We propose a simple scheme for merging two neural networks trained with different starting initialization into a single one with the same size as the original ones. We do this by carefully selecting channels from each input network. Our procedure might be used as a finalization step after one tries multiple starting seeds to avoid an unlucky one. We also show that training two networks and merging them leads to better performance than training a single network for an extended period of time.

Keywords Neural networks · Machine learning · Training improvement

1 Introduction

Typical neural network training starts with random initialization and is trained until reaching convergence in some local optima. The final result is quite sensitive to the starting random seed as reported in [1, 2], who observed a 0.5% difference in accuracy between the worst and best seed on the Imagenet dataset and a 1.8% difference on the CIFAR-10 dataset. Thus, one might need to run an experiment several times to avoid hitting the unlucky seed. The final selected network is just the one with the best validation accuracy.

We believe that the discrepancy between starting seeds performance can be explained by selecting slightly different features in hidden layers in each initialization. One might ask a question: can we somehow select better features for network training? One approach is to train a bigger network and select the most important channels via channel pruning [3–6]. Training a big network, which is subsequently pruned, might be in many cases prohibitive, since increasing network width by a factor of two results in a four times increase in FLOPs and also might require a change in some hyperparameters (e.g. regularization, learning rate).

Here, we propose an alternative approach demonstrated in Fig. 1. Instead of training a bigger network and pruning it, we will train two same-sized networks and merge them together into one. The idea is that each training run would fall into different local optima and

Availability: https://github.com/fmfi-compbio/neural-network-merging.

✉ Vladimír Boža
boza@fmph.uniba.sk

1 Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia

2 Max Planck Institute for Multidisciplinary Sciences, Göttingen, Germany

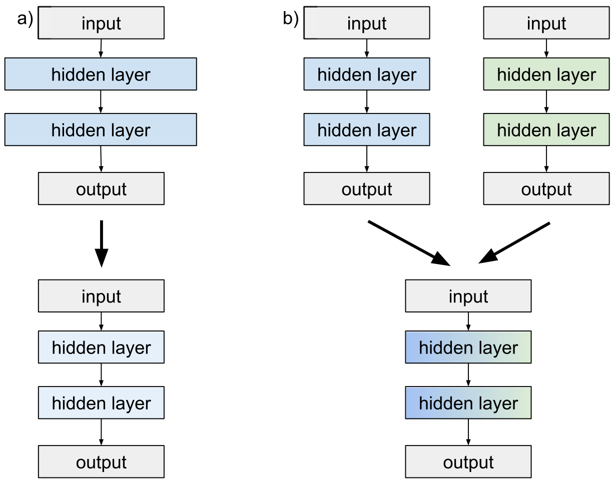

Fig. 1 Comparison between (**a**) training a bigger network and then pruning and (**b**) training two separate networks and then merging them together. Width of rectangles denotes the number of channels in the layer

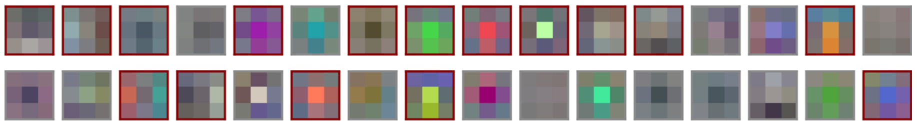

Fig. 2 Set of filters in the first layer of two ResNet20 networks trained on CIFAR-100 dataset with different starting seeds. Each row shows filters from one network. Selected filters for merged network are marked with a red outline

thus have different sets of filters in each layer, as shown in Fig. 2. We then can select a better set of filters than in the original networks and achieve better accuracy.

In summary, in this paper:

- We propose a procedure for merging two networks with the same architecture into one with the same architecture as the original ones.
- We demonstrate that our procedure produces a network with better performance than the best of original ones. On top of that, we also show that the resulting network is better than the same network trained for an extended number of epochs (matching the whole training budget for the merged network).

1.1 Related Work

Multiple approaches try to improve accuracy/size tradeoff for neural networks without the need for specialized sparse computation (such as in case of weight pruning). The most notable one is **channel pruning** [3–6]. Here, we first train a bigger network and then select the most important channels in each layer. The selection process usually involves assigning some score to each channel and then removing channels with the lowest score.

Another approach is **knowledge distillation** [7]. Knowledge distillation first trains a bigger network (teacher) and then uses its outputs as targets for a smaller network (student). It is hypothesized that by using outputs from larger network, the smaller network can also learn hidden aspects of data which are not visible in the dataset labels. However, it was shown that successful knowledge distillation requires training for a huge number of epochs (i.e. 1200) [8]. A slight twist to distillation was applied in [9] where bigger and smaller networks were cotrained together.

One can also use auxiliary losses to reduce redundancy and increase diversity between various places in the neural network [10].

A closely related approach to ours is known as model fusion [11, 12] (with further advancements detailed in [13]). Model fusion, like our method, combines multiple networks into a unified model. Given that neural networks exhibit permutation invariance, model fusion initially focuses on aligning neurons across various models, essentially identifying the optimal permutation of neurons before averaging their respective parameters. Notably, when neurons are aligned only by looking at relevant weights, there is no need for training data, which offers a significant advantage in federated learning scenarios. In contrast, our approach directly selects an optimal subset of neurons from both networks. Our strategy might be particularly advantageous in scenarios where specific neurons from one network cannot be seamlessly aligned with those of another.

2 Methods

Here, we describe our training and merging procedure. We will denote two networks, which will be merged as **teachers** and the resulting network as a **student**.

Our training strategy is composed of three stages:

1. Training of two teachers
2. **Merging procedure**, i.e. creating a student, which consists of the following substeps:
   (a) Layerwise concatenation of teachers into a big student
   (b) Learning importance of big student neurons
   (c) Compression of big student
3. Fine-tuning of the student

Training of teachers and fine-tuning of the student is just standard training of a neural network by backpropagation. Below, we describe how we derive a student from two teachers.

2.1 Layerwise Concatenation of Teachers into a Big Student

First, we create a “big” student by layerwise concatenation of teachers. The big student simulates the two teachers and averages their predictions in the final layer. This phase is just network transformation without any training, see Fig. 3. Concatenation of the convolutional layer is done in the channel dimension, see Fig. 4. Concatenation of the linear layer is done analogically in the feature dimension. We call the model "big student" because it has a doubled width.

2.2 Learning Importance of Big Student Neurons

We want the big student to learn to use only half of the neurons in every layer. So, after removing unimportant neurons, we will end up with the original architecture. Besides learning the relevance of neurons, we also want the two computational flows to interconnect.

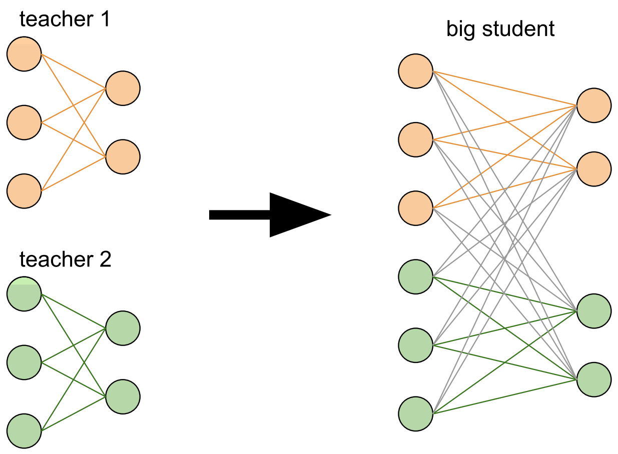

Fig. 3 Concatenation of linear layer. Orange and green weights are copies of the teacher's weights. Gray weights are initialized to zero. In the beginning, big student simulates two separate computational flows. But during the training, they can be interconnected

```
def merge_conv(conv1: nn.Conv2d, conv2: nn.Conv2d) -> nn.Conv2d:
    in_channels = conv1.in_channels
    out_channels = conv1.out_channels
    conv = nn.Conv2d(in_channels * 2, out_channels * 2,
                     kernel_size=conv1.kernel_size,
                     stride=conv1.stride, padding=conv1.padding,
                     bias=False)
    conv.weight.data *= 0
    conv.weight.data[:out_channels, :in_channels] = \
        conv1.weight.data.detach().clone()
    conv.weight.data[out_channels:, in_channels:] = \
        conv2.weight.data.detach().clone()
    return conv
```

Fig. 4 Pytorch code for concatenation of a convolutional layer in ResNet. Since convolutions are followed by Batch normalization, they do not use biases

There are multiple ways to find the most relevant channels. One can assign scores to individual channels [3, 4], or use an auxiliary loss to guide the network to select the most relevant channels. We have chosen the latter approach, inspired by [14]. It leverages the L0 loss presented in [15].

Let ℓ be a linear layer with k input features. Let g_i be gate assigned to feature f_i. Gate can be either opened $g_i = 1$ (student is using the feature) or closed $g_i = 0$ (student isn't using the feature). Before computing outputs of the layer, we first multiply inputs by gates, i. e. instead of computing $Wf + b$, we compute $W(f \cdot g) + b$. To make our model use only half of the features, we want $\frac{1}{n}\sum_1^k g_i = \frac{1}{2}$.

The problem with this approach is that g_i is discrete and is not trainable by the gradient descent. We used stochastic gates and continuous relaxation of L_0 norm presented in [15] to overcome this issue. The stochastic gates contain a random variable that has nonzero probability to be 0: $P[g_i = 0] > 0$, nonzero probability to be 1 $P[g_i = 1] > 0$, and is continuous on interval (0, 1). The reparameterization trick makes the distribution of the gates trainable by gradient descent.

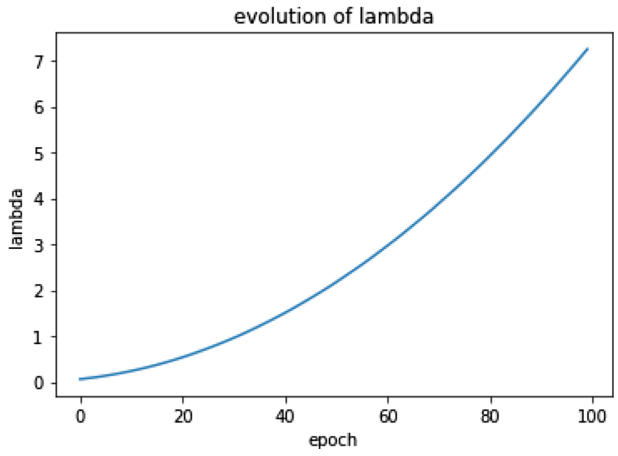

Fig. 5 Evolution of λ during training. For the sine problem we have used $\lambda_{t+1} = \lambda_t + 0.05 * \sqrt{\lambda_t}$ (where t is the epoch number)

To encourage the big student to use only half of the features of the layer, we use an auxiliary loss:

$$L^{\ell}_{half} = \left(\frac{1}{2} - \frac{1}{k}\sum_{1}^{k} P[g_i > 0]\right)^2$$

Note that our loss is different from the loss used in [15]. Whereas our loss forces the model to have exactly half of the gates opened, their loss pushes the model to use as few gates as possible.

Thus we are optimizing $L = L_E + \lambda \sum_{\ell} L^{\ell}_{half}$, where L_E is error loss measuring fit on the dataset and new hyperparameter λ is the proportion of importance of error loss and auxiliary loss.

Hyperparameter λ is sensitive and needs proper tuning. At the beginning of the training, it can not be too big, or the student will set every gate to be closed with a probability of 0.5. At the end of the training, it can not be too small, or the student will ignore the auxiliary loss in favor of the error loss. It will use more than half of the neurons of the layer and will significantly drop performance after the compression. We found that using the quadratic increase of λ during big student's training works sufficiently well, see Fig. 5.

We have implemented gates in a separate layer. We have used two designs of gate layers, one for 2d channels and one for 1d data. The position of gate layers is critical. For example, if a gate layer is positioned right before the batch norm, its effect (i. e. multiplying the channel by 0.1) would be countered by the batch norm, see Fig. 6.

2.3 Compression of Big Student

After learning of importance is finished, we select half of the most important neurons for every layer. Then, we compress each layer by keeping only the selected neurons as visualized in Fig. 7.

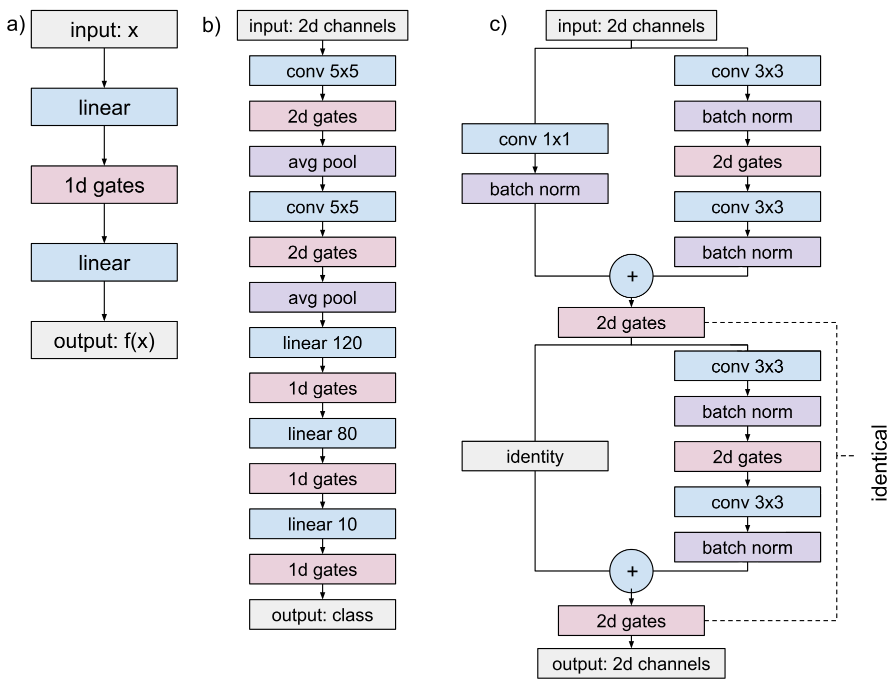

Fig. 6 Positions of gate layers (**a**) sine problem (**b**) LeNet c) two consecutive blocks in ResNet. Two of the ResNet gate layers have to be identical. If the layers would not be linked and for some channel i, the first gate would be closed while the second gate would be opened, the result of the second block, for that channel, would be $0 + f(x)$ instead of $x_i + f(x)$ which would defeat the whole purpose of ResNet and skip connections

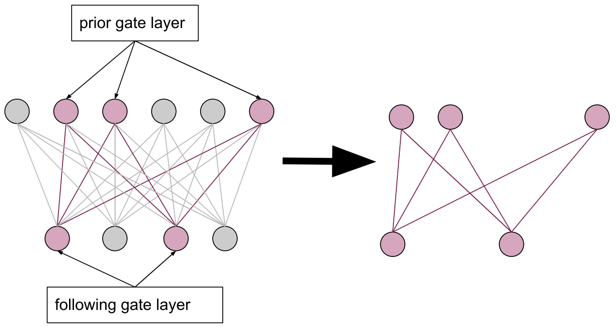

Fig. 7 Compression of the big student. On the left side is a linear layer of a big student. Prior and following gate layers decide which neurons are important. On the right side is a compressed layer. It consists of only the important neurons

3 Experimental Results

We compare our merging strategy to generic neural network training on several problems. First, we test our training strategy on a synthetic problem. We show that our merging strategy can learn better features than typical training. Then we test various architectures on image classification problems. We show that after the merging procedure, the resulting network is

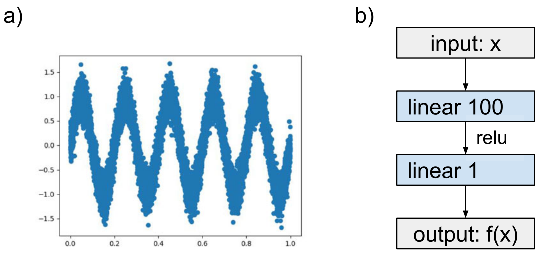

Fig. 8 a) Training dataset for sine problem consisting of 10000 samples where $x \sim \mathcal{U}(0, 1)$ and $y = sin(10\pi x) + z;\ z \sim \mathcal{N}(0,\ 0.2)$. b) Architecture of model for sine problem

Table 1 Summary of experimental results for sine problem

Task	Strategy	Min.	Max.	Median	Mean	Std.
Sine problem test MSE	Student	**0.049**	**0.116**	**0.078**	**0.077**	**0.015**
	bo3 model	0.105	0.373	0.253	0.240	0.076
	one model	0.053	0.363	0.281	0.249	0.097

Bold values indicate the best result (group in case of Table 2)
We report the mean squared error for each strategy. Strategy *student* (our strategy) uses 2/3 to train two teachers and 1/3 to train student (1/6 finding important features, 1/6 finetuning). Strategy *bo3 model* trains three models and picks the best. Strategy *one model* uses all epochs to train one model

better than the original ones and also better than training one network for an extended amount of time.

3.1 Training Strategies

We compare our network merging with generic training strategies using the same number of training epochs. Except for Imagenet and comparison with model fusion, we are comparing our training strategy with strategies *bo3 model* and *one model*. In the CIFAR-100 experiment, we also compare with *pruning* strategy. We run tests multiple times and report summary statistics (minimum, maximum, median, mean, and standard deviation of accuracy or mean squared error) over multiple runs.

In our merging strategy, *student*, we use two-thirds of epochs to train teachers and one-third to train student (one-sixth to find important neurons and one-sixth to fine-tune).

In the *bo3 model*, we train three models, each for one-third of epochs, and then we choose the best.

In the *one model* strategy, we use all epochs to train one model.

In the *pruning* strategy, we use the channel pruning method from [4]. We first train network with doubled width using two thirds or training budget and then incrementally prune for one sixth of the training budget and finally fine-tune for the rest of the time.

Note that each strategy uses a similar training budget. Also, during inference, all models use the equivalent amount of resources since they have exactly the same architecture.

3.2 Synthetic Dataset-Sine Problem

First, we want to confirm the idea that a network trained from random initialization might end in suboptimal local optima, and our merging procedure finds higher quality local optima.

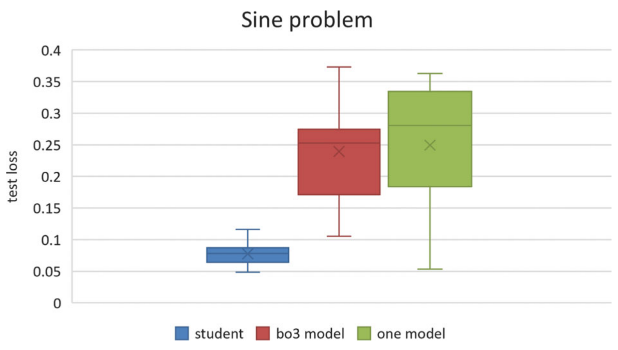

Fig. 9 Box plot of testing losses of 50 experiments on the sine problem. The vertical line inside the box represents the median, and the cross presents the mean

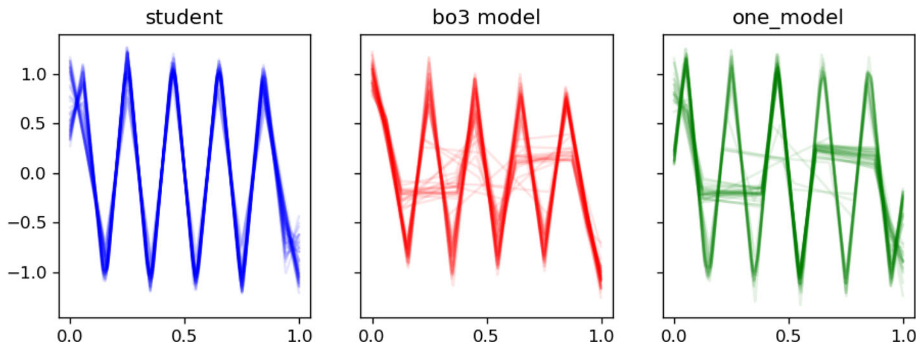

Fig. 10 Plots of learned sine curves by models trained with different strategies. We plot every train result as one line and overlay on top of each other. As we can see, all the students resulting from merging get all the peaks. However, models without merging often missed some peaks

To verify, we have created a synthetic dataset—five sine waves with noise. The input is scalar x and the target is $y = sin(10\pi x) + z$ where $z \sim \mathcal{N}(0,\ 0.2)$, see Fig. 8.

Our architecture is composed of two linear layers (i. e. one hidden layer) with 100 hidden neurons (Fig. 8). In every strategy, we have used 900 epochs and SGD with starting learning rate 0.01 and momentum 0.9. Then, we have decreased the learning rate to 0.001 after the 100th epoch for student fine-tuning, the 250th epoch for teachers and bo3 models, and the 800th epoch for a model in *one model*. We repeat all experiments 50 times.

We can observe that our strategy has significantly smaller error than other strategies (Table 1, Fig. 9).

Digging deeper (Fig. 10), we observe networks trained by our strategy to predict all the peaks correctly. However, networks trained by generic strategy often miss some peaks. This indicates that our training strategy helps the network to select better features for later use. In some cases, bo1 network is lucky and predicts all the peaks correctly, which can be seen in having similar minimal error as our training strategy.

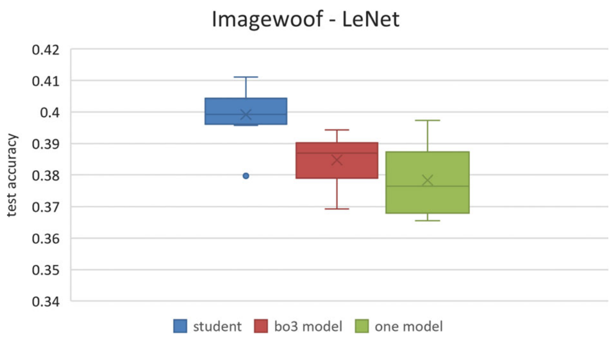

Fig. 11 Box plot of the testing accuracies of 10 experiments on Imagewoof with LeNet

Table 2 Summary of experimental results on image classification tasks

Task	Strategy	Min	Max	Median	Mean	Std
Imagewoof LeNet test accuracy [%]	Student	**38.0**	**41.1**	**39.9**	**39.9**	0.8
	bo3 model	36.9	39.4	38.7	38.5	**0.7**
	one model	36.5	39.7	37.7	37.8	1.0
Imagewoof ResNet18 test accuracy [%]	Student	**82.4**	**82.8**	**82.5**	**82.6**	**0.2**
	bo3 model	80.1	80.9	80.7	80.6	0.3
	one model	81.0	81.8	81.3	81.3	0.3
CIFAR-100 ResNet20 test accuracy [%]	Student	**68.8**	**69.6**	**68.8**	**69.0**	0.3
	Pruning	68.5	69.1	**68.8**	68.9	0.2
	bo3 model	67.0	67.2	67.0	67.0	**0.1**
	one model	67.0	67.9	67.5	67.5	0.4

Bold values indicate the best result (group in case of Table 2)
We report testing accuracy. Considering specific task, all strategies used the same number of epochs. Strategy *student* (our strategy) uses 2/3 to train two teachers and 1/3 to train student (1/6 finding important features, 1/6 finetuning). Strategy *bo3 model* trains three models and picks the best. Strategy *one model* uses all epochs to train one model

3.3 Image Classification

Here, we test our training strategy on various combinations of dataset and architecture. First, we use Imagewoof (Imagenet-1k using only 10 classes of dog breeds) dataset [16, 17] with LeNet [18] and ResNet18 [19] architectures. Then, we test our approach on CIFAR-100 dataset [20] using ResNet20 [19] architecture. To compare with other model fusion approach, we run a test on CIFAR-10 dataset using VGG-11 architecture used in [11, 13]. Finally, we evaluate our approach on Imagenet-1k dataset [21].

In all cases, our training strategy with merging provides better results than generic training strategies. Results are summarized in Tables 2 and 4 and details about the training setup are provided below.

3.3.1 Imagewoof on LeNet

LeNet is composed of two convolutional layers followed by three linear layers. The shape of an input image is (28, 28, 3). The convolutional layers have 6 and 16 output channels,

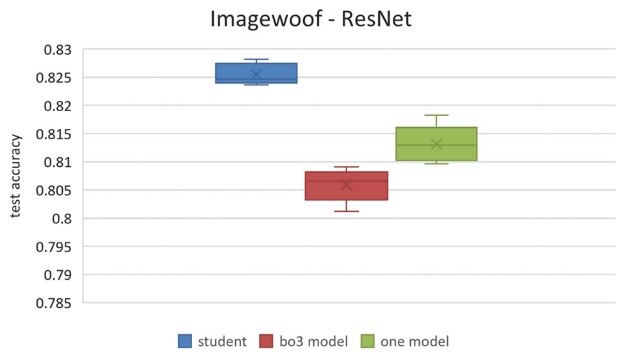

Fig. 12 Results of 5 experiments on Imagewoof with ResNet18. The worst student (0.819) had slightly better accuracy than the best long teacher (0.818)

respectively. The linear layers have 400, 120, and 80 input features, respectively. For the architecture of the big student, see Fig. 6.

Every strategy has used 6000 epochs cumulatively and SGD with starting learning rate 0.01 and momentum 0.9. Every training except finding important neurons (teachers, student fine-tuning, bo3 models, and one model) decreased the learning to 0.001 in the third quarter and 0.0001 in the last quarter of the training.

We have conducted 10 experiments, see Fig. 11 for visualisation and Table 2 for detailed statistics. Our strategy has consistently better results than other strategies. It has a greater sample variance than *one model* due to an outlier, see Fig. 11.

3.3.2 Imagewoof on ResNet18

ResNet has two information flows (one through blocks, one through skip connections). Throughout the computation, its update is $x = f(x) + x$, instead of the original $x = f(x)$. To conserve this property, some gate layers have to be synchronized—share weights and realization of random variables, see Fig. 6.

Every strategy has used 600 epochs cumulatively. The optimizer and the learning rate scheduler is analogical to the LeNet experiment.

We have conducted 5 experiments, see Fig. 12 for visualisation and Table 2 for detailed statistics. Similarly, as with LeNet, our strategy has consistently better results than other strategies.

3.3.3 CIFAR-100 on ResNet20

We also tested our approach on the CIFAR-100 dataset using ResNet20. Our total training budget is 600 epochs. We optimize models using SGD with starting learning rate 0.1 and then divide it by 10 during half and three quarters of the training of one network. We run all strategies 5 times and report results in Table 2. We can see that our strategy is more than 1% better than training one model for an extended period of time. Our strategy is also competitive with typical channel pruning, but compared to channel pruning it can reuse already trained networks of target size.

Table 3 Comparison with model fusion algorithms: OTFusion from [11] and GAMF from [13] on CIFAR-10 dataset using VGG-11

	Individual models	OTFusion	GAMF	Student (ours)
Test accuracy [%]	90.31, 90.50	90.73	90.75	**91.13**

Bold values indicate the best result (group in case of Table 2)

Table 4 Results on ResNet-18 on Imagenet benchmark

Teacher epochs	Big student epochs	Finetuning epochs	Total epochs	Validation accuracy [%]
–	–	90	90	69.76
–	–	150	150	70.28
20	20	90	150	**70.47**

Bold values indicate the best result (group in case of Table 2)

3.3.4 Comparison with Other Model Fusion Algorithms on CIFAR-10 Using VGG-11

Model fusion presented in [11–13] can achieve the same goal as our approach. However, they first select the optimal alignment of neurons from teacher networks, and the resulting student is just an average between aligned teachers. Our approach tries to find the best subset of neurons in each layer. In this test, we compare on the CIFAR-10 dataset using VGG-11 [22] network, which is a setup used both in [11] and [13]. We took baseline networks provided by [11] and merged them using our strategy. We found important neurons during 100 epochs and finetuned the resulting student for another 100 epochs. As shown in Table 3, our approach provides more accurate final results than the model fusion approaches.

3.3.5 Imagenet on ResNet18

We tested our merging approach also on Imagenet-1k dataset [21]. However, as seen in [2] high quality training requires 300 to 600 epochs, which is quite prohibitive. We opted for the approach from Torchvision [23], which achieves decent results in 90 epochs. We train networks using SGD with starting learning rate 0.1, which decreases by factor of 10 in third and two thirds of training. For the final finetuning of student, we used a slightly smaller starting learning rate of 0.07.

For merging, we used slightly different appoach than in previous experiments. We trained teachers only for a short amount of 20 epochs, which gives teacher accuracy around 65%. Then we spend 20 epochs in tuning big student and finding important neurons, and finally finetune for 90 epochs. In the total of 150 epochs, we get better results than ordinary training for 90 epochs and also better results than training for equivalent amount of 150 epochs. Results are summarized in Table 4.

4 Conclusions and Future Work

We proposed a simple scheme for merging two neural networks trained from different initializations into one. Our scheme can be used as a finalization step after one trains multiple copies of one network with varying starting seeds. Alternatively, we can use our scheme to get higher quality networks under a similar training budget, which we experimentally demonstrated.

One of our scheme's downsides is that we need to instantiate a rather big neural network during the selection of important neurons. In the future, we would like to optimize this step to be more resource efficient. One option is to select important neurons in a layerwise fashion. Another option is to align neurons first, as in other model fusion approaches, find highly similar neurons, and run the selection step with the remaining neurons.

Other options for future research include merging more than two networks and merging networks pretrained on different datasets.

Acknowledgements This research was supported by grant 1/0538/22 from Slovak research grant agency VEGA.

Funding Open access funding provided by The Ministry of Education, Science, Research and Sport of the Slovak Republic in cooperation with Centre for Scientific and Technical Information of the Slovak Republic.

References

1. Picard D (2021) Torch. manual_seed (3407) is all you need: on the influence of random seeds in deep learning architectures for computer vision. arXiv preprint arXiv:2109.08203
2. Wightman R, Touvron H, Jégou H (2021) Resnet strikes back: an improved training procedure in timm. arXiv preprint arXiv:2110.00476
3. Molchanov P, Tyree S, Karras T, Aila T, Kautz J (2016) Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440
4. Molchanov P, Mallya A, Tyree S, Frosio I, Kautz J (2019) Importance estimation for neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 11264–11272
5. Luo J-H, Wu J, Lin W (2017) Thinet: a filter level pruning method for deep neural network compression. In: Proceedings of the IEEE International Conference on Computer Vision, pp 5058–5066
6. Yu R, Li A, Chen C-F, Lai J-H, Morariu VI, Han X, Gao M, Lin C-Y, Davis LS (2018) Nisp: pruning networks using neuron importance score propagation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 9194–9203
7. Hinton G, Vinyals O, Dean J, et al (2015) Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 **2**(7)
8. Beyer L, Zhai X, Royer A, Markeeva L, Anil R, Kolesnikov A (2021) Knowledge distillation: a good teacher is patient and consistent. arXiv preprint arXiv:2106.05237
9. Nath U, Kushagra S (2020) Better together: resnet-50 accuracy with 13x fewer parameters and at 3x speed. arXiv preprint arXiv:2006.05624
10. Chen T, Zhang Z, Cheng Y, Awadallah AH, Wang Z (2022) The principle of diversity: training stronger vision transformers calls for reducing all levels of redundancy. In: In IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2022)
11. Singh SP, Jaggi M (2020) Model fusion via optimal transport. Adv Neural Inform Process Syst 33:22045–22055
12. Wang H, Yurochkin M, Sun Y, Papailiopoulos D, Khazaeni Y (2020) Federated learning with matched averaging. arXiv preprint arXiv:2002.06440

13. Liu C, Lou C, Wang R, Xi AY, Shen L, Yan J (2022) Deep neural network fusion via graph matching with applications to model ensemble and federated learning. In: International Conference on Machine Learning, pp 13857–13869. PMLR
14. Voita E, Talbot D, Moiseev F, Sennrich R, Titov I (2019) Analyzing multi-head self-attention: specialized heads do the heavy lifting, the rest can be pruned. arXiv preprint arXiv:1905.09418
15. Louizos C, Welling M, Kingma DP (2017) Learning sparse neural networks through l_0 regularization. arXiv preprint arXiv:1712.01312
16. Imagenette HJ. https://github.com/fastai/imagenette
17. Shleifer S, Prokop E (2019) Using small proxy datasets to accelerate hyperparameter search. arXiv preprint arXiv:1906.04887
18. LeCun Y, Boser B, Denker J, Henderson D, Howard R, Hubbard W, Jackel L (1989) Handwritten digit recognition with a back-propagation network. Adv Neural Inform Process Syst 2
19. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 770–778
20. Krizhevsky A, Hinton G, et al (2009) Learning multiple layers of features from tiny images
21. Deng J (2009) A large-scale hierarchical image database. In: Proc of IEEE Computer Vision and Pattern Recognition
22. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556
23. Torchvision. https://pytorch.org/vision/stable/index.html