

MASTER'S THESIS

Partial Model Merging

submitted by

Korbinian Koch

Universität Hamburg

MIN Faculty

Department of Computer Science

Course of studies: Intelligent Adaptive Systems (M. Sc.)

Matriculation number: 6928742

First reviewer: Prof. Dr. Stefan Wermter

Second reviewer: Dr. Sven Magg

Licence



This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this licence, visit <https://creativecommons.org/licenses/by-sa/4.0/>.

Acknowledgements

I am grateful to Samuel K. Ainsworth, Keller Jordan, and Jason Yosinski for insightful discussions and feedback throughout the creation of this thesis. Thank you to Rosanne Liu for the opportunity to present my work at ML Collective and the generous support from them.

This thesis was supported by a GCP compute grant from Google, provided through ML Collective.

Abstract

It is commonly known that ensembling two models trained on the same task often outperforms each model individually, while doubling the inference cost. A novel stream of research on model merging explores whether it is possible to combine multiple models into one by interpolating their weights, and could serve as a more efficient alternative to ensembles.

It is already viable to interpolate models fine-tuned from a shared base model with the same performance increase as when they are ensembled. However, the interpolation of models that were trained independently or on different datasets still poses a significant challenge, and results in merged models with a higher loss and lower accuracy that is only avoidable when the models are prohibitively wide.

In this thesis we postulate that model merging and ensembling represent two extremes of a spectrum by either enforcing a complete overlap or keeping all parameters separate. We evaluate methods that are situated in between, overlapping and interpolating the parameters of both models just in parts, while keeping the remaining parameters unchanged. We demonstrate the ability of such partial model merging to gracefully eliminate any existing barriers, with loss and accuracy approaching those of ensembling as the width increase approaches 100%. We observe this gradual improvement across all architectures and model dimensions, even if the endpoints were trained on disjoint or biased datasets.

We also show that specific layers of the models have a significantly higher contribution towards the occurring barriers, and that reducing their overlap first allows us to reduce the barriers quicker. Contrary to previous beliefs these layers cannot be identified just by looking at the average correlations between units.

Using a simple baseline method inspired by our findings, we are able to achieve zero-barrier connectivity with respect to both test loss and test accuracy between two regular-width VGG11s independently trained on CIFAR10 and SVHN with a parameter increase of just 22% and 24% respectively, and without any additional training after merging. Similar interpolation-based connectivity on independently trained VGGs has never been achieved experimentally using full merging, even for high width multipliers.

Contents

1	Introduction	1
1.1	Research objective	2
2	Background	4
2.1	Weight averaging	4
2.1.1	Unpermuted weight averaging	5
2.2	Permutation symmetry	6
2.2.1	Permutation symmetry in MLPs	6
2.2.2	Permutation symmetry in CNNs	9
2.2.3	Permutation symmetry in ResNets	11
2.3	Mode connectivity	13
2.4	Neuron alignment	16
2.4.1	Activation matching	16
2.4.2	Weight matching	17
2.4.3	Other matching techniques	17
2.5	Variance collapse	18
2.6	Federated learning	19
3	A method for partial merging	23
3.1	Good and bad matches	23
3.2	Relaxing the linear assignment problem	25
3.3	Constructing the merged model	27
4	Experiments	30
4.1	Baselines	30
4.2	Partial merging with forced buffer assignment	36
4.3	Does the choice of units even matter?	42

Contents

4.4	Parameter efficiency	44
4.5	Partial merging with adaptive buffer assignment	46
4.6	Split data training	51
4.7	Finetuned models	54
4.8	Partially merging single layers	56
5	Discussion	60
6	Conclusion	63
	Appendix A Additional tables and plots	64
	Appendix B Weight matrix partitioning	76
	Bibliography	79
	Eidesstattliche Versicherung	88
	Erklärung zu Bibliothek	90

1 Introduction

The history of machine learning is replete with methods aiming to improve the generalization ability of models. One such method that has stood the test of time exceptionally well is ensembling. It dates back at least until 1979, when Dasarathy and Sheela [1] proposed a composite classifier system consisting of two or more classifiers that partition the feature space in a divide-and-conquer manner. This basic idea of using the predictions of multiple classifiers at once would become a cornerstone of classical machine learning techniques for the following decades. This status of ensemble methods was cemented in the 1990s by the success of bagging and boosting. While today ensembling is still taught in practically every undergraduate machine learning course, it is often viewed as old-fashioned and impractical by the generation of researchers that grew up with large neural networks being the de-facto standard for learning from ever-growing piles of data.

We argue that this notion could not be further from the truth, and that ensembling remains an indispensable tool even in the age of billion-parameter neural nets. We want to support this claim by taking a closer look at the recently released Mixtral 8x7B model [2]. Mixtral 8x7B is a so-called sparse mixture of experts (SMoE), which in this case means that a purposefully trained router network chooses two from a set of eight distinct groups of parameters in each feedforward block [2], effectively turning the model into an ensemble with two constituent models at inference time. At the time of writing, Mixtral 8x7B represents the best language model overall regarding cost/performance trade-offs [2], and it is rumored that the ensembling of experts is also part of the “secret sauce” that powers the unsurpassed performance of OpenAI’s GPT-4.

While ensembles come with more benefits than just improved accuracy, namely including robustness [3], protection against membership inference attacks [4] and

1 Introduction

improved machine unlearning [5], they come with the notable downside of significantly higher inference costs. This downside is especially costly if the number of inference steps is high, as is the case for the transformer-based language models of today.

The question of whether it is possible to combine the parameters of multiple models into a single one is addressed by a rapidly growing number of papers on model merging, also known as model fusion. Originating in theoretical research on minima, saddle points and permutation symmetries in the loss landscape, model merging has become an increasingly useful tool. At the time of writing, model merging is able to improve the accuracy of models fine-tuned on the same objective [6] and interpolate two models trained on the same objective from different random seeds without a drop in performance, given that both models are sufficiently wide [7]. However, despite incremental advancements that improved the resulting performance of the merged model, model merging is yet unable to combine models of arbitrary size without a drop in accuracy and loss.

This leads to a compelling question that is at the heart of this thesis: If model merging tends to degrade performance, but ensembling, despite its effectiveness, is resource-heavy, is there a middle ground?

1.1 Research objective

“If two neural networks had a child, what would be its weights?” [8]

Model merging fuses the parameters of two parent models of an identical architecture and size into a child model with the exact same depth and width. This requires each unit of the first parent model to be matched with a unit of the second parent model, no matter how well they are aligned or whether they learned the same concept. In contrast, the units of two ensembled models remain separate even during inference, and are influenced only by the features that they learned to represent. If some of the features of the parent models correspond well to each other, but others are wholly incompatible, can we merge only those that match?

1 Introduction

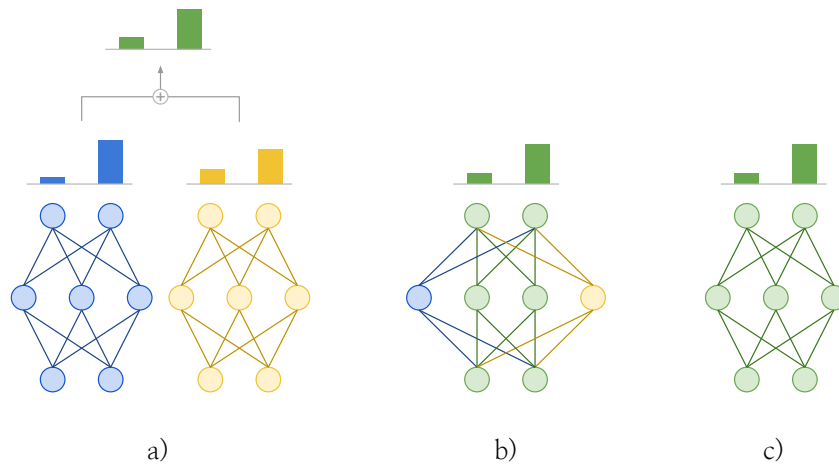


Figure 1.1: Visualization of a) ensembling, b) partial merging (66% overlap/33% width increase), and d) conventional merging (100% overlap).

We can achieve this partial merging of units by “pulling apart” the units of both parent models to some degrees, resulting in the models to overlap *just in parts*. If a partial overlap – visualized in Figure 1.1 – enables models to have better unit alignment and preserve the integrity of dissimilar features in the non-overlapping parts, can we reap performance benefits approaching those of ensembling for a significantly cheaper cost? With this thought in mind, we define our research question as follows:

Research Question

Can partial model merging^a provide model performance^b approaching that of ensembling^c while avoiding the need for expensive width multipliers faced by existing model merging approaches?

^arequiring the merged models to overlap just in parts

^bmeasured by test loss and test accuracy

^caveraging of prediction vectors

2 Background

This chapter provides an overview of the concepts and methods related to model merging and some of its applications. It draws inspiration from two main sources: First, personal conversations with the authors of some related papers, and second, a survey paper on *Deep Model Fusion* [9], which at this point only exists as a preprint. Historically, model merging has been approached from two perspectives. First, one line of research that was mainly interested in properties of the loss landscape that connects SGD solutions. This line was not primarily motivated by practical applications of model merging, but ought to explore it to develop a deeper understanding of generalization and optimizer behavior. The second line of research, which includes model soups and federated learning, explored model merging as a tool to achieve practical goals, e.g. a better generalization performance or fewer federated communication rounds. This thesis falls somewhere in between these two lines of research, and tries to bridge the gap between theoretical insights and practicability.

2.1 Weight averaging

Weight averaging describes the recombination of two or more models of the same architecture and size by averaging all their parameters – not just the weights. In its simplest form, a model average of n models is obtained by

$$\sum_{i=1}^n \alpha_i \theta_i, \quad \alpha_i = \frac{1}{n}, \quad (2.1)$$

where each constituent model θ_i has an equal contribution to the average. However, through modification of each α_i , one could also give some models more weight

than others. For the special case of two constituent models we will use the following definition:

Definition 2.1

An **interpolated model** θ_α , determined by an interpolation factor $0 < \alpha < 1$, is defined [10] as the linear combination of the parameters of two **endpoint models** θ_1 and θ_2 such that

$$\theta_\alpha = (1 - \alpha) * \theta_1 + \alpha * \theta_2 \quad (2.2)$$

While other methods of aggregation exist, such as *Fisher averaging* [11] or *Regression Mean (RegMean)* [12], this thesis will only consider linear parameter interpolation due to its simplicity and popularity in the model merging literature.

2.1.1 Unpermuted weight averaging

The linear combination of two models is not guaranteed to perform better or even equally well as the two constituent models [13]. In fact, simple parameter interpolation typically results in a drastic performance decrease in the resulting model [7], [13]. Models only benefit from being averaged when their parameters are, as Jolicoeur-Martineau et al. put it, “different enough to benefit from combining them, but similar enough to average well” [14]. But how can the similarity of parameters be achieved without permutation (see Section 2.2)? The key is to enforce model alignment through a shared training history.

The idea of using averaging for stochastic approximation dates back at least until the 1990s [15]. In the context of modern neural networks, **Stochastic Weight Averaging (SWA)** [16] is one of the simplest such methods that averages the parameters of a single model at different points along its training trajectory. It is inspired by Snapshot Ensembles (SSE) [17] and Fast Geometric Ensembling (FGE) [13], which steer a single model towards multiple different optima during training using a cyclical or constant learning rate. Several refinements of SWA have been developed since [18]–[20].

SWA and its variants approach optima one after the other. **Model Soups** [6]

2 Background

train a single model until convergence, then fine-tune a large number of model copies using different hyperparameters in parallel. Using the *greedy soup* recipe, models are ordered by their validation accuracy and only included in the final average if the inclusion increases the validation accuracy. Adaptations of model soups include Sparse Soups [21], which optimizes the method for pruning, and Diverse Weight Averaging (DiWA) [22], which is used to optimize performance under distribution shifts.

Population Parameter Averaging (PAPA) [14] is a method that merges multiple models not just once at the end of training, but multiple times throughout the training procedure. A population of models is trained using random data orderings, augmentations, and regularizations, and the weights of each model are slowly pushed towards the population average every few SGD steps [14]. In the end, the models are combined into a model soup.

2.2 Permutation symmetry

Unpermuted models can only be averaged well if they have a shared training history. Simply training two models on the same random initialization is insufficient for ensuring that they produce a useful average [23], especially if trained for a large number of epochs. It is practically impossible if they have been trained separately on different random initializations [7]. In cases like these, permuting the weights of the models can restore their ability to be averaged effectively.

2.2.1 Permutation symmetry in MLPs

Most deep neural network architectures are permutation symmetric in sub-spaces of their weight space. This property originates from the permutation invariance at the neuron level, which was first noted by Hecht-Nielsen in 1990 [24]. A function f is permutation invariant if the order of its inputs has no influence on its outputs:

2 Background

Definition 2.2

A function f is **permutation invariant** if

$$f(x_1, x_2, \dots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)}) \quad (2.3)$$

for any permutation π of the indices $\{1, 2, \dots, n\}$.

This property holds for both the activation function σ , which only takes a scalar as input, and the weighted sum z in a single neuron

$$y = \sigma(z), \quad z = \sum_{i=1}^n w_i x_i + b \quad (2.4)$$

when considering the inputs as pairs (w_i, x_i) that are permuted together. This can be extended to an L -layer multi-layer perceptron (MLP)

$$f(\mathbf{x}; \Theta) = \mathbf{z}_{L+1}, \quad \mathbf{z}_{\ell+1} = \sigma(\mathbf{W}_\ell \mathbf{z}_\ell + \mathbf{b}_\ell), \quad \mathbf{z}_1 = \mathbf{x} \quad [7]. \quad (2.5)$$

Given the parameters Θ and a set of any appropriately sized permutation matrices $\{\mathbf{P}_1, \dots, \mathbf{P}_L\}$, we can obtain a functionally equivalent MLP with

$$f(\mathbf{x}; \Theta) = \mathbf{z}_{L+1}, \quad \mathbf{z}_{\ell+1} = \sigma(\mathbf{P}_\ell \mathbf{W}_\ell \mathbf{P}_{\ell-1}^\top \mathbf{z}_\ell + \mathbf{P}_\ell \mathbf{b}_\ell), \quad \mathbf{z}_1 = \mathbf{x}, \quad (2.6)$$

$$\mathbf{P}_0, \mathbf{P}_L \in I,$$

where I denotes the set of identity matrices. Note the necessity for the zero-th permutation matrix to be the identity, as the input order is fixed. This also means that the MLP as a whole is not a permutation invariant function.

For the permutation matrices, consider the following definition:

2 Background

Definition 2.3

A square $n \times n$ matrix \mathbf{P} is a **permutation matrix** if exactly one entry in each row and column is equal to 1 and all other entries are 0 [25]. It represents a permutation π of n elements and is constructed such that its (i, j) -th entry \mathbf{P}_{ij} is given by:

$$\mathbf{P}_{ij} = \begin{cases} 1 & \text{if } \pi(i) = j \\ 0 & \text{otherwise.} \end{cases} \quad (2.7)$$

To permute the rows of a matrix \mathbf{A} with dimensions $m \times n$, the necessary permutation matrix \mathbf{P} needs dimensions $m \times m$:

$$\mathbf{P}^{m \times m} \mathbf{A}^{m \times n} = \mathbf{A}'^{m \times n} \quad (2.8)$$

To permute the columns of \mathbf{A} the permutation matrix needs dimensions $n \times n$:

$$\mathbf{A}^{m \times n} \mathbf{P}^{n \times n} = \mathbf{A}'^{m \times n} \quad (2.9)$$

In the context of Equation 2.6, this implies that if weight matrix \mathbf{W}_ℓ has shape $m \times n$, then “appropriately sized” means \mathbf{P}_ℓ needs shape $m \times m$ and $\mathbf{P}_{\ell-1}$ needs shape $n \times n$. Each permutation matrix \mathbf{P}_ℓ thus represents a permutation of the m neurons in layer ℓ .

Compared to permutation invariance, permutation symmetry is a more general concept that applies to all MLPs with at least one hidden layer containing more than one neuron:

2 Background

Definition 2.4

A function $F(\mathbf{x}; \Theta)$ exhibits **permutation symmetry** if there exists at least one permutation π acting on Θ such that

$$F(\mathbf{x}; \Theta) = F(\mathbf{x}; \pi(\Theta)) \quad (2.10)$$

for any input \mathbf{x} and parameters Θ , where $\pi(\Theta) = \{\theta_{\pi(1)}, \theta_{\pi(2)}, \dots, \theta_{\pi(n)}\}$.

While the terms permutation invariance and permutation symmetry are sometimes used interchangeably in the literature, we will stick to the definitions provided in this section.

2.2.2 Permutation symmetry in CNNs

While the permutations in an MLP act at the neuron level, the permutations in the feature extractor of a CNN act on the kernel level. We can view the feature map computation in layer ℓ of a CNN as

$$\begin{aligned} f(\mathbf{X}; \Theta) &= \mathbf{O}_{L+1}, \quad \mathbf{O}_1 = \mathbf{X}, \\ \mathbf{O}_{\ell+1}^{i,j,k} &= \sigma \left(\sum_{m,n,k'} \mathbf{O}_{\ell}^{i+m,j+n,k'} \cdot \mathbf{W}_{\ell}^{k,k',m,n} + \mathbf{b}_{\ell}^k \right), \end{aligned} \quad (2.11)$$

2 Background

- $\mathbf{O}_{\ell+1}^{i,j,k}$: the value at the (i, j) -th position of the k -th input feature map in layer $\ell + 1$,
- $\mathbf{O}_{\ell}^{i+m,j+n,k'}$: the value at the $(i + m, j + n)$ -th position of the k' -th input feature map from layer ℓ ,
- $\mathbf{W}_{\ell}^{k,k',m,n}$: the value at the (m, n) -th position of the kernel connecting the k' -th input feature map to the k -th output feature map in layer ℓ ,
- \mathbf{b}_{ℓ}^k : the bias term for the k -th output feature map in layer ℓ ,
- σ : the activation function,
- \mathbf{X} : the input image.

This definition ignores details such as padding, stride, dilation, or pooling.

Given L such convolutional layers and a set of L appropriately sized permutation matrices $\{\mathbf{P}_1, \dots, \mathbf{P}_L\}$ with matching $\{\pi_1(\cdot), \dots, \pi_L(\cdot)\}$, we can construct a functionally equivalent feature extractor with

$$\begin{aligned}
 f(\mathbf{X}; \Theta) &= \hat{\mathbf{O}}_{L+1}, \quad \hat{\mathbf{O}}_{L+1}^{i,j,k} = \mathbf{O}_{L+1}^{i,j,\pi_L^{-1}(k)}, \quad \mathbf{O}_1 = \mathbf{X}, \quad \mathbf{P}_0 \in I, \\
 \mathbf{O}_{\ell+1}^{i,j,k} &= \sigma \left(\sum_{m,n,k'} \mathbf{O}_{\ell}^{i+m,j+n,\pi_{\ell-1}^{-1}(k')} \cdot \mathbf{W}_{\ell}^{\pi_{\ell}(k),\pi_{\ell-1}^{-1}(k'),m,n} + \mathbf{b}_{\ell}^{\pi_{\ell}(k)} \right), \quad (2.12)
 \end{aligned}$$

where $\pi_{\ell}^{-1}(\cdot)$ represents the permutation belonging to \mathbf{P}_{ℓ}^{\top} that undoes $\pi_{\ell}(\cdot)$. When looking just at the parameters Θ , implementing this modification consists of applying permutation \mathbf{P}_{ℓ} to dimension 1 of W_{ℓ} , applying permutation $\mathbf{P}_{\ell-1}^{\top}$ to dimension 2 of W_{ℓ} , and setting $\mathbf{b}_{\ell} = \mathbf{P}_{\ell} \mathbf{b}_{\ell}$. In practice, we would not use the equivalent output $\hat{\mathbf{O}}_{L+1}$, but the permuted \mathbf{O}_{L+1} , as the last permutation $\pi_L(\cdot)$ will be undone in the attached fully-connected classifier that follows the MLP logic already established in Equation 2.6, where we set $\mathbf{P}_0^{MLP} = \mathbf{P}_L^{CNN}$. As pooling functions such as max pooling and average pooling are permutation invariant, they are not influenced by the used kernel permutations.

2.2.3 Permutation symmetry in ResNets

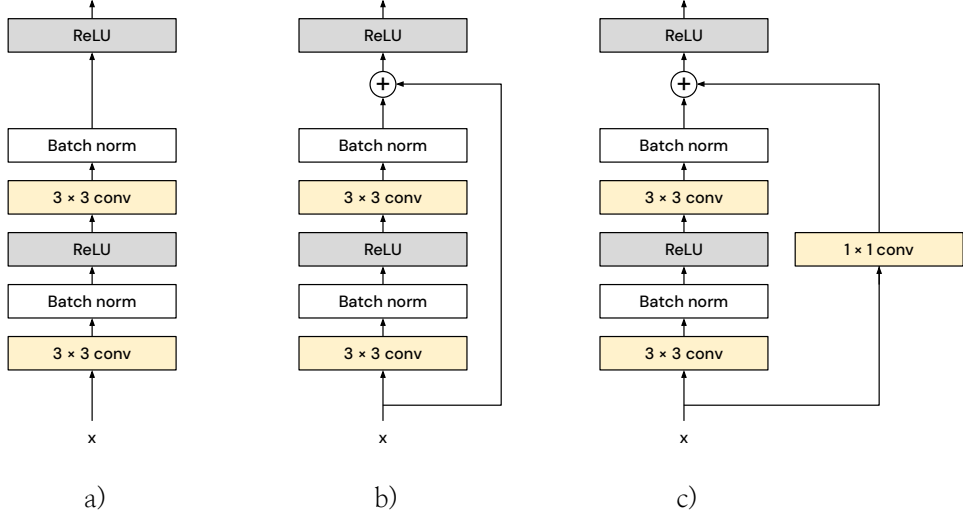


Figure 2.1: **a)** regular CNN layers **b)** a ResNet block without downsampling
c) a ResNet block with pointwise convolution for downsampling

Compared to simple CNN architectures like VGG [26], ResNets [27] contain *residual connections* (also known as *skip connections* or *shortcut connections*) that make it easier for the model to learn the identity mapping. The residual connections add the feature maps of a previous layer to the newly computed feature maps in the current layer (see Figure 2.1b compared to 2.1a). This demands that the dimensionality of feature maps stay the same in each layer, which in return requires the number of kernels in each layer to be identical. When the depth of the feature maps is to be increased, the residual connection must include a point-wise convolution (see Figure 2.1c) to increase the dimensionality correspondingly. This operation is typically combined with a downsampling of the spatial resolution in both streams.

Consider a general skip connection¹ with

$$\mathbf{X}_{\ell+1} = \sigma(\mathbf{W}_{\ell} \sigma(\mathbf{X}_{\ell-1} \mathbf{X}_{\ell-1}) + \mathbf{X}_{\ell-1}). \quad (2.13)$$

¹We ignore batch norm layers and bias terms in this notation, which have to be permuted too. Finding a permutation for them is trivial if the permutation of weights is known.

2 Background

Due to the addition, $\mathbf{X}_{\ell+1}$ and $\mathbf{X}_{\ell-1}$ must share the same indexing of their units. Attempting to permute the hidden units in $\mathbf{X}_{\ell-1}$ without simultaneously permuting the units in $\mathbf{X}_{\ell+1}$ would disrupt the identity mapping and thus break functional equivalence [28].

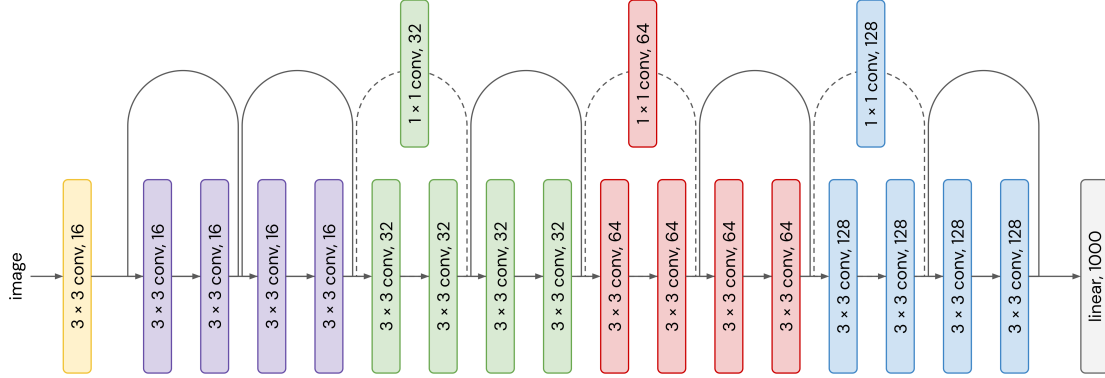


Figure 2.2: The architecture of a ResNet18 with a first-layer width of 16.

ResNets typically consist of an initial convolutional layer and then several stages, indicated in Figure 2.2 with the colors purple, green, red, and blue. Each stage consists of several concatenated residual blocks like in Figure 2.1b, while the first block in each stage (except in stage one) contains a point-wise convolution in the residual connection (Figure 2.1c) and performs a downsampling operation. Consider our ResNet to be defined analogous to Equation 2.11, but including the residual additions. Our ResNet has $L - 1$ convolutional layers organized in S stages (except layer 1). We can then obtain a functionally equivalent ResNet using L permutation matrices $\{\mathbf{P}_1, \dots, \mathbf{P}_L\}$ representing the permutations on the regular layers, and $S - 1$ permutation matrices $\{\hat{\mathbf{P}}_2, \dots, \hat{\mathbf{P}}_S\}$ representing the permutations on the point-wise convolutions between stages. The permutations can be applied analogous to Equation 2.12, but must satisfy the following constraints, where $s(\ell)$ yields the stage to which layer ℓ belongs, or 0 for layer 1 and $S + 1$ for layer L :

$$\begin{aligned} \forall i \in \{2, \dots, L - 1\}: i \bmod 2 = 0 \wedge ((\exists j \in \{2, \dots, L - 1\}: j < i \\ \wedge s(j) = s(i)) \vee s(i) = 1) \Rightarrow \mathbf{P}_{i+1} = \mathbf{P}_i^\top \end{aligned} \quad (2.14)$$

“All odd layer permutations must undo the permutation of the immediately preceding even layer, except when a downsampling operation is used.”

2 Background

$$\forall i \in \{2, \dots, L-1\}: s(i) \neq s(i-1) \wedge s(i) > 1 \Rightarrow \mathbf{P}_{i+1}\mathbf{P}_i = \hat{\mathbf{P}}_{s(i)}$$

“When a point-wise convolution is used, the product of the permutations of the two skipped layers must be identical to the permutation of the point-wise convolution layer.” (2.15)

The influence of skip connections on permutation symmetry has been explored by Orhan and Pitkow [29], who argue that skip connections speed up convergence by breaking specific permutation symmetries and thus eliminating overlap singularities. A first detailed description of which permutations yield a functionally equivalent ResNet model was given in [28].

2.3 Mode connectivity

The optimization of neural networks is often understood as the approach of a strictly convex and thus isolated valley of low loss in the network’s loss surface. This understanding is challenged by research on **mode connectivity**, which asks to which extent different optima in a network’s loss surface are connected via paths of nearly constant loss.

Draxler et al. [30] were the first to conjecture that optima are not distinct valleys, but rather points on a connected manifold of low loss. They provide a method to find paths in weight space (polygonal chains with multiple pivot points) that connect arbitrary optima obtained with different random initializations with nearly constant loss (see Figure 2.3). Independently, Garipov et al. [13] demonstrated the existence of much simpler polygonal chains with just one bend and quadratic Bézier curves that connect the optima.

The baseline against which these paths compete is a straight interpolation path, which reflects model interpolation as defined in Definition 2.1. Interpolation along this path usually yields models with worse performance, and thus an increase in loss, which determines the so-called *loss barrier* (see Definition 2.5), sometimes also called *energy barrier*. The definition typically refers to the test loss.

2 Background

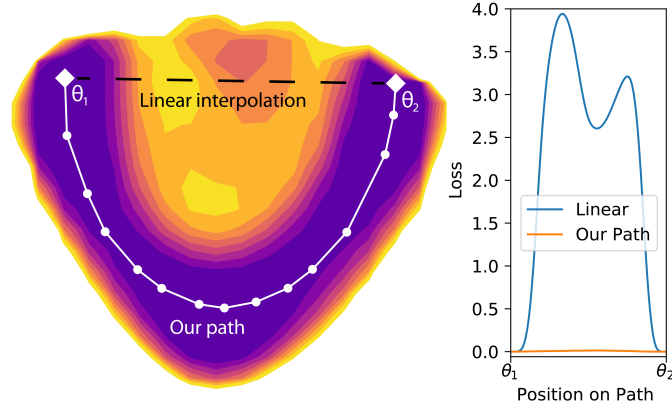


Figure 2.3: **Left:** A polygonal chain with 13 pivot points connecting two optima. The high-dimensional parameter space has been projected onto two dimensions, with the heatmap indicating areas of high and low loss. **Right:** The corresponding losses along the linear interpolation and the polygonal path. [30]

Definition 2.5

The **loss barrier** $B(\theta_1, \theta_2)$ between a pair of models is defined [31] as the maximum increase in loss incurred along the linear interpolation path in parameter space between θ_1 and θ_2 , relative to the corresponding linear interpolation of the two endpoint losses:

$$B(\theta_1, \theta_2) = \sup_{\alpha \in [0,1]} [\mathcal{L}((1 - \alpha)\theta_1 + \alpha\theta_2)] - [(1 - \alpha)\mathcal{L}(\theta_1) + \alpha\mathcal{L}(\theta_2)] \quad (2.16)$$

Mode connectivity has been extensively discussed in the literature, but never precisely defined. For Garipov et al. [13] it suffices that metrics such as train loss or test error “remain low” along the path, while Tatro et al. [28] demand the loss stays “nearly constant”, not considering cases in which endpoints have meaningfully different losses. For the purpose of this thesis, we will utilize Definition 2.6, which aims to unify the definitions from the literature. Please note that it allows two models to have a non-zero loss barrier, even if they are linearly mode connected.

2 Background

Definition 2.6

Two models θ_1, θ_2 are **mode connected** if there exists a path in weight space connecting θ_1 to θ_2 , along which the loss is never greater than $\max(\mathcal{L}(\theta_1), \mathcal{L}(\theta_2))$. They are **linearly mode connected** if this path is a straight line.

Entezari et al. [31] conjectured that linear mode connectivity can be established between almost all models θ_1, θ_2 , by applying a function-preserving permutation $\pi(\cdot)$ on one of them. Their conjecture, verbatim, is provided here as Conjecture 2.1, albeit with a slightly different notation.

Conjecture 2.1

Let $f(\theta)$ be the function representing a feedforward network with parameters $\theta \in \mathbb{R}^k$. Let P be the set of all valid permutations for the network. Let $P : \mathbb{R}^k \times P \rightarrow \mathbb{R}^k$ be the function that applies a given permutation to parameters and returns the permuted version. Let $B(\cdot, \cdot)$ be the function that returns [the] barrier value between two solutions as defined in Equation 1 [this thesis: Definition 2.5]. Then, there exists a width $h > 0$ such that for any network $f(\theta)$ of width at least h the following holds: There exists a set of solutions $\mathcal{S} \subseteq \mathbb{R}^k$ and a function $Q : \mathcal{S} \rightarrow P$ such that for any $\theta_1, \theta_2 \in \mathcal{S}$, $B(P(\theta_1, Q(\theta_1)), \theta_2) \approx 0$ and with high probability over an SGD solution θ , we have $\theta \in \mathcal{S}$ [31].

Note that their conjecture states that linear mode connectivity only sets in above some width h . In their experiments, they show that linear mode connectivity is approached quicker and therefore at smaller widths for simpler tasks like MNIST, and more slowly for more complex tasks like CIFAR10. Interestingly, they also demonstrate that even the loss barrier $B(\theta, \pi(\theta))$ between a model θ and a permutation of itself becomes smaller as the width increases, and specifically at the same rate as the barrier $B(\theta_1, \theta_2)$ between two random models.

If their conjecture holds, the obvious question is how to find permutations that enable linear mode connectivity, a challenge we will discuss in the next section.

2.4 Neuron alignment

The action of permuting a model θ_1 so that it can be usefully interpolated with a model θ_2 is called **neuron alignment**. The success of a neuron alignment strategy manifests itself in the establishment of linear mode connectivity – or the lack thereof. Generally speaking, neuron alignment methods aim to minimize the test loss barrier $B(\theta_1, \theta_2)$, but depending on the task it can also be useful to measure the utility of the method via the test accuracy of interpolated models θ_α .

Broadly speaking, most neuron alignment methods consist of two parts: First, a measure of similarity between permutable units (neurons, kernels), and second, a procedure for determining the optimal permutation based on this measure.

2.4.1 Activation matching

Li et al. [28] were the first to study whether independently trained networks learn the same features by measuring the correlation between their activations. This analysis is based on the notion that neural network units that have learned the same feature get activated by the same inputs, following the Hebbian mantra² “neurons that fire together, wire together” [32]. As collecting activations requires inputs, activation matching requires access to plausible input data distribution, despite requiring no labels.

While the similarity of activations can be measured by simple matrix multiplication, as done in [7], a more commonly used [10], [33] approach is to use the only minimally more compute-intensive, yet more easily interpretable correlation coefficient. In any given layer ℓ , we are then looking for a permutation π_ℓ^* of model θ_2 that minimizes the following equation:

$$\pi_\ell^* = \arg \min_{\pi_\ell} \left(\sum_i \text{corr} \left(X_{\ell,i}^{(1)}, X_{\ell,\pi_\ell(i)}^{(2)} \right) \right), \quad (2.17)$$

where $X_{\ell,i}^{(n)}$ are the activations of the i -th hidden unit in layer ℓ of model θ_n [10].

²This analogy was borrowed from [7].

2 Background

Finding this permutation is straightforward, as the problem formulation in Equation 2.17 amounts to a linear assignment problem (LAP; sometimes also called linear sum assignment problem or LSAP) [10], which can be solved using the Hungarian algorithm [34] in polynomial time.

2.4.2 Weight matching

The simplest way to determine the similarity between two neurons is to look at their weights. If their weights are the same, they must calculate the same thing, and the more different they are, the more different their learned feature. For the parameters of model θ_1 and θ_2 this means minimizing their squared distance by finding the optimal permutation π^* of model θ_2 :

$$\pi^* = \arg \min_{\pi} (\|\text{vec}(\theta_A) - \text{vec}(\pi(\theta_B))\|^2) \quad (2.18)$$

However, as any single permutation affects rows and columns in different weight matrices (see Equation 2.6), this minimization is not straightforward. For this problem, Ainsworth et al. [7] coin the term *sum of a bilinear assignments problem* (SOBLAP) and determine it to be NP-hard. They approximate a solution through a coordinate descent algorithm called permutation coordinate descent. It repeatedly solves weight matching for just a single randomly selected layer in the network (this constitutes a LAP) until convergence [7]. No input data is needed for weight matching, which makes it well-suited for federated learning use cases.

2.4.3 Other matching techniques

Git Re-Basin [7] proposes a third method for obtaining a permutation using a straight-through estimator. They determine the permutation by training a surrogate model $\tilde{\theta}_b$ which is projected onto θ_b . Due to the necessary training, the straight-through estimator method requires not just input data, but also labels. Singh and Jaggi [8] propose a method that aligns neurons using optimal transport and Wasserstein barycenters. Guerrero Peña et al. [35] propose a differentiable

based method on the Sinkhorn algorithm that learns soft permutation matrices which are then discretized for the final permutation.

2.5 Variance collapse

Especially for deeper models, model merging produces a phenomenon called **variance collapse** [10]. To understand variance collapse, let’s look at the following toy example: Imagine two six-sided dice D_1 and D_2 , each yielding random variables d_1 and d_2 . The distribution of d_1 and d_2 is uniform over $\{1, 2, 3, 4, 5, 6\}$, with a mean of $\mu_1 = \mu_2 = 3.5$ and a standard deviation $\sigma_1 = \sigma_2 \approx 1.7$. Imagine we have a magical knob with which we can make the two dice dependent. The knob represents the correlation coefficient ρ between d_1 and d_2 and ranges from 0 (the dice are independent, like normal dice) to 1 (d_1 and d_2 are perfectly correlated and always take on a random but identical value). If both dice are perfectly correlated, the distribution of their average $\frac{d_1+d_2}{2}$ remains the same as each dice individually. However, as the correlation coefficient shrinks, the mean of the average will stay the same while the standard deviation will get smaller. Finally, when both dice are completely uncorrelated, the standard deviation takes on $\sigma_{avg} = \frac{\sigma_1+\sigma_2}{\sqrt{2}} \approx 1.2$. Generally, and depending on ρ , the standard deviation of the average will be $\sigma_{avg} = \frac{\sigma_1^2+\sigma_2^2+2\rho\sigma_1\sigma_2}{2}$ and lie in between.

In the context of model merging, imagine D_1 and D_2 as two units in our models θ_1 and θ_2 that produce activations d_1 and d_2 . In order to function well, each of the units in the next layer expects a specific learned distribution (μ_n, σ_n) from each of their inputs. If the activations of D_1 and D_2 are not perfectly correlated but their weights are averaged nonetheless, the new activations of D_{avg} will have a smaller standard deviation than before. This decrease in variance compounds with model depth, as the input activations for any layer $\ell + 1$ now already have a lower variance, which further diminishes with each additional interpolation. Finally, the activations in the output layer become nearly constant and are not able to distinguish between inputs anymore [10].

While the phenomenon that merged model performance decreases with an increase in model depth has been described earlier [7], [13] Jordan et al. [10] were the first to point out variance collapse as the underlying reason and proposed REPAIR as a solution. Given two endpoint models θ_1, θ_2 and an interpolated

2 Background

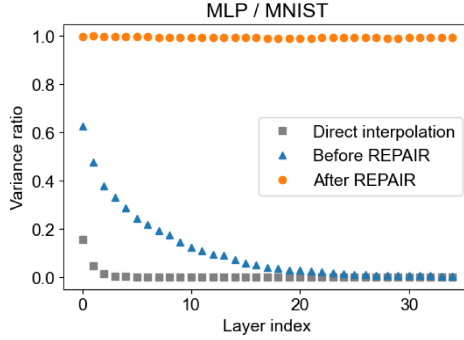


Figure 2.4: The ratio between average endpoint model variance and interpolated model variance per layer in a 35-layer MLP, using naive interpolation, permuted interpolation, and permuted interpolation with REPAIR; taken from [30].

model θ_α , REPAIR modifies the weights and biases of θ_α in such a way that for each unit X_α in θ_α $\mu_\alpha = (1 - \alpha)\mu_1 + \alpha\mu_2$ and $\sigma_\alpha = (1 - \alpha)\sigma_1 + \alpha\sigma_2$. They propose two variants of adjusting the weights: First, a data-free one that simply approximates the necessary rescaling parameters using the correlation coefficient ρ between X_1 and X_2 . This method is only able to approximate the true variance collapse beyond the first layer. Second, a data-driven method that records the true parameters μ, σ for each unit X_1 and X_2 in the endpoint models, sets these values as desired affine weights in temporarily added batch normalization layers and trains the batch normalization statistics using a plausible input data distribution. The learned running mean and running variance are then fused into the preceding weights and biases using batch normalization layer fusion [36]. The resulting activation patterns can be seen in Figure 2.4, demonstrating a full recovery of activation variance even in the final layers. While applying REPAIR is usually not able to establish linear mode connectivity that was not there before, it dramatically reduces the loss barrier of interpolated models, typically between 70 and 90 percent [10].

2.6 Federated learning

Given the previous background on model merging techniques, we will now take a closer look at its application in federated learning. Please note that federated

2 Background

learning is just one of many possible applications of model merging, and as such has been selected only as an example. It is, however, especially suited as such due to its approximately parallel development with the aforementioned merging techniques. Additionally, it is a use case in which the use of ensembling is prohibitive, and that would therefore benefit substantially from better merging methods.

The term **federated learning** first appeared in 2016 [37] and describes methods that allow a model to be trained across multiple decentralized clients, each processing a different subset of training data. It is useful in scenarios where training data cannot be pooled in a centralized location (e.g. for privacy reasons) and allows for more efficient training on large datasets by parallelizing the computational effort across multiple devices. The first method introduced alongside the term federated learning is called **Federated Averaging (FedAvg)** [37] and consists of repeatedly training a model on one or more batches of data by each client, collecting the weights of each client, averaging the weights into a central model, and broadcasting the new set of weights to each client. Alternatively, instead of averaging the weights, the gradients of each client can also be averaged and applied to the central model, yielding an equivalent result. If clients train for a higher number of epochs before exchanging their updates, the necessary communication bandwidth becomes lower, but the found optima for each subset of data may drift wider apart, making it more difficult to average the models without a loss in performance.

Probabilistic Federated Neural Matching (PFNM) [38] modified FedAvg by identifying subsets of neurons in each client model that match with neurons in other models. The neurons are then combined into a potentially larger central model using maximum a-posteriori estimation of a generative Beta-Bernoulli process model. One of the evaluated settings was a single-communication scenario, in which “legacy” models are matched and averaged only once, without any further training, which is similar to the model merging settings outside of the federated learning domain³. In this setting, PFNM performed better than FedAvg (unper-

³In fact, the paper was rejected at ICLR 2019 in part because of this, as the conditions under which the method was evaluated were considered unrealistic in a federated learning scenario and the results were considered not strong enough.

2 Background

mented averaging), but worse than ensembling, except in one case⁴. The authors of PFNM are the first to take the permutation symmetry of neural networks into account in the context of federated learning. However, their method only works for fully connected feed-forward DNNs.

Matching neurons based on their weights in deep architectures is an NP-hard optimization problem [7], [39], which the authors of PFNM don't explicitly mention, yet solve by greedily and consecutively matching the neurons once in each layer. They are doing this using not the incoming, but the outgoing weights, starting from the last layer going forward. This is much more simplistic than the permutation coordinate descent algorithm presented in Git Re-Basin [7], in which all layers are matched multiple times until convergence. However, as PFNM executes this procedure once in each communication round, the same effect is achieved over time. We hypothesize that the convergence of PFNM could be sped up significantly by using permutation coordinate descent [7] or activation matching, the latter of which is more challenging in a federated setting.

Federated Learning with Matched Averaging (FedMA) [39] adapted PFNM to simple CNNs and LSTMs. They measure the similarity between neurons with the squared Euclidian distance of the weights and allow neurons in each layer to overlap just partially. To determine the overlap, they are using a threshold ϵ : If the cost of merging (i.e. the weight distance) is higher than ϵ , a new neuron is created in the central model instead. This procedure is used in each communication round. To ensure that the model does not become prohibitively large during training, the threshold ϵ increases over time depending on the current layer size. When evaluating CNNs of different complexity, they find that LeNet [40] (4 layers) performs much better than VGG-9 [26] (9 layers), foreshadowing the “deep-is-bad” phenomenon described in [7] and addressed in [10]. As FedMA only works for CNNs without residual connections, it is not applicable to more modern architectures such as ResNets [27], U-Nets [41], or ConvNeXts [42].

Concurrent to the creation of this thesis, a refinement of FedMA called **Federated Generalized Matched Averaging (FedGMA)** [43] was published in August 2023, which adapts FedMA to architectures with residual connections and

⁴These results align approximately with the split data training experiment in the Git Re-Basin paper [7].

2 Background

drastically limits the otherwise quickly and perpetually increasing model size that comes with using FedMA by preferably just reordering neurons instead of adding new ones.

Efficient strategies for neuron alignment and averaging are integral to the success of the mentioned federated learning methods. Unfortunately, model merging in a federated setting and a non-federated setting seem to be two independent research bubbles that are largely unaware of their mutual existence, despite trying to solve the same problem. For example, the authors of FedGMA falsely claim that their method “is the first to tackle neuron matching issues on complex network structures such as ResNet-18” [43], a problem that has been solved prior in [29], [28], and [7]. Methods such as REPAIR [10] have also not yet been applied in the federated setting, despite their high effectiveness.

3 A method for partial merging

In this section, we will describe where full model merging falls short, motivate the need for partial merging, and describe a novel alignment method for partial merging.

3.1 Good and bad matches

Any neuron alignment method, not matter whether it uses the Hungarian algorithm or another match selection strategy, will have to make compromises when deciding on an optimal permutation for merging. The narrower the model, the lower the likelihood to find a good match for any given neuron in the other endpoint model. If no good match can be found (or the only good match is already assigned to be matched with a different, even better matching sibling neuron), the neuron must be merged nonetheless. This “sacrificial” match does not only influence the neuron itself, but also the unlucky partner neuron from the other endpoint model. Because of this, the goal of the alignment process should not not only be viewed as *matching the best candidates* when permuting but also as *causing the least harm* when permuting.

To aid the understanding of this phenomenon, we can look at RGB visualizations of first-layer kernels in a VGG11 in Figure 3.1. The kernels of endpoint model B have been permuted with activation matching and the Hungarian algorithm. As the inputs to the first layer are always the same for both endpoint models, the same result would have been achieved with a single bottom-up weight matching pass instead of activation matching. An example for a good match can be seen in green, and an example for a bad, “sacrificial” match in red. While the linear interpolation of the green kernel will be visually indistinguishable from either endpoint kernel,

3 A method for partial merging

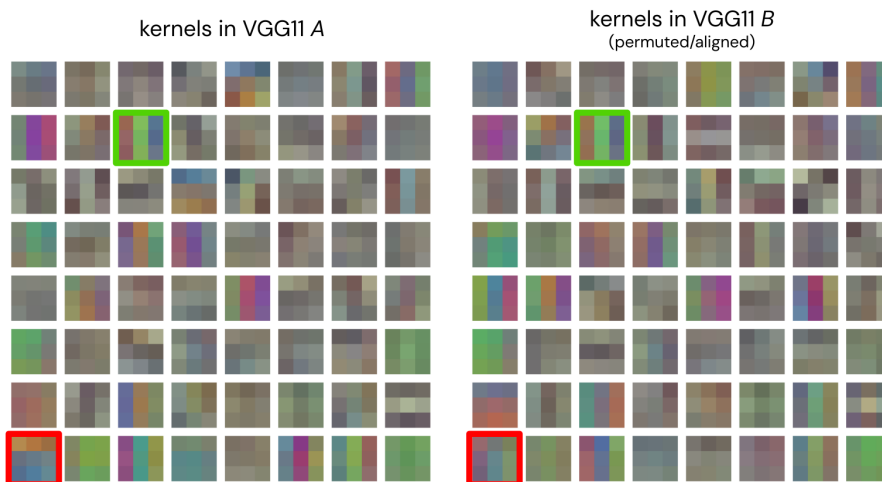


Figure 3.1: Aligned first-layer kernels of two VGG11 models trained on CIFAR10. In green: a good match. In red: a bad match. Kernel saturation increased for better visibility.

the interpolation of the red kernel will resemble neither of the endpoint kernels well and thus exhibit a different activation behavior.

When we are interested in combining the knowledge embedded into neural networks, there seem to be two promising avenues: First, executing models in parallel, as done with ensembling, which keeps all kernels as they are, or second, merging kernels from finetuned models that are sufficiently similar, as done with model soups [6]. Both methods typically yield a performance that is better than either endpoint model on its own. However, when the constituent endpoint models are trained independently instead of from the same finetuned base model, there is no sufficient similarity *across all kernels*. This distinction is very important - it's not that the models would have incompatible kernels altogether, but that *only some of the kernels* are similar, while others are incompatible.

This motivates the question whether we can use model merging specifically for those kernels that *are* compatible, while using an ensembling-like method for the incompatible kernels. This motivation is the same in principle as the one for FedMA (see Section 2.6). However, the alignment method of FedMA is suboptimal, and FedMA does also not use REPAIR to boost performance after merging. Instead, it relies on the client-side re-training to mitigate the variance collapse.

3.2 Relaxing the linear assignment problem

		model B				
		n_1	n_2	n_3	n_4	n_5
model A	n_1	.98	.93	.13	.37	.56
	n_2	.52	.09	.05	.27	.55
	n_3	.07	.20	.28	.56	.69
	n_4	.69	.85	.79	.42	.77
	n_5	.14	.51	.36	.80	.68

Figure 3.2: The LAP solution for a normal correlation matrix (full merging).

In order to understand how our proposed partial alignment and merging method is different from current methods we will look at a toy example. Consider two models A and B , where the layer we are currently aligning has 5 neurons in each model. Using activation matching or weight matching we will arrive at a matrix representing the pairwise similarities of the neurons in each model. We can then use the Hungarian algorithm to solve the linear assignment problem, as is typically done. As higher correlation coefficients are better, we let the LAP solver maximize the assignment sum instead of minimizing it¹. The result of this can be seen in Figure 3.2. Note that while the total sum of correlations is optimal, individual neurons have been matched suboptimally for the greater good. What FedMA would do now is take the worst matches below a threshold ϵ (for example $n_{2/1}$ and $n_{3/5}$ for $\epsilon = 0.7$) and split them up. This is however not optimal, as the removal of these neurons from the LAP frees up better candidates for the already existing matches.

Instead, we propose to allow the LAP solver to take the planned expansion already into account. This is done by adding virtual *buffer neurons* to the endpoint models by increasing the size of the correlation matrix by an expansion factor $\gamma \in [0, 1]$, where $\gamma = 0$ means adding no buffer at all (thus performing full merging),

¹In practice, this means multiplying our correlation matrix with -1 to get a proper cost matrix, then using the Hungarian algorithm. The used `scipy.optimize.linear_sum_assignment` function provides this functionality via an argument.

3 A method for partial merging

and $\gamma = 1$ means adding as many buffer neurons as there are neurons in the original layer. For any value in between, the resulting merged layer will have $n + \lceil \gamma n \rceil$ instead of n units (rounded to the next integer value).

We then have two options on how to fill these newly created spaces in our correlation matrix. Option one, which we name **forced buffer assignment**, sets the correlation between a buffer neuron and another buffer neuron to the worst possible value (-1), and the correlation between a buffer neuron and a regular neuron to the best possible value (+1). This ensures that buffer neurons are never matched with each other, and always matched with a regular neuron. Formally, given an original cost matrix $\mathbf{C}^{n \times n}$ and an expansion factor γ , the drop-in cost matrix $\hat{\mathbf{C}}^{n + \lceil \gamma n \rceil \times n + \lceil \gamma n \rceil}$ is constructed as:

$$\hat{\mathbf{C}}_{ij} = \begin{cases} \mathbf{C}_{ij} & \text{if } i \leq n, j \leq n, \\ -1 & \text{if } i > n, j > n \\ 1 & \text{otherwise.} \end{cases}$$

where \mathbf{C}_{ij} denotes the value at position i, j and n is the size of the original correlation matrix.

The alternative option, which we will name **adaptive buffer assignment**, sets the correlation of matching a buffer neuron with any other neuron to a threshold ϵ . The effect of this is that regular neurons and buffer neurons will only be matched if the correlation would otherwise be worse than ϵ . Otherwise, buffer neurons are matched amongst each other. This means that the relative width increase of the layer after merging is determined adaptively, but limited to a maximum of γ . In this case, the construction of $\hat{\mathbf{C}}^{n + \lceil \gamma n \rceil \times n + \lceil \gamma n \rceil}$ is defined as:

$$\hat{\mathbf{C}}_{ij} = \begin{cases} \mathbf{C}_{ij} & \text{if } i \leq n, j \leq n, \\ \epsilon & \text{otherwise.} \end{cases}$$

The resulting LAP solutions for modifying the correlation matrix from Figure 3.2 for forced and adaptive buffer assignment can be seen in Figure 3.3 a) and b). Note that not only are the worst matches separated, but also the previously best matches have been further improved through the relaxation of the problem

3 A method for partial merging

statement. When adaptive buffer assignment does not use up all available buffer neurons, i.e. buffer neurons are matched among themselves, like the case in the Figure 3.3 b), the superfluous buffer neurons can simply be discarded.

		model B					buffer				model B					buffer	
		n_1	n_2	n_3	n_4	n_5	b_1	b_2			n_1	n_2	n_3	n_4	n_5	b_1	b_2
model A	n_1	.98	.93	.13	.37	.56	1.0	1.0	n_1	.98	.93	.13	.37	.56	.60	.60	
	n_2	.52	.09	.05	.27	.55	1.0	1.0	n_2	.52	.09	.05	.27	.55	.60	.60	
	n_3	.07	.20	.28	.56	.69	1.0	1.0	n_3	.07	.20	.28	.56	.69	.60	.60	
	n_4	.69	.85	.79	.42	.77	1.0	1.0	n_4	.69	.85	.79	.42	.77	.60	.60	
	n_5	.14	.51	.36	.80	.68	1.0	1.0	n_5	.14	.51	.36	.80	.68	.60	.60	
buffer	b_1	1.0	1.0	1.0	1.0	1.0	-1.0	-1.0	b_1	.60	.60	.60	.60	.60	.60	.60	
	b_2	1.0	1.0	1.0	1.0	1.0	-1.0	-1.0	b_2	.60	.60	.60	.60	.60	.60	.60	

a)
b)

Figure 3.3: **a)** The modified correlation matrix and LAP solution with forced buffer assignment. **b)** The modified correlation matrix and LAP solution with adaptive buffer assignment (threshold $\epsilon = 0.6$). Both examples use an expansion factor of $\gamma = 0.45$, which, after rounding down, adds up to two buffer neurons to the layer.

3.3 Constructing the merged model

Given the two original endpoint models and the LAP solution with model and buffer assignments, we construct the new, partially merged model as follows. Consider two weight matrices $\mathbf{W}_\ell^{(1)}$ and $\mathbf{W}_\ell^{(2)}$ from layer ℓ in endpoint models θ_1 and θ_2 . Both weight matrices have shape $n \times m$, where n represents the current layer size and m the number of inputs, determined by the size of the previous layer. Per convention, we will let n be the number of rows and m be the number of columns. We then construct two functionally equivalent weight matrices $\mathbf{W}'_\ell^{(1)}$ and $\mathbf{W}'_\ell^{(2)}$ of shape $n + \lceil \gamma n \rceil \times m + \lceil \gamma m \rceil$:

3 A method for partial merging

$$\mathbf{W}'_{\ell,ij} = \begin{cases} \mathbf{W}_{\ell,ij} & \text{if } i \leq n, j \leq m, \\ 0 & \text{otherwise.} \end{cases}$$

Next, we construct our permutation matrices \mathbf{P}_ℓ from the LAP solutions such that

$$\mathbf{P}'_{\ell,ij} = \begin{cases} 1 & \text{if combination } \langle i, j \rangle \text{ is part of the LAP solution,} \\ 0 & \text{otherwise.} \end{cases}$$

Next, weight matrix $\mathbf{W}'_{\ell}{}^{(2)}$ is permuted for optimal alignment in accordance with Equation 2.6, i.e.

$$\hat{\mathbf{W}}_{\ell}'{}^{(2)} = \mathbf{P}_\ell \mathbf{W}'_{\ell}{}^{(2)} \mathbf{P}_{\ell-1}^\top$$

The interpolation of both models is non-trivial. Specifically, as only parts of the model shall be merged, we need to keep track of which parts are real units, and which ones are buffers. The final interpolated weight matrix \mathbf{W}_ℓ^α is constructed such that:

$$\mathbf{W}_\ell^{(\alpha)} = \begin{cases} (1 - \alpha) \mathbf{W}'_{\ell,ij}{}^{(1)} + \alpha \hat{\mathbf{W}}'_{\ell,ij}{}^{(2)} & \text{if } \text{mask}(i, j), \\ \mathbf{W}'_{\ell,ij}{}^{(1)} + \hat{\mathbf{W}}'_{\ell,ij}{}^{(2)} & \text{otherwise.} \end{cases}$$

where $\text{mask}(i, j) = i \leq m \wedge j \leq n \wedge \pi_\ell^{-1}(i) \leq n \wedge \pi_{\ell-1}^{-1}(j) \leq m$ is true if both values originate from $\mathbf{W}_\ell^{(1)}$ and $\mathbf{W}_\ell^{(2)}$ and false if at least one value originates from a buffer unit (thus turning at least one operand of the addition into a 0).

The entire process of extending, permuting and (partially) merging the weight matrices in an MLP can be seen in Figure 3.4. The permutation of CNN kernels, bias and batch norm parameters follows the same logic, using the appropriate dimensions and permutation matrices established in Section 2.2 on permutation symmetry.

3 A method for partial merging

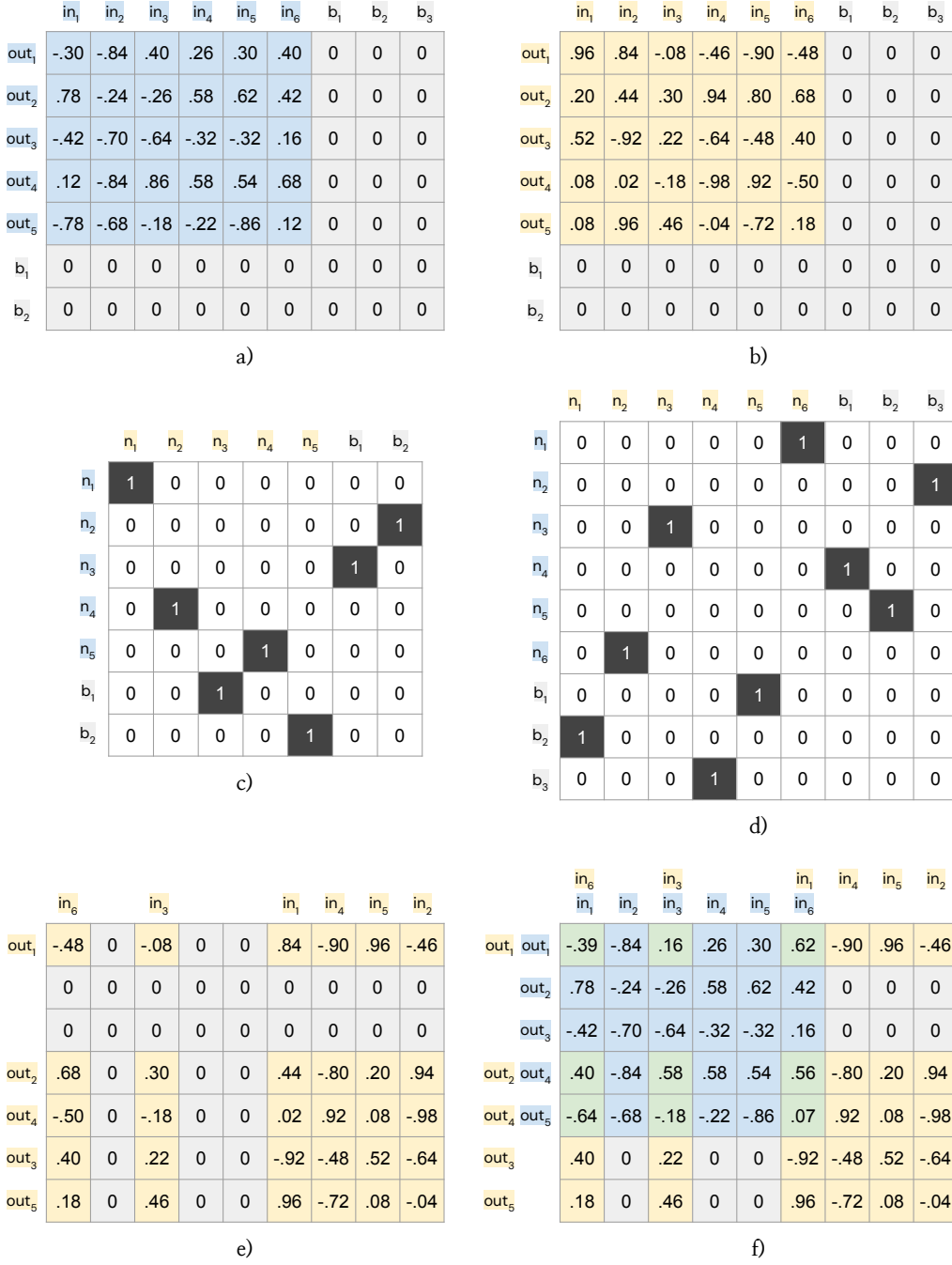


Figure 3.4: **a)**, **b)** The extended weight matrices $\mathbf{W}'_{\ell}^{(1)}$ and $\mathbf{W}'_{\ell}^{(2)}$ **c)** The permutation matrix derived from the LAP solution in Figure 3.3 **a)** **d)** The permutation matrix from the previous layer. **e)** The permuted weight matrix $\hat{\mathbf{W}}'_{\ell}^{(2)}$ **f)** The weight matrix of the merged model $\mathbf{W}'_{\ell}^{(\alpha)}$ for $\alpha = 0.5$.

4 Experiments

This section documents the experiments conducted in this thesis. In it, we will establish baselines to compare against, evaluate our proposed method in a simple setting, and finally conduct more detailed experiments for specific training regimes such as split data training or layer-wise expansions.

4.1 Baselines

As a first step, we will try to reproduce the existing results of Ainsworth et al. [7] and Jordan et al. [10] on regular or “full” model merging. Only after that are we going to test our method that allows for partial merging.

Following [7] and [10] we are evaluating three different architectures: MLPs, VGGs, and ResNets. We construct models of varying depth and width, yielding a total of 56 MLP, 8 VGG, and 4 ResNet architectures. Each architecture is trained two times on each dataset – once for each interpolation endpoint – which yields a total of 160 distinct models. The used datasets are MNIST [44] for the MLPs and CIFAR10 [45] and SVHN [46] for the convolutional models. The training hyperparameters for each model type can be seen in Table 4.1, with supplementary information on data augmentations in Table 4.2. Note that not all possible combinations are evaluated due to computational constraints. Additionally, we use CIFAR100 [45] in combination with a ResNet20 for the split data experiment.

For each model, we will report the test accuracy and loss of each endpoint model individually. Additionally, we perform ensembling of both endpoint models at logit-level (pre-softmax) and model merging using activation matching (see Subsection 2.4.1) for 21 interpolation steps $\alpha \in \{0.0, 0.05, \dots, 0.95, 1.0\}$. The

4 Experiments

Hyper-parameter	MLP{3..10}	VGG{11,13,16,19}	ResNet{18,20}
First layer width	512	64	16
Trained widths	0.125, 0.25, 0.5, 1, 2, 4, 8	0.25, 0.5, 1, 2, 4	1, 2, 4, 8
Normalization	Batch norm	Batch norm	Batch norm
Batch size	1000	500	500
Epochs	100	100	200
Learning rate	Cosine 0.2	Cosine 0.08	Cosine 0.4
Weight decay	0.0	0.0005	0.0001
Datasets	MNIST	CIFAR10, SVHN	CIFAR10, SVHN

Table 4.1: Training hyperparameters

Dataset	RandomTranslate/ RandomAffine	RandomHorizontalFlip
MNIST [44]	✓	✗
CIFAR10 [45]	✓	✓
CIFAR100 [45]	✓	✓
SVHN [46]	✓	✗

Table 4.2: Dataset training augmentations

choice of activation matching as alignment method follows [10] and is motivated by the need to collect activations for applying REPAIR anyway. As we will compare the performance of partial merging relatively to the performance of full merging the actual choice of alignment method should have no major effect – even if the absolute performance is slightly better or worse by a few percentage points. If anything, as activation matching performed slightly worse in [7] in terms of test loss, it is reasonable to assume that the performance metrics obtained in this thesis represent a lower bound in absolute terms and could be further improved by adapting a more sophisticated alignment method such as Sinkhorn-Rebasin [35] to work for partial merging.

For reference, we report the average test accuracy and test loss of each model in Table A.1, A.2, and A.3 in Appendix A.

4 Experiments

For our evaluation, we will also report an *accuracy barrier* and *loss barrier*, which differ from the logic in the original Definition 2.5. First, as partial merging and REPAIR do not represent a strict interpolation along a linear path, we will look at combinations of the models of any kind, guided by a factor α . This also includes ensembling, where α denotes the weighting factor for each models logits. Second, we will only look at the difference in accuracy or loss at $\alpha = 0.5$, as this position not only represents the highest barrier in most cases, but also allows for our metric to become negative, representing an improvement instead of a degradation. The barrier metrics used for the remainder of this thesis are given in Definition 4.1.

Definition 4.1

The **loss barrier** $BL(\theta_1, \theta_2)$ between a pair of models is defined as the increase in loss of the combination $C(\theta_1, \theta_2, \alpha)$ of both models, guided by a factor $\alpha = 0.5$, relative to the average of the two endpoint losses:

$$BL(\theta_1, \theta_2) = \mathcal{L}(C(\theta_1, \theta_2, 0.5)) - 0.5(\mathcal{L}(\theta_1) + \mathcal{L}(\theta_2)) \quad (4.1)$$

The **accuracy barrier** $BA(\theta_1, \theta_2)$ between a pair of models is defined as the decrease in accuracy of the combination $C(\theta_1, \theta_2, \alpha)$ of both models, guided by a factor $\alpha = 0.5$, relative to the average of the two endpoint accuracies:

$$BA(\theta_1, \theta_2) = 0.5(\text{acc}(\theta_1) + \text{acc}(\theta_2)) - \text{acc}(C(\theta_1, \theta_2, 0.5)) \quad (4.2)$$

We report the accuracy and loss barriers of each model combination using full merging in Table 4.3, 4.4, and 4.5. Each table also includes the resulting barrier decreases by adding REPAIR, as well as by using ensembling instead. The barrier reductions of ensembling are greater than 100% if they not only remove the barrier but also improve the combined performance beyond the average of both endpoints. Note that the reductions are relative to the original barrier, and not to the endpoint average. Thus, an accuracy barrier decrease by 105% means that the magnitude of the performance bump from ensembling is 5% of the original barrier size.

The results of our baselines follow the observations in the original publications. First, the absolute barrier sizes decrease with width and increase with depth. Second, applying REPAIR decreases the barriers drastically and approximately

4 Experiments

equals the barriers regardless of depth (still, wider models have a smaller barrier). Third, ensembling improves loss and accuracy across almost all models.

		MNIST							
		MLP3	MLP4	MLP5	MLP6	MLP7	MLP8	MLP9	MLP10
0.125×		3.3%p	7.4%p	5.3%p	7.5%p	11.2%p	8.9%p	6.1%p	13.6%p
		0.101	0.229	0.169	0.243	0.335	0.287	0.186	0.43
		1.2%p (-64.9%)	1.2%p (-84.0%)	2.1%p (-60.3%)	1.8%p (-75.7%)	2.1%p (-80.9%)	2.7%p (-69.4%)	2.1%p (-65.2%)	2.0%p (-85.2%)
		0.044 (-56.2%)	0.052 (-77.3%)	0.087 (-48.4%)	0.102 (-58.1%)	0.111 (-66.8%)	0.139 (-51.6%)	0.109 (-41.3%)	0.118 (-72.5%)
0.25×		-0.2%p (-107.5%)	-0.2%p (-102.5%)	-0.3%p (-105.2%)	-0.2%p (-103.3%)	-0.4%p (-103.6%)	-0.3%p (-103.0%)	-0.2%p (-104.0%)	-0.3%p (-102.1%)
		-0.01 (-109.7%)	-0.012 (-105.3%)	-0.011 (-106.8%)	-0.013 (-105.5%)	-0.015 (-104.5%)	-0.014 (-104.9%)	-0.015 (-108.1%)	-0.015 (-103.5%)
		1.1%p	2.9%p	6.0%p	7.6%p	6.0%p	8.0%p	9.1%p	5.6%p
		0.04	0.097	0.223	0.267	0.213	0.352	0.342	0.19
0.5×		0.6%p (-46.4%)	0.8%p (-72.0%)	1.1%p (-82.4%)	0.9%p (-88.0%)	1.3%p (-78.9%)	1.0%p (-87.7%)	1.7%p (-81.3%)	1.3%p (-76.9%)
		0.03 (-24.3%)	0.038 (-60.5%)	0.062 (-72.3%)	0.06 (-77.7%)	0.069 (-67.6%)	0.056 (-84.1%)	0.108 (-68.6%)	0.073 (-61.4%)
		-0.2%p (-121.3%)	-0.2%p (-105.8%)	-0.2%p (-103.2%)	-0.2%p (-102.6%)	-0.2%p (-103.7%)	-0.1%p (-101.6%)	-0.2%p (-102.1%)	-0.1%p (-102.3%)
		-0.007 (-118.4%)	-0.008 (-108.1%)	-0.008 (-103.5%)	-0.01 (-103.6%)	-0.009 (-104.0%)	-0.009 (-102.6%)	-0.01 (-103.0%)	-0.011 (-105.6%)
1×		1.0%p	1.8%p	5.6%p	3.3%p	5.1%p	6.1%p	6.2%p	5.9%p
		0.031	0.064	0.197	0.118	0.174	0.221	0.211	0.217
		0.3%p (-66.7%)	0.4%p (-77.5%)	0.6%p (-89.2%)	0.6%p (-81.6%)	0.8%p (-83.8%)	0.8%p (-86.7%)	0.9%p (-85.6%)	0.6%p (-89.8%)
		0.015 (-51.9%)	0.022 (-65.6%)	0.035 (-82.5%)	0.04 (-66.2%)	0.049 (-72.1%)	0.043 (-80.6%)	0.056 (-73.3%)	0.051 (-76.5%)
2×		-0.1%p (-110.1%)	-0.1%p (-103.8%)	-0.0%p (-100.9%)	-0.1%p (-104.0%)	-0.1%p (-101.3%)	-0.1%p (-101.6%)	-0.1%p (-101.3%)	-0.1%p (-101.5%)
		-0.004 (-113.3%)	-0.005 (-107.6%)	-0.005 (-102.3%)	-0.007 (-105.6%)	-0.006 (-103.4%)	-0.007 (-103.0%)	-0.006 (-102.8%)	-0.009 (-104.1%)
		0.7%p	1.4%p	2.6%p	4.1%p	3.4%p	6.0%p	8.0%p	5.7%p
		0.022	0.05	0.091	0.134	0.131	0.254	0.322	0.209
4×		0.3%p (-55.0%)	0.3%p (-78.7%)	0.5%p (-80.7%)	0.4%p (-89.4%)	0.7%p (-79.9%)	0.7%p (-88.4%)	0.6%p (-92.2%)	0.8%p (-86.4%)
		0.011 (-49.3%)	0.017 (-65.2%)	0.026 (-71.5%)	0.027 (-80.1%)	0.037 (-71.3%)	0.045 (-82.4%)	0.046 (-85.6%)	0.055 (-73.6%)
		-0.1%p (-107.4%)	-0.1%p (-104.7%)	-0.1%p (-103.3%)	-0.1%p (-101.2%)	-0.1%p (-102.5%)	-0.1%p (-101.2%)	-0.1%p (-100.9%)	-0.0%p (-100.3%)
		-0.002 (-110.9%)	-0.003 (-105.4%)	-0.004 (-104.1%)	-0.004 (-102.9%)	-0.004 (-102.8%)	-0.004 (-101.8%)	-0.005 (-101.5%)	-0.006 (-102.9%)
8×		0.6%p	1.5%p	2.3%p	3.3%p	4.3%p	4.0%p	4.7%p	6.2%p
		0.017	0.051	0.072	0.113	0.157	0.148	0.175	0.26
		0.1%p (-80.0%)	0.3%p (-82.1%)	0.3%p (-84.9%)	0.6%p (-81.8%)	0.6%p (-86.4%)	0.6%p (-85.5%)	0.6%p (-86.2%)	0.6%p (-90.9%)
		0.007 (-57.7%)	0.015 (-71.3%)	0.018 (-75.0%)	0.032 (-71.4%)	0.038 (-75.7%)	0.038 (-74.7%)	0.049 (-71.9%)	0.048 (-81.4%)
16×		0.0%p (-99.2%)	0.0%p (-100.0%)	-0.1%p (-103.1%)	-0.0%p (-101.5%)	-0.0%p (-100.5%)	-0.0%p (-101.1%)	-0.1%p (-101.6%)	-0.1%p (-101.5%)
		-0.001 (-109.0%)	-0.002 (-103.4%)	-0.003 (-103.7%)	-0.004 (-103.5%)	-0.003 (-101.8%)	-0.003 (-102.3%)	-0.003 (-102.0%)	-0.004 (-101.6%)
		0.4%p	1.6%p	2.6%p	3.2%p	4.6%p	4.5%p	5.3%p	5.3%p
		0.014	0.045	0.085	0.108	0.167	0.173	0.201	0.202
32×		0.1%p (-74.4%)	0.3%p (-83.5%)	0.3%p (-88.6%)	0.5%p (-84.3%)	0.6%p (-87.5%)	0.5%p (-89.4%)	0.5%p (-89.9%)	0.6%p (-88.3%)
		0.007 (-47.6%)	0.013 (-71.3%)	0.018 (-78.9%)	0.029 (-73.3%)	0.034 (-79.8%)	0.038 (-78.2%)	0.041 (-79.7%)	0.046 (-77.5%)
		0.0%p (-94.9%)	0.0%p (-100.0%)	-0.0%p (-100.8%)	-0.1%p (-102.5%)	-0.1%p (-101.8%)	-0.1%p (-101.1%)	0.0%p (-99.9%)	-0.1%p (-101.8%)
		-0.001 (-109.7%)	-0.002 (-103.7%)	-0.002 (-102.3%)	-0.003 (-102.5%)	-0.003 (-101.9%)	-0.003 (-101.9%)	-0.004 (-102.1%)	-0.005 (-102.3%)
64×		0.4%p	1.3%p	2.6%p	2.9%p	4.3%p	4.2%p	5.4%p	5.0%p
		0.011	0.039	0.08	0.097	0.155	0.16	0.214	0.202
		0.1%p (-71.4%)	0.4%p (-72.5%)	0.3%p (-86.9%)	0.3%p (-89.0%)	0.5%p (-89.3%)	0.5%p (-87.6%)	0.6%p (-89.3%)	0.5%p (-89.3%)
		0.008 (-26.9%)	0.014 (-63.3%)	0.021 (-73.3%)	0.026 (-72.7%)	0.031 (-80.2%)	0.035 (-77.9%)	0.043 (-80.0%)	0.041 (-79.6%)
128×		-0.0%p (-100.0%)	-0.0%p (-101.5%)	-0.1%p (-102.3%)	0.0%p (-100.0%)	-0.1%p (-101.9%)	-0.0%p (-100.5%)	-0.1%p (-101.0%)	-0.0%p (-100.6%)
		-0.001 (-112.6%)	-0.002 (-103.9%)	-0.002 (-102.6%)	-0.003 (-102.9%)	-0.003 (-101.8%)	-0.003 (-102.1%)	-0.003 (-101.4%)	-0.005 (-102.5%)

Table 4.3: Black: Absolute test accuracy barrier (top) and test loss barrier (bottom) of fully merged MLP models trained on MNIST for different depths and widths. Purple: The respective absolute barrier (and relative reduction to full merging) after additionally applying REPAIR. Grey: The respective absolute barrier (and relative reduction to full merging) when using ensembling instead.

4 Experiments

		<i>CIFAR10</i>			
		VGG11	VGG13	VGG16	VGG19
0.25×	48.3%p				
	1.596				
	11.9%p (-75.4%)				
	0.495 (-69.0%)				
	-1.9%p (-103.9%)	-	-	-	-
0.5×	61.4%p				
	1.81				
	5.7%p (-90.7%)				
	0.232 (-87.2%)				
	-1.4%p (-102.3%)	-	-	-	-
1×	34.8%p		57.2%p	78.6%p	80.5%p
	0.938		1.861	2.176	2.2
	3.9%p (-88.9%)	3.8%p (-93.3%)	4.2%p (-94.6%)	4.9%p (-93.9%)	
	0.164 (-82.5%)	0.149 (-92.0%)	0.176 (-91.9%)	0.214 (-90.3%)	
	-0.9%p (-102.6%)	-0.9%p (-101.5%)	-1.1%p (-101.4%)	-1.1%p (-101.4%)	
2×	26.6%p				
	0.753				
	2.2%p (-91.6%)				
	0.096 (-87.2%)				
	-0.6%p (-102.1%)	-	-	-	-
4×	21.1%p				
	0.612				
	1.4%p (-93.5%)				
	0.066 (-89.2%)				
	-0.4%p (-101.9%)	-	-	-	-
	-0.026 (-104.2%)				

Table 4.4: Black: Absolute test accuracy barrier (top) and test loss barrier (bottom) of fully merged VGG models trained on CIFAR10 for different depths and widths. Purple: The respective absolute barrier (and relative reduction to full merging) after additionally applying REPAIR. Grey: The respective absolute barrier (and relative reduction to full merging) when using ensembling instead.

4 Experiments

	<i>CIFAR10</i>	<i>SVHN</i>
	ResNet18	ResNet18
	81.9%p	74.9%p
	4.446	2.512
1×	22.2%p (-72.9%)	6.4%p (-91.4%)
	1.293 (-70.9%)	0.24 (-90.5%)
	-1.2%p (-101.5%)	-0.5%p (-100.6%)
	-0.087 (-102.0%)	-0.03 (-101.2%)
	80.4%p	53.3%p
	3.398	1.523
2×	8.1%p (-89.9%)	2.2%p (-95.9%)
	0.361 (-89.4%)	0.071 (-95.3%)
	-0.7%p (-100.9%)	-0.4%p (-100.7%)
	-0.055 (-101.6%)	-0.021 (-101.4%)
	84.8%p	67.7%p
	3.911	1.623
4×	4.5%p (-94.7%)	1.0%p (-98.5%)
	0.171 (-95.6%)	0.018 (-98.9%)
	-0.4%p (-100.5%)	-0.3%p (-100.5%)
	-0.041 (-101.1%)	-0.017 (-101.1%)
	85.1%p	63.8%p
	3.326	1.624
8×	2.4%p (-97.1%)	0.7%p (-99.0%)
	0.087 (-97.4%)	0.014 (-99.2%)
	-0.5%p (-100.5%)	-0.3%p (-100.4%)
	-0.033 (-101.0%)	-0.013 (-100.8%)

Table 4.5: Black: Absolute test accuracy barrier (top) and test loss barrier (bottom) of fully merged ResNet models trained on CIFAR10 and SVHN for different depths and widths. Purple: The respective absolute barrier (and relative reduction to full merging) after additionally applying REPAIR. Grey: The respective absolute barrier (and relative reduction to full merging) when using ensembling instead.

4.2 Partial merging with forced buffer assignment

We will now evaluate whether the *forced buffer assignment* partial merging method proposed in Chapter 3 is able to beat full merging, and to which extent it can reach the performance boost that ensembling yields when we increase the expansion factor γ .

For this, we partially merge the just evaluated models using values of $\gamma \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$, yielding an expansion of the layer widths by γ across all layers. We additionally report results after applying REPAIR to only the merged section of the model – the weights pointing to buffer neurons and their associated batch norm target values remain the same.

The resulting test accuracies and losses (depending on the interpolation factor α) for a VGG11 trained on CIFAR10 can be seen in Figure 4.1 and the results for a ResNet18 trained on SVHN in Figure 4.2. The plots for an MLP are included as Appendix Figure A.1. The pattern of gradually improving accuracy and loss with increasing γ is present across all architectures, widths, and depths, the Figures therefore shall just serve as representative examples. The red lines for $\gamma = 1.0$ are exactly on top of the black lines of ensembling, as an expansion of 100% (no overlap at all) is functionally equivalent to executing both models in parallel.

Another observation is that the added layer width and the resulting performance increase have a non-linear relationship. For example, using REPAIR and increasing the final width through addition of just 20% buffer neurons ($\gamma = 0.2$) yields, on average, an additional accuracy barrier reduction of 53.6% in VGGs when comparing to full merging with REPAIR. The resulting additional decreases for $\gamma = 0.2$ across all widths and depths are reported in Table 4.6 (MLPs), Table 4.7 (VGGs) and Table 4.8 (ResNets). This non-linear relationship is further explored in Figures 4.3 to 4.6, where the resulting accuracy or loss at $\alpha = 0.5$ is plotted against γ . A complete elimination of the accuracy and loss barriers is typically seen between $\gamma = 0.5$ and $\gamma = 0.8$ when additionally using REPAIR.

A graphic example of first layer kernel alignment after +50% buffer neuron expansion can be seen in appendix Figure A.2, and serves as a counterpart to Figure 3.1 that shows partial merging.

4 Experiments

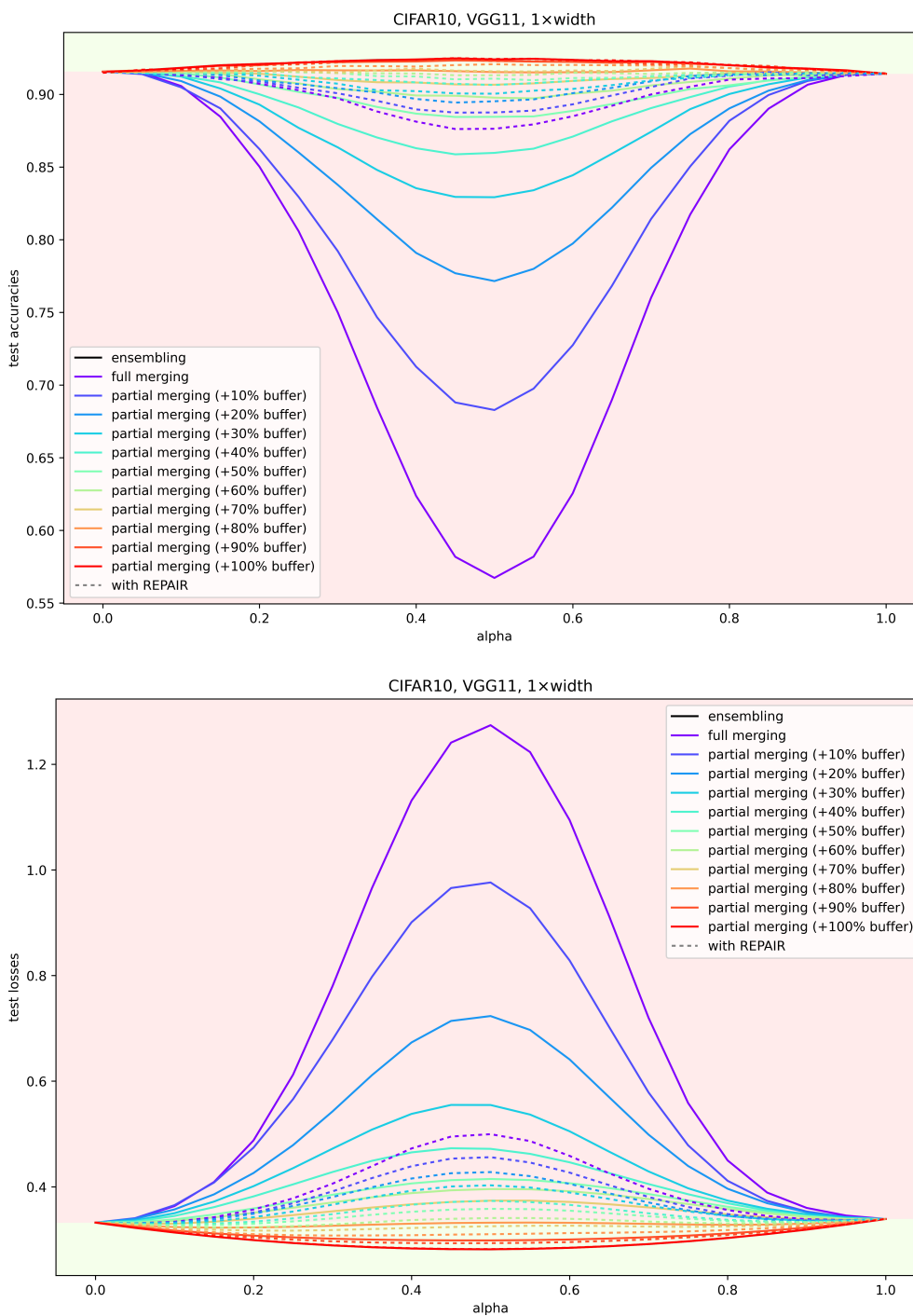


Figure 4.1: Test accuracies and losses with and without REPAIR for a regular-width VGG11. An improvement in accuracy and loss compared to the endpoint model average (green area) is achieved beyond an addition of 80% buffer neurons to the original layer width and beyond 70% when using REPAIR.

4 Experiments

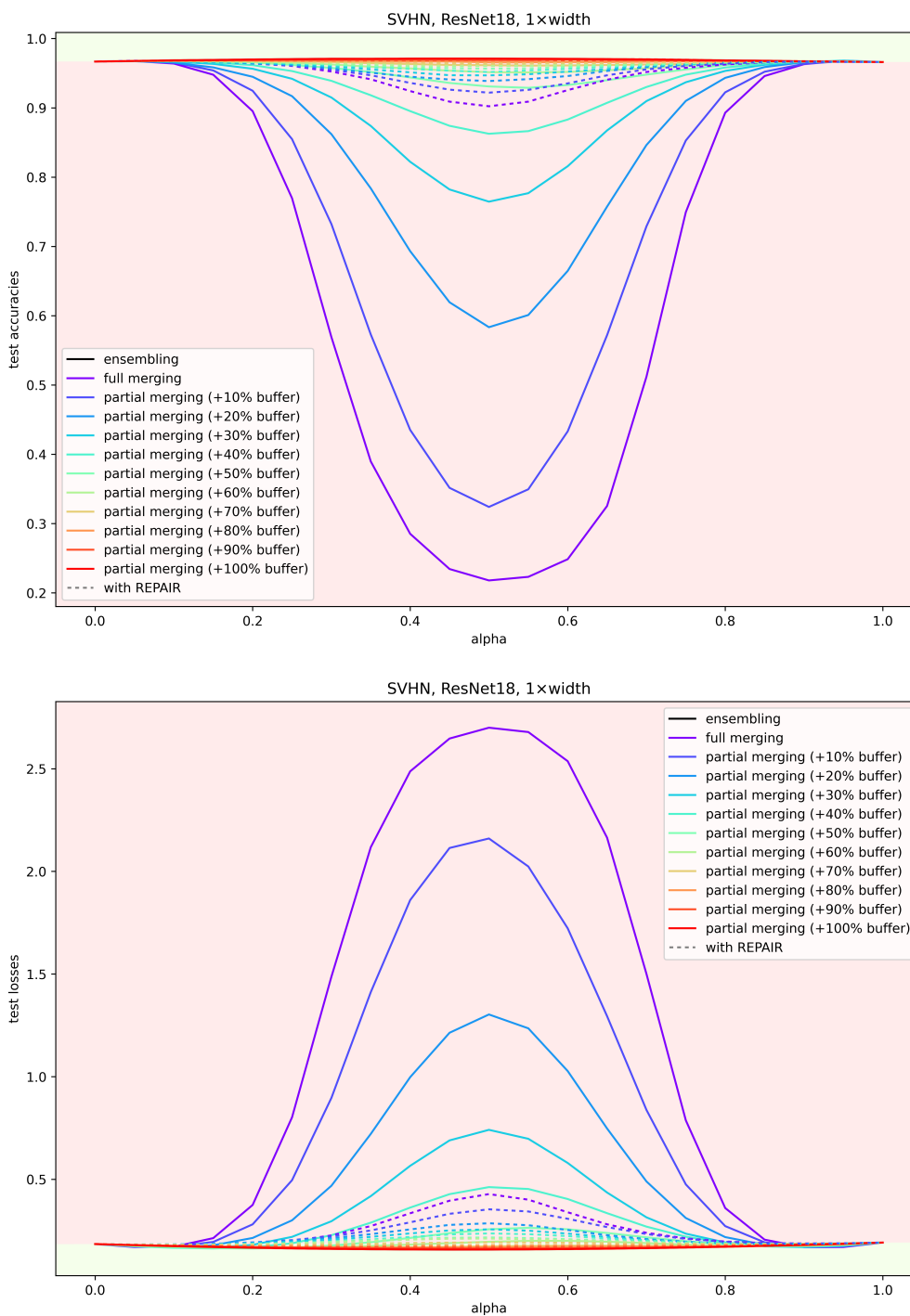


Figure 4.2: Test accuracies and losses with and without REPAIR for a regular-width ResNet18. An improvement in accuracy and loss compared to the endpoint model average (green area) is achieved beyond an addition of 90% buffer neurons to the original layer width and beyond 80% when using REPAIR.

4 Experiments

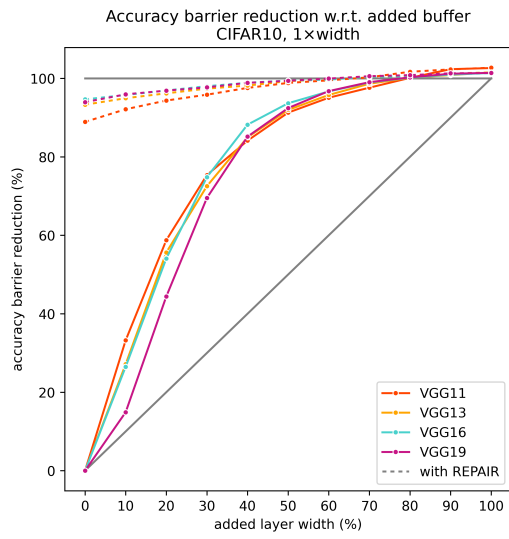


Figure 4.3: Added layer width through buffer neurons vs. accuracy barrier reduction, for different depth VGGs.

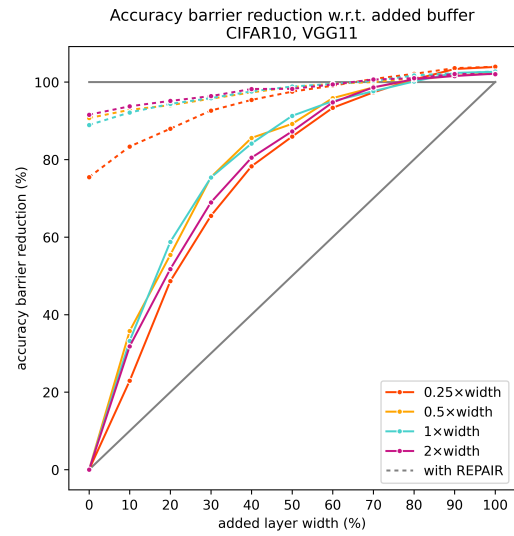


Figure 4.4: Added layer width through buffer neurons vs. accuracy barrier reduction, for different width VGG11s.

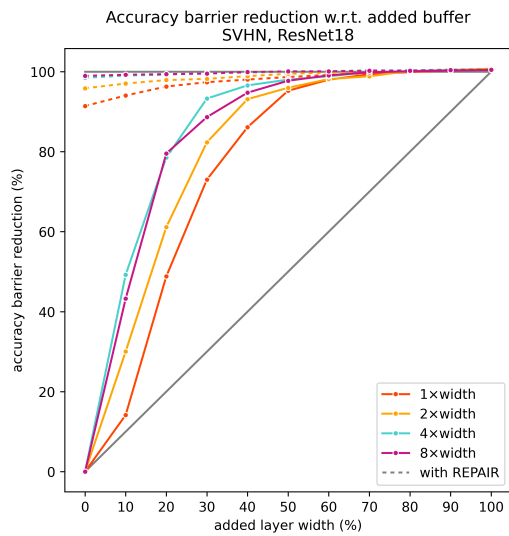


Figure 4.5: Added layer width through buffer neurons vs. accuracy barrier reduction, for different width ResNets.

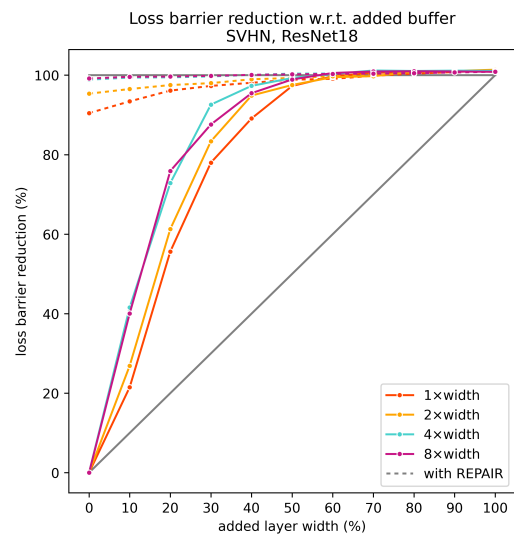


Figure 4.6: Added layer width through buffer neurons vs. loss barrier reduction, for different width ResNets.

4 Experiments

		<i>MNIST</i>							
		MLP3	MLP4	MLP5	MLP6	MLP7	MLP8	MLP9	MLP10
0.125×		-56.4%	-29.5%	-40.6%	-57.9%	-51.3%	-49.3%	-50.4%	-39.2%
		-52.6%	-44.2%	-31.5%	-58.7%	-50.0%	-48.6%	-56.4%	-34.1%
0.25×		-46.0%	-70.7%	-48.8%	-47.3%	-38.9%	-24.4%	-55.9%	-45.0%
		-38.6%	-66.9%	-41.6%	-48.9%	-43.3%	-15.6%	-44.3%	-40.3%
0.5×		-75.8%	-12.2%	-28.3%	-39.0%	-32.7%	-29.6%	-46.1%	-41.7%
		-64.0%	-28.1%	-41.5%	-45.7%	-31.1%	-30.3%	-28.5%	-28.5%
1×		-74.6%	-37.3%	-23.8%	-18.2%	-40.9%	-31.4%	-32.0%	-35.9%
		-38.7%	-33.8%	-40.9%	-23.4%	-37.8%	-30.6%	-43.7%	-33.9%
2×		-40.0%	-25.9%	-52.9%	-43.3%	-37.3%	-46.2%	-43.4%	-24.8%
		-52.2%	-40.3%	-45.1%	-44.6%	-45.2%	-33.0%	-37.1%	-35.3%
4×		-70.0%	-34.6%	-46.7%	-42.0%	-43.5%	-31.3%	-37.4%	-44.8%
		-33.8%	-36.1%	-26.8%	-35.5%	-38.6%	-36.4%	-33.2%	-40.2%

Table 4.6: Relative test accuracy barrier reduction (top value) and test loss barrier reduction (bottom value) reduction of a partially merged MLP with $\alpha = 0.5$ and $\gamma = 0.2$ (+20% width), after REPAIR, compared to the same value for $\gamma = 0$ (full merging).

4 Experiments

		<i>CIFAR10</i>				<i>SVHN</i>			
		VGG11	VGG13	VGG16	VGG19	VGG11	VGG13	VGG16	VGG19
0.25×		-50.9%	–	–	–	-51.6%	–	–	–
		-52.9%	–	–	–	-49.1%	–	–	–
0.5×		-35.8%	–	–	–	-58.4%	–	–	–
		-32.7%	–	–	–	-47.6%	–	–	–
1×		-49.0%	-43.2%	-42.3%	-48.6%	-53.6%	-55.0%	-71.0%	-56.4%
		-43.8%	-41.8%	-49.8%	-41.4%	-51.2%	-65.9%	-59.3%	-67.4%
2×		-42.2%	–	–	–	-49.0%	–	–	–
		-47.0%	–	–	–	-61.0%	–	–	–
4×		-51.1%	–	–	–	-100.6%	–	–	–
		-49.5%	–	–	–	-102.1%	–	–	–

Table 4.7: Relative test accuracy barrier reduction (top value) and test loss barrier reduction (bottom value) reduction of a partially merged VGG with $\alpha = 0.5$ and $\gamma = 0.2$ (+20% width), after REPAIR, compared to the same value for $\gamma = 0$ (full merging). The 20% width increase was sufficient to surpass complete barrier elimination in a 4× width VGG11 on SVHN.

		<i>CIFAR10</i>	<i>SVHN</i>
		ResNet18	ResNet18
1×		-43.7%	-56.7%
		-47.4%	-59.3%
2×		-38.8%	-49.4%
		-38.9%	-46.5%
4×		-45.9%	-50.7%
		-43.5%	-51.3%
8×		-36.0%	-41.4%
		-40.2%	-57.3%

Table 4.8: Relative test accuracy barrier reduction (top value) and test loss barrier reduction (bottom value) of a partially merged ResNets with $\alpha = 0.5$ and $\gamma = 0.2$ (+20% width), after REPAIR, compared to the same value for $\gamma = 0$ (full merging).

4.3 Does the choice of units even matter?

As we could see in the previous Section, having a partial overlap between two models results in a performance that lies between full merging and ensembling. But does the choice of which units to merge and which to keep separate even matter? In other words, is the increase in performance that we observe simply a result of a more diverse set of units in general (through the added width), or does our problem formulation as relaxed linear assignment problem provide an additional benefit by allowing for better correlated matches in the merged part?

In order to test this, we construct a simple baseline method that works as follows. First, we do align the units of our models as we would for full merging. Then, we randomly select $\lceil \gamma n \rceil$ of the n units in each layer and split them apart. Some of the split-apart units will have had high correlation coefficients, others will have had low ones. However, the resulting width increase is exactly the same as if doing forced buffer assignment with the reformulated LAP.

As visible in Figure 4.7 to 4.10, the LAP-based buffer assignment tends to outperform the random unit splitting, both before and after REPAIR. Only sporadically, the accuracy of the random method is equivalent to or better than the LAP-based assignment – after all, a random selection may pull apart a majority of bad matches too. However, even the random splitting of neurons results in a better-than-linear decrease of the accuracy barrier with respect to width in most cases, especially after applying REPAIR.

The large similarity between both curves suggests that the increased diversity of units that we have when going in the direction of ensembling plays a significant role in the improved performance we observe. Alternatively, it is possible that even the better matches in our merged model are still so bad that they would be better off remaining separate. Nonetheless, using the LAP formulation focusing on maximizing correlations in each layer almost always performs better, and should therefore be preferred.

4 Experiments

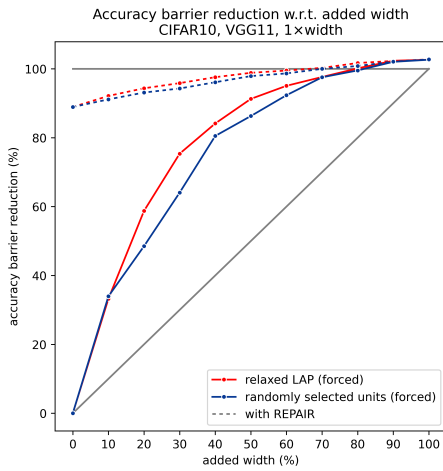


Figure 4.7: Randomly splitting apart neurons vs using the forced buffer assignment method, on two VGG11s trained on CIFAR10.

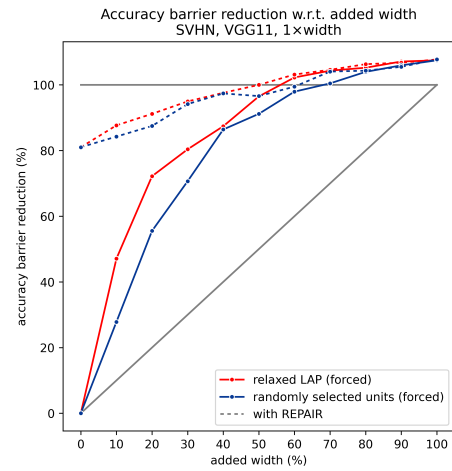


Figure 4.8: Randomly splitting apart neurons vs using the forced buffer assignment method, on two VGG11s trained on SVHN.

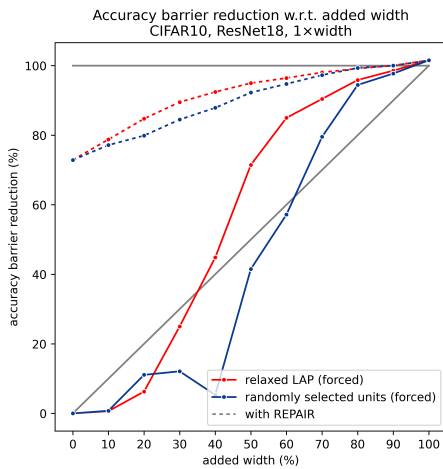


Figure 4.9: Randomly splitting apart neurons vs using the forced buffer assignment method, on two Resnet18s trained on CIFAR10.

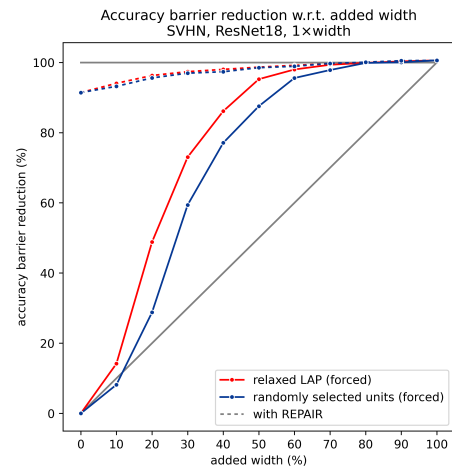


Figure 4.10: Randomly splitting apart neurons vs using the forced buffer assignment method, on two Resnet18s trained on SVHN.

4 Experiments

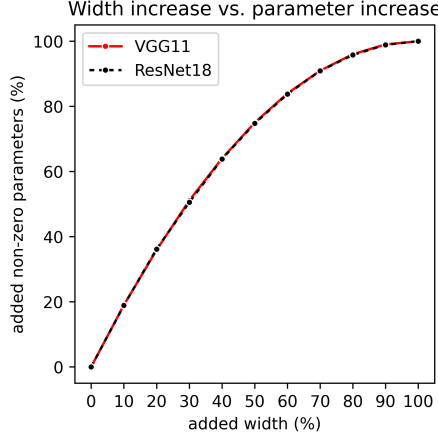


Figure 4.11: The relationship between width increase through added buffer neurons and parameter increase is not linear.

4.4 Parameter efficiency

The non-linear relationship between added width and barrier elimination described in section 4.2 is slightly misleading when it comes to the cost of layer widening. Specifically, not only is the relationship between added width and performance improvement non-linear, but so is the relationship between added width and model parameter increase. As an example, the first 10 percent of width increase from $\gamma = 0$ to $\gamma = 0.1$ are much more costly in terms of added parameters than the last 10 percent from $\gamma = 0.9$ to $\gamma = 1.0$. This phenomenon is grounded in the fact that the overlapping area of both weight matrices shrinks simultaneously in both dimensions, and as such approximately quadratically. We plot the relationship between γ and the added non-zero parameters in Figure 4.11.

Given this context, is the barrier decrease from partial merging still better than linear when comparing to added non-zero parameters instead of added layer width? We report the equivalent of Figure 4.3 to 4.6 with the non-zero parameter increase instead of the added width can be seen in Figure 4.12 to 4.15. While the relationship tends more towards linearity than before, and is sometimes even worse than linear (e.g. the VGG19 at $\gamma = 0.1$), the “S-shape” of the curves is only present without REPAIR (if at all), and in most cases the addition of x percent parameters still decreases the barrier by more than x percent.

4 Experiments

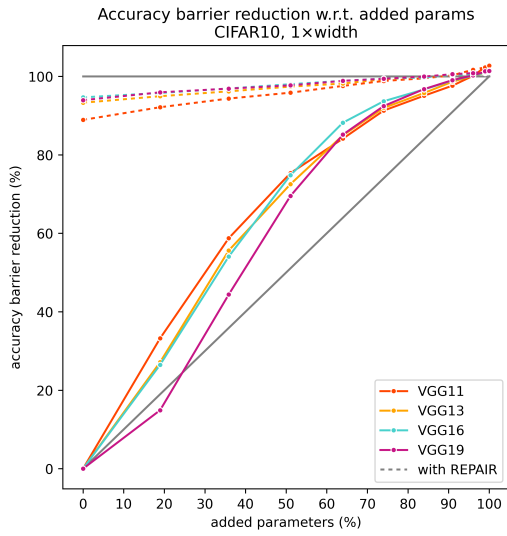


Figure 4.12: Added non-zero parameter count vs. accuracy barrier reduction, for different depth VGGs.

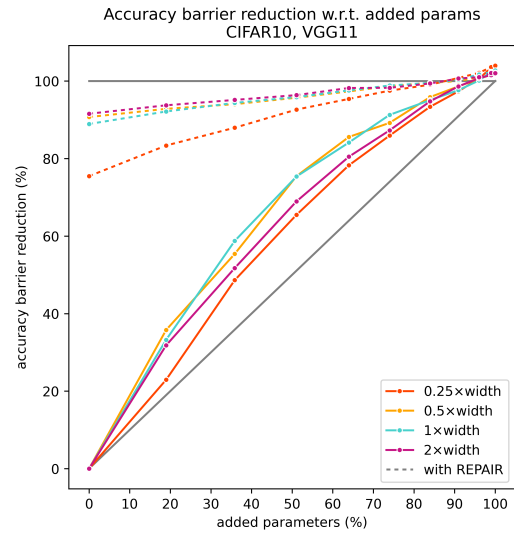


Figure 4.13: Added non-zero parameter count vs. accuracy barrier reduction, for different width VGG11s.

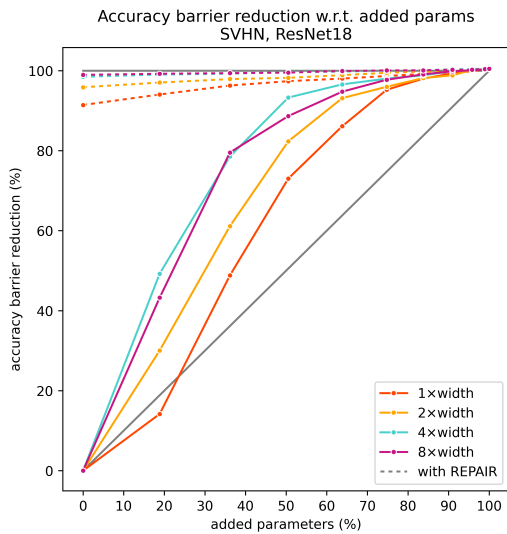


Figure 4.14: Added non-zero parameter count vs. accuracy barrier reduction, for different width ResNets.

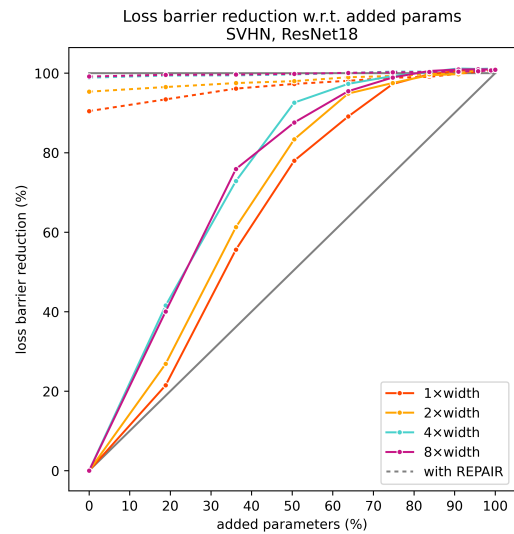


Figure 4.15: Added non-zero parameter count vs. loss barrier reduction, for different width ResNets.

4.5 Partial merging with adaptive buffer assignment

In addition to the previous experiments where the expansion of layers was hard-coded and uniform across all layers, we will now evaluate adaptive buffer assignment with different threshold values ϵ . We evaluate values of $\{0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.5, 0.6, 0.8\}$, which are not uniformly spaced, but have a higher resolution between 0.1 and 0.4, where the bulk of the correlations in the correlation matrices are located that are selected by the LAP solver, with wider models having a slightly higher average correlation.

When comparing adaptive buffer assignment to forced buffer assignment we cannot simply look at the total model width, as adaptive buffer assignment results in different widths per layer, purely dependent on the correlation coefficient makeup in that layer. We will report the used portion of buffer neurons per layer in our results (as an analogue to γ), but mainly look at the resulting parameter increase for the different values of ϵ . The maximum width increase that we allow in each layer is always set to $\gamma = 1.0$ in our experiments.

As an example for the distribution of correlation coefficients in different layers, we plot the correlations selected by the LAP solver with full merging for the VGG11s trained on CIFAR10 in Figure 4.16. Additional correlation plots for more models can be found in Figures A.3, A.4, and A.5 in the Appendix. The average selected correlations in the first and last layers are generally higher ($\mu_{\text{corr}} > 0.5$) and tend to be lower in the middle layers ($0.5 > \mu_{\text{corr}} > 0$)¹. Please note that the adaptive buffer assignment strategy does not necessarily pull apart all matches that previously had a correlation below ϵ ; instead ϵ represents a hard lower bound for correlations in the LAP solution. It is possible that satisfying this lower bound is also possible without pulling apart all matches that previously violated it, specifically if better matches are “freed up” in the process. This does, however, happen rarely, and as such the adaptive assignment strategy is similar to the method used in FedAvg [37].

¹This pattern is present as well when looking at the candidate correlation coefficients, i.e. not just the selected ones, but with significantly lower averages.

4 Experiments

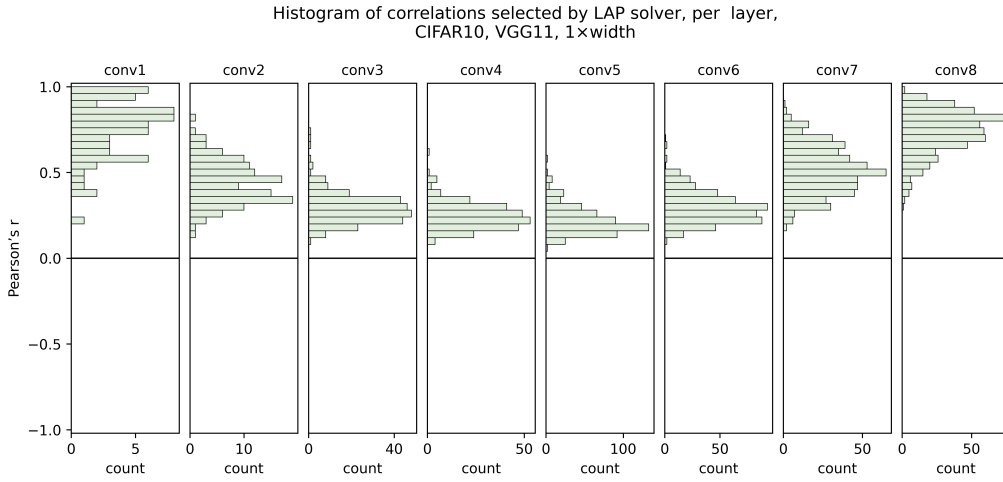


Figure 4.16: The activation correlations chosen by the LAP solver when fully merging two VGG11s trained on CIFAR10, per convolutional layer.

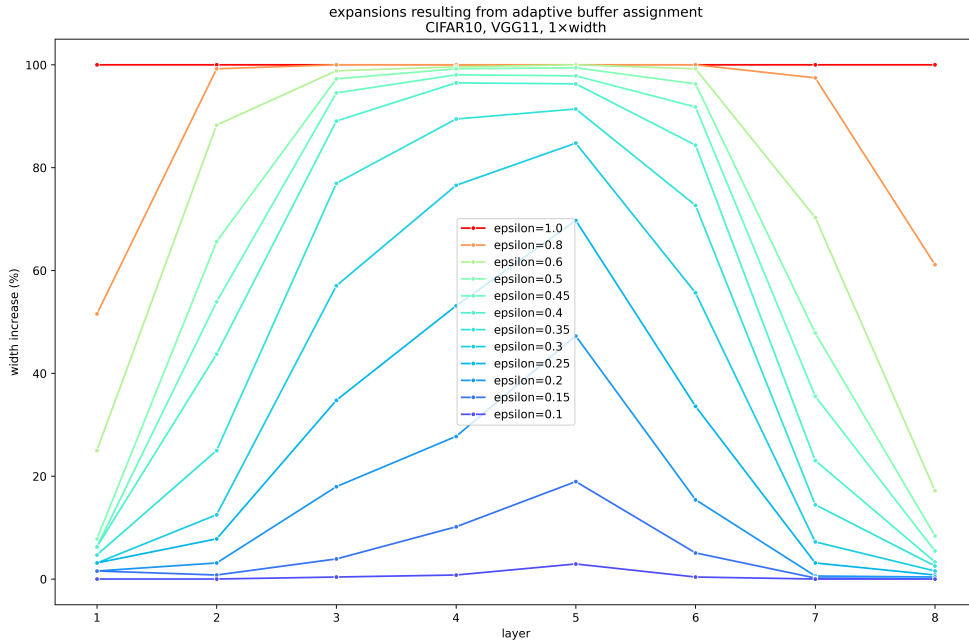


Figure 4.17: The resulting width expansion per layer for different values of ϵ for adaptive buffer assignment in two VGG11s trained on CIFAR10.

In addition to the correlations per layer we provide the resulting width increases per layer, depending on the values of ϵ , in Figure 4.17. They correspond well to the correlations recorded in the full merging setting, with the worst correlations –

4 Experiments

and thus the quickest layer expansion – in layer 5. While the pattern for ResNets is different, the general shape of correlation histogram and layer widths resulting from adaptive buffer assignment remains approximately aligned. However, the fact that some layers can only be expanded together, namely the residually connected layers groups $\{1, 3, 5\}$, $\{7, 9\}$, $\{11, 13\}$, and $\{15, 17\}$, results in some artefacts visible in Figure 4.18.

The most striking such anomaly is the low expansion of layer 15, despite its relatively moderate correlation coefficients. The reason behind this is that the permutation of each same-width residual layer group is commonly determined purely by the correlation coefficients of its last layer. In this case, the permutation (and expansion, depending on ϵ) of layer 15 was determined by layer 17, which has much better correlation coefficients. It could be worth it to collect the correlations in an earlier layer, or average the correlations across multiple layers inside each group, to get a more reliable measure.

The barrier decreases resulting from the adaptive buffer assignment strategy can be seen in Figure 4.19 to 4.22, where they are compared to forced buffer assignment. Which strategy outperforms which is inconsistent, with the adaptive strategy beating the forced assignment in the MLPs and VGG11s on CIFAR10, both strategies being approximately equivalent in the VGG11s on SVHN, and the forced assignment beating the adaptive assignment on the ResNet18s on SVHN (and CIFAR10, which is not plotted). When looking just at the performance after REPAIR, the adaptive assignment strategy seems to cross the 100%-barrier-reduction threshold slightly sooner than the forced assignment strategy, but the differences are small and not significant.

4 Experiments

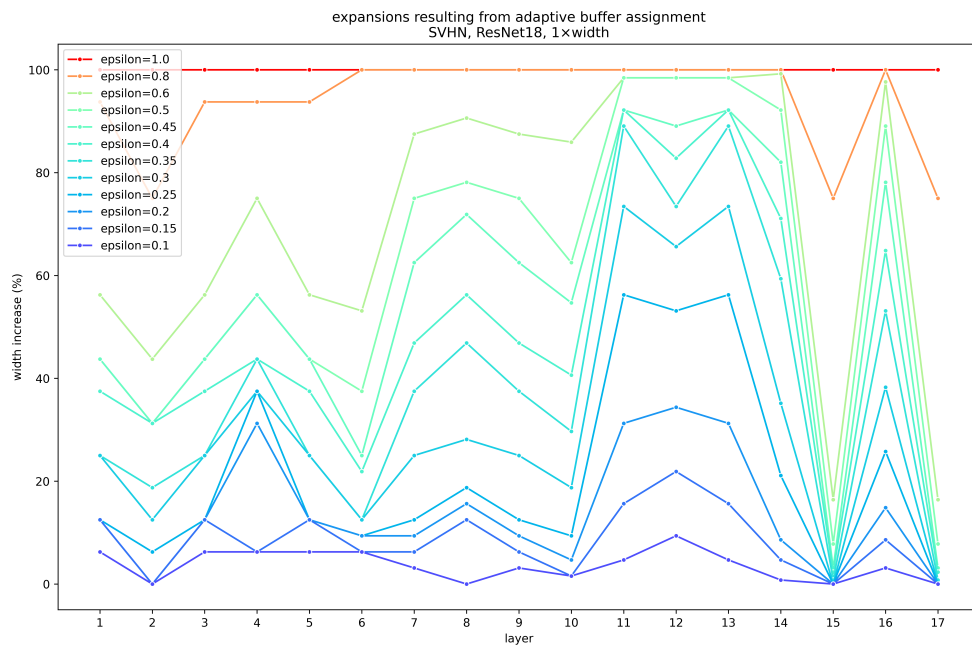


Figure 4.18: The resulting width expansion per layer for different values of ϵ for adaptive buffer assignment in two ResNet18s trained on SVHN.

4 Experiments

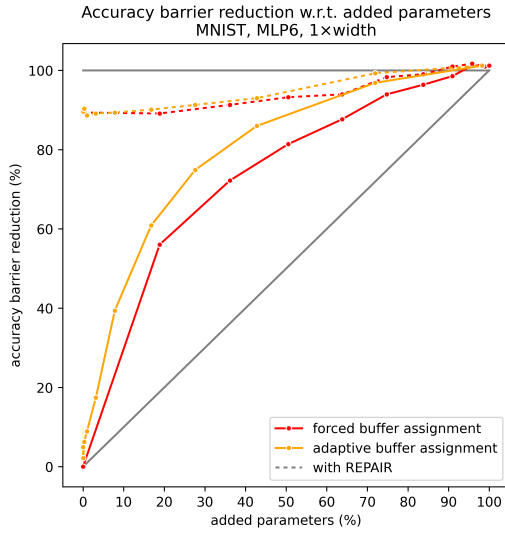


Figure 4.19: Added non-zero parameter count vs. accuracy barrier reduction, for forced and adaptive buffer assignment on two 6-layer MLPs trained on MNIST.

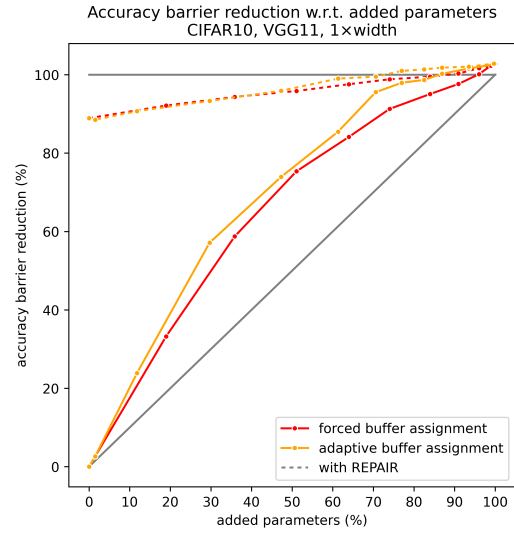


Figure 4.20: Added non-zero parameter count vs. accuracy barrier reduction, for forced and adaptive buffer assignment on two VGG11s trained on CIFAR10.

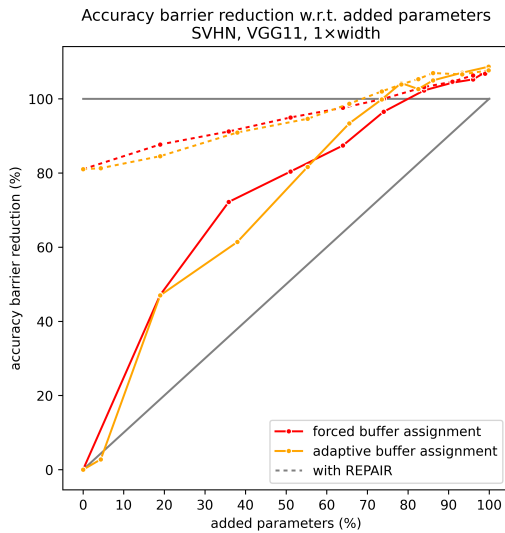


Figure 4.21: Added non-zero parameter count vs. accuracy barrier reduction, for forced and adaptive buffer assignment on two VGG11s trained on SVHN.

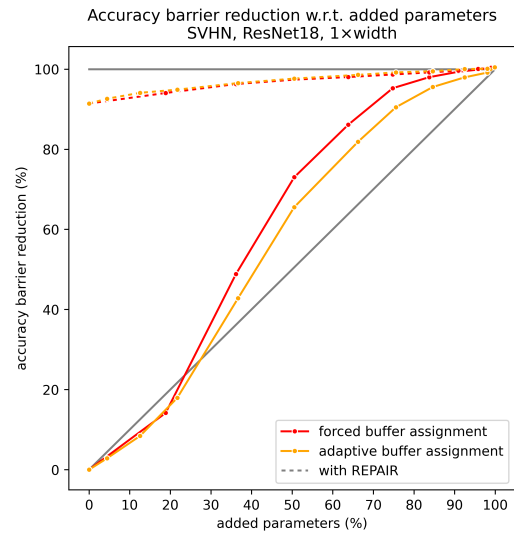


Figure 4.22: Added non-zero parameter count vs. accuracy barrier reduction, for forced and adaptive buffer assignment on two ResNet18s trained on SVHN.

4.6 Split data training

Following the split data training experiment from [7], we construct two disjoint subsets of CIFAR100 [45], CIFAR100-*A* and CIFAR100-*B*. Subset *A* contains 400 samples each of classes 0 to 49, and 100 samples each for classes 50 to 99, representing a 80%/20% split, with the exact opposite for subset *B*. While the original experiment uses two ResNet20s with $32\times$ width, this is computationally infeasible for us. Instead, we will evaluate two ResNets with $16\times$ width², which should not yield significantly different results.. When calculating the batch norm statistics for REPAIR, we will use the entire unbiased CIFAR100 dataset³.

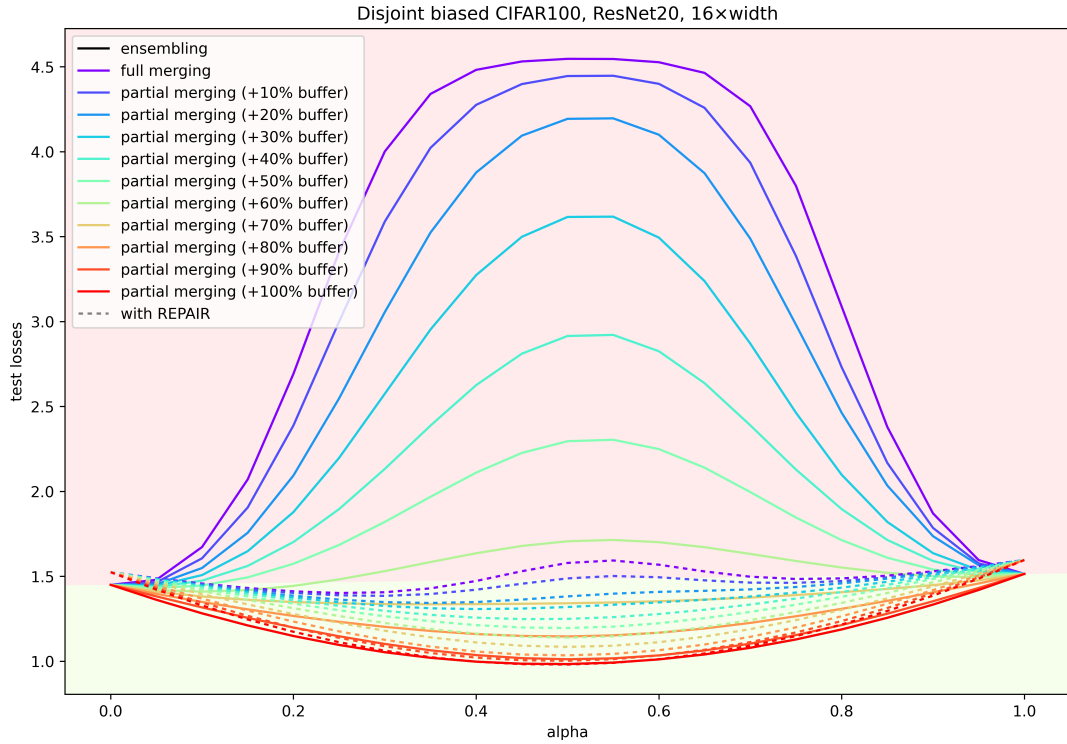


Figure 4.23: The resulting test losses from the interpolation of the biased endpoint models.

²Even this required access to an A100 40GB GPU. A $16\times$ width ResNet20 has 256 times as many parameters as a single-width one.

³The authors of [10] did not specify which data they used for REPAIR, hence this is just our best guess.

4 Experiments

The resulting test losses can be seen in Figure 4.23. A complete elimination of the loss barrier can be achieved beyond $\gamma = 0.7$ and beyond $\gamma = 0.2$ when using REPAIR. The REPAIred endpoints have a slightly higher loss than before due to the different distribution of our unbiased REPAIR data. However, for the interpolation at $\alpha = 0.5$ this is not harmful. The results for test accuracy can be seen in Appendix Figure A.6. The elimination of the accuracy barrier occurs roughly at the same values of γ . These results are relevant for federated learning cases in which the data distribution for each client has a different makeup.

In addition to the just reproduced experiment, we also construct two unbiased disjoint subsets of CIFAR10, CIFAR10-*A* and CIFAR10-*B*, both of which contain exactly half of the samples per class, and train a single-width ResNet18 on each subset. We do the same with single-width VGG11s. We then evaluate forced buffer assignment in this split data setting that is supposed to approximate federated learning with a balanced data distribution across clients. The resulting accuracies and losses for the ResNet18s can be seen in Figure 4.24, with the same plots for the VGGs in Appendix Figure A.7.

The overall pattern of gradual improvement through partial merging persists even in the split data setting, albeit with an understandably lower test accuracy of the endpoints, the ensemble, the fully merged model and all partial merging steps in between.

4 Experiments

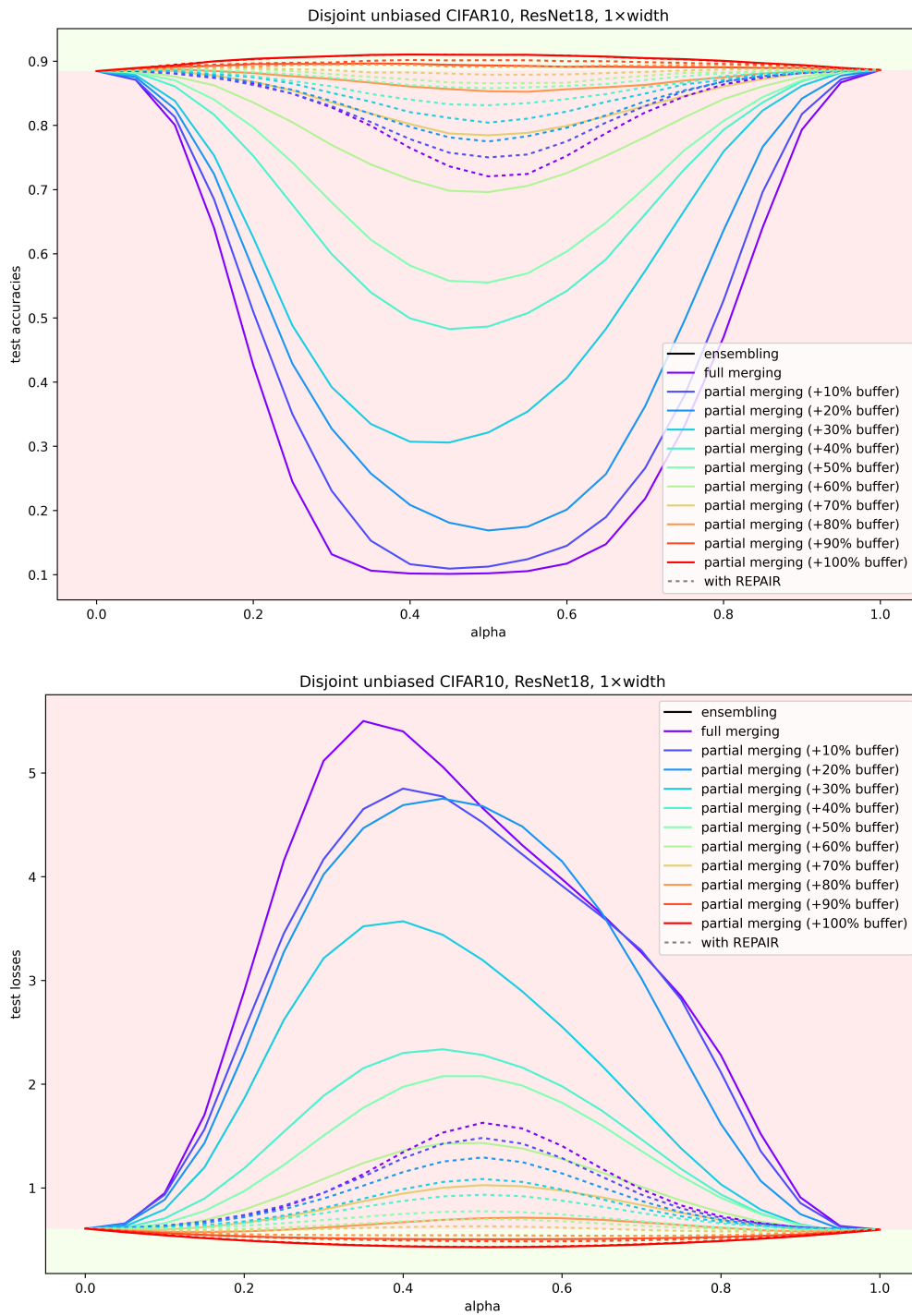


Figure 4.24: Test accuracies and losses with and without REPAIR for two regular-width ResNet18s trained on unbiased disjoint subsets of CIFAR10.

4.7 Finetuned models

Model merging can be beneficial for the test accuracy of the interpolated model when the endpoint models were finetuned from a shared base model, a property exploited by model soups [6]. To determine whether partial merging provides any additional benefit in this regime, we prepare a simple two-model-soup following the uniform soup recipe from [6].

Starting from a fully trained CIFAR10 VGG11 endpoint model from the previous experiments, we create two copies of the model and further train each copy for 50 epochs using a cosine learning rate of 0.04 and the regular CIFAR10 dataset. The finetuning procedure for both models differs only in the training sample order and data augmentations, which are both sampled at random.

The resulting accuracies and losses for different values of γ and α can be seen in Figure 4.25. Both finetuned endpoints reached a slightly different final accuracy and loss after 50 finetuning epochs. REPAIR and partial merging are both successful in improving the loss of the combined models, with the REPAIR loss being significantly lower than without REPAIR. Both ensembling and merging also result in an accuracy increase that peaks at approximately $\alpha = 0.3$. However, despite the benefits in loss, partial merging results in no additional benefit in terms of accuracy, full merging already surpasses the accuracy of ensembling. REPAIR made the accuracy of models worse across all values of γ .

4 Experiments

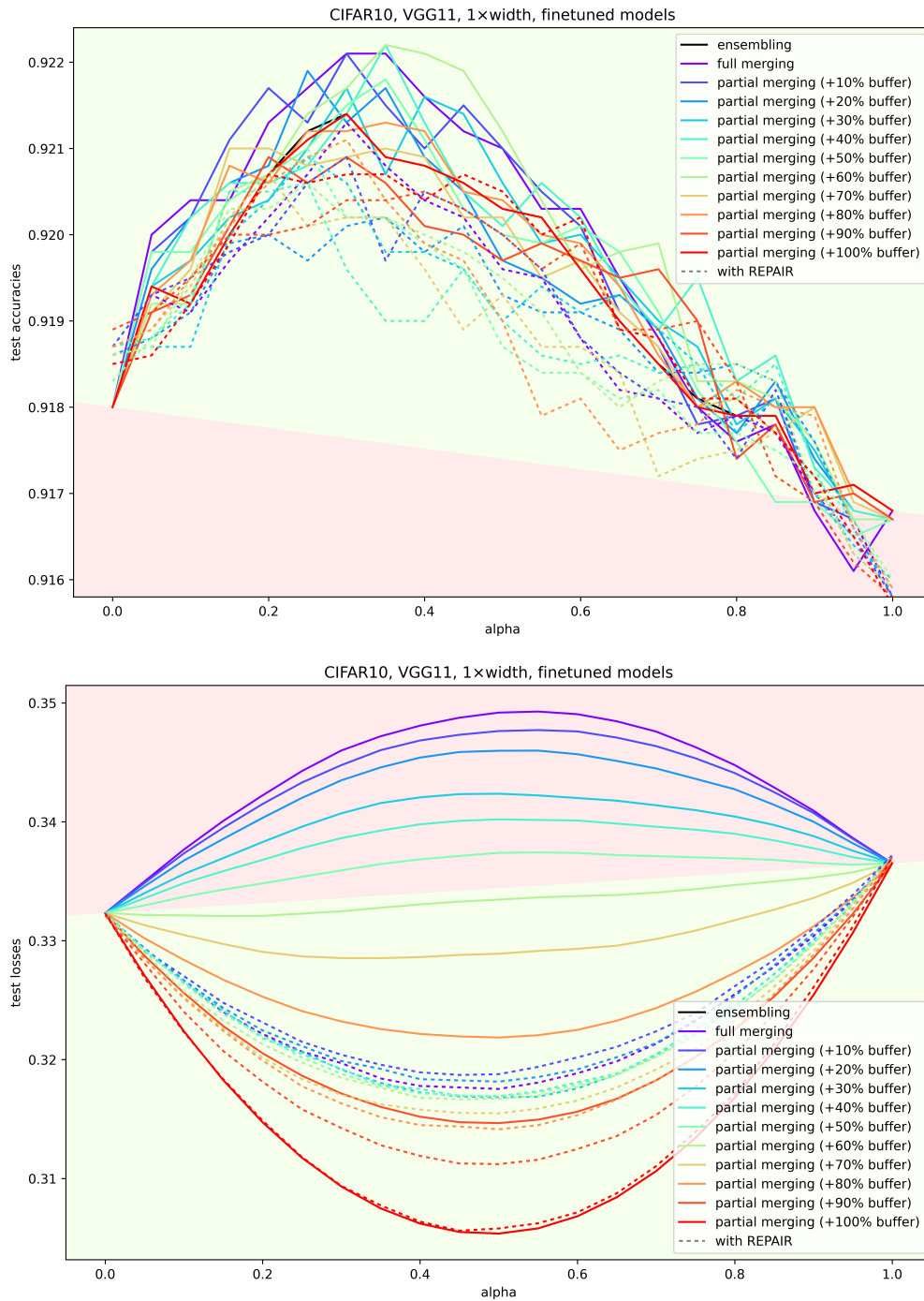


Figure 4.25: The test accuracies and losses from two single-width VGG11 models finetuned on the same base model. The ensembling line is hidden behind the partial merging (+100% buffer) line.

4.8 Partially merging single layers

The observation that the individual layers of the models can have vastly different correlations and are merged at different rates when using an adaptive strategy raises the question whether specific layers are responsible for different portions of the barriers incurred from merging. This question is especially relevant when individual layers have different widths, as is the case in convolutional architectures, and thus are responsible for a different portion of the parameter increase in the model when being partially merged. More specifically, are those layers that have the lowest correlation coefficients also the layers that yield the biggest benefit when being only partially merged? And, secondly, can we achieve a better parameter increase vs. barrier reduction tradeoff when we first partially merge layers that add only few parameters when being pulled apart?

We answer this question by manually defining γ as 0 for all layers except one, where we evaluate different values of γ from 0.1 to 1.0. This means fully merging the entire model, except one layer, which will have a width increase between 10% and 100%⁴. We repeat this for every layer of the model and record the resulting accuracy and loss barrier decreases and non-zero parameter increases, both before and after applying REPAIR. We do this for single-width VGG11s and ResNet18s trained on CIFAR10 and SVHN.

As an example, we plot the resulting de- and increases for the VGG11s trained on CIFAR10, evaluated before applying REPAIR, in Figure 4.26. The same values after applying REPAIR can be seen in Figure A.9, the values for VGG11s trained on SVHN in Figure A.8, and the results for the ResNet18s trained on CIFAR10, evaluated after REPAIR, in Figure A.10, all to be found in the Appendix.

The results are highly surprising, as they contradict the established notion that the correlation coefficients are a good predictor for the resulting barrier decrease. For example, while the correlation coefficients selected by the LAP solver in the VGG11s trained on CIFAR10 are the lowest in convolutional layer 5 (see Figure 4.16), an expansion of layer 5 does not yield the biggest barrier decrease (neither in test accuracy nor loss). In fact, even expanding layer 2 provides a bigger barrier

⁴What we do is effectively forced buffer assignment, but restricted to one layer.

4 Experiments

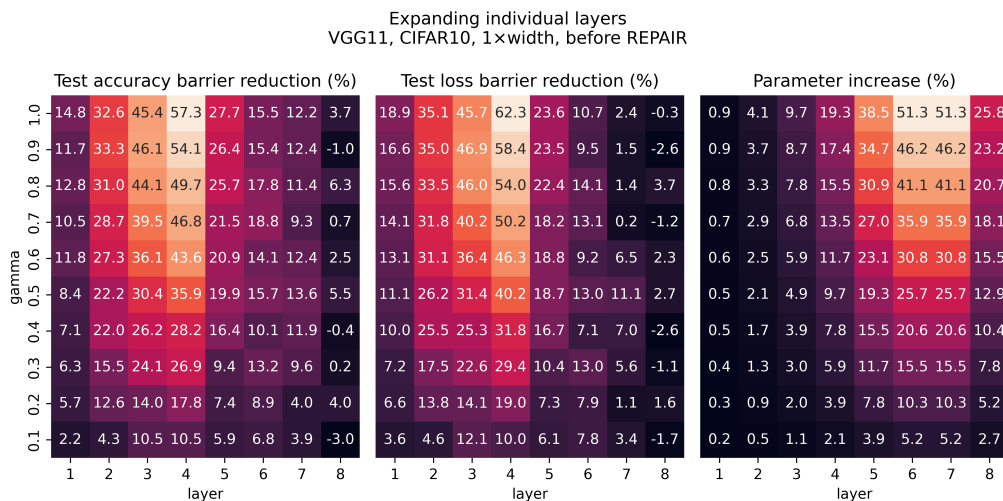


Figure 4.26: The accuracy and loss barrier reductions and (non-zero) parameter increases when only one layer is partially merged on a regular-width VGG11 trained on CIFAR10.

decrease than expanding layer 5, despite having much better correlation coefficients and being four times less wide (i.e. having 128 instead of 512 kernels). Not only does it provide a bigger barrier decrease, but it also costs us just nearly 10 times less to expand layer 2 in terms of added non-zero parameters⁵. This disparity becomes more pronounced when looking at the barrier decreases after REPAIR (see Figure A.9), where pulling apart earlier layers becomes even more favorable. In ResNets, the expansion of even layers is most beneficial in the earlier layers, and the expansion of the residually connected odd layer groups is most beneficial in the beginning and end of the model. Unsurprisingly, the expansion of layer groups also tends to be more parameter-expensive than the expansion of individual layers.

We can use the just collected statistics to construct a partial merging strategy that is not informed by correlation coefficients, but empirical observations about

⁵It should be noted that the added non-zero parameters are interdependent between layers when expanding more than one layer. For example, if layer ℓ has already been expanded, additionally expanding layer $\ell - 1$ or $\ell + 1$ is cheaper than expanding any other layer, even if they have the exact same width. Similarly, and as the only potential exception to this rule, it is especially cheap to expand the first or last layer, as the shape of the inputs and output classes remains constant, and thus limits the shifting of weight matrices to one dimension.

4 Experiments

the barrier reductions resulting from specific layers. For each layer, we calculate a *benefit factor*, which is nothing more than the accuracy barrier reduction at $\gamma = 1.0$, divided by the parameter increase at $\gamma = 1.0$, for each layer. We calculate this benefit factor both from the barrier decreases recorded before and after applying REPAIR to determine which of them are better suited as guiding metric. Starting from a value of $\gamma = 0$ for each layer, we consecutively assign $\gamma = 1.0$ to one layer after another, ordered by their benefit factor. We must stress that this strategy is incredibly simplistic, and assumes that the benefit and parameter increase of each layer is independent of other layers. Both of these assumptions are wrong, and as such the strategy can only serve as a proof-of-concept.

The resulting accuracy barrier reductions, plotted against their parameter increase, are shown in Figure 4.27 to 4.30. Equivalent plots for the loss barrier reductions can be seen in Figure A.11 to A.14 in the Appendix.

Our empirically grounded proof-of-concept baseline outperforms the forced buffer assignment (and adaptive buffer assignment) significantly. For example, an addition of between 22% and 24% non-zero parameters is able to eliminate the accuracy and loss barrier between the two VGG11s trained on CIFAR10 or SVHN completely when combined with REPAIR. An addition of 7% and 9% of non-zero parameters is sufficient to half the remaining accuracy barrier after REPAIR in the ResNets trained on CIFAR10 and SVHN. Unfortunately, a full barrier elimination can still only be achieved for large parameter increases on the ResNets using our proof-of-concept strategy.

4 Experiments

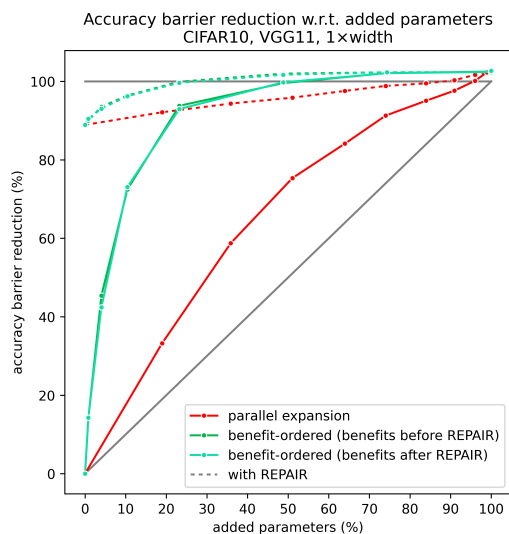


Figure 4.27: Parallel expansion compared to the strategies using the pre-REPAIR and post-REPAIR benefit factors w.r.t. accuracy, on VGG11s trained on CIFAR10.

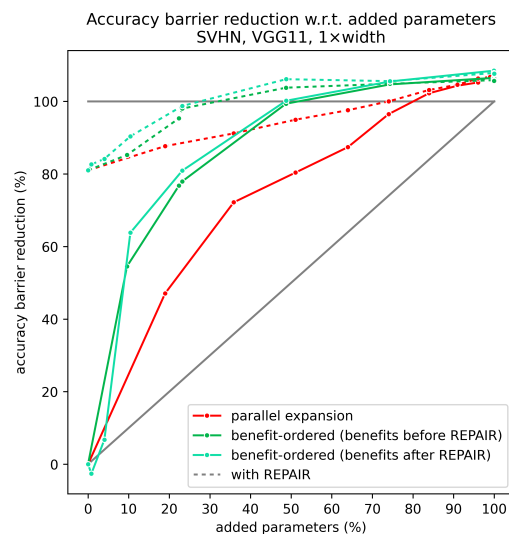


Figure 4.28: Parallel expansion compared to the strategies using the pre-REPAIR and post-REPAIR benefit factors w.r.t. accuracy, on VGG11s trained on SVHN.

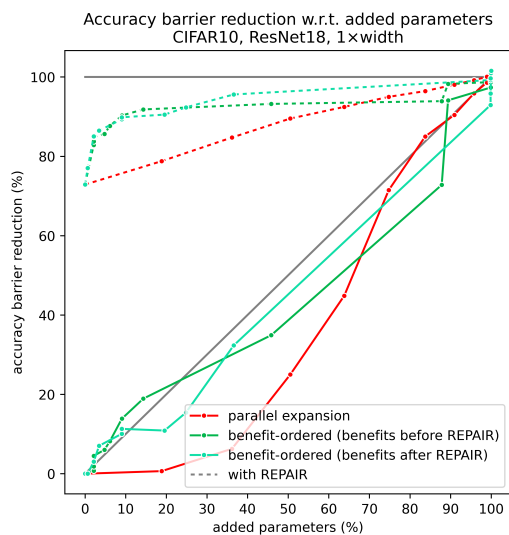


Figure 4.29: Parallel expansion compared to the strategies using the pre-REPAIR and post-REPAIR benefit factors w.r.t. accuracy, on ResNet18 trained on CIFAR10.

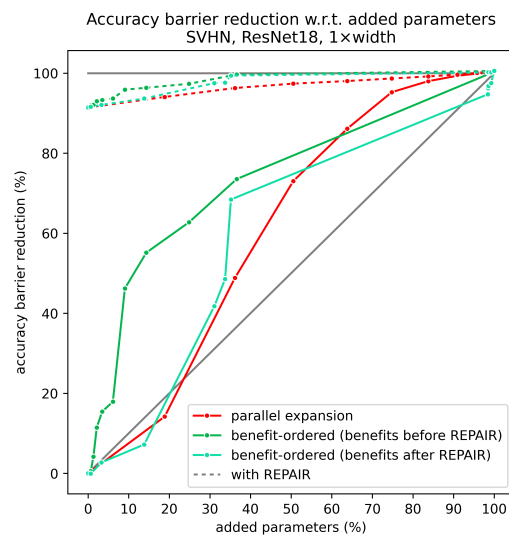


Figure 4.30: Parallel expansion compared to the strategies using the pre-REPAIR and post-REPAIR benefit factors w.r.t. accuracy, on ResNet18s trained on SVHN.

5 Discussion

The insights that we can gain from the conducted experiments are manifold. First and foremost, partial merging always allows us to gradually bridge the performance gap between full merging and ensembling, except when no such gap exists (as is the case for the finetuned models). There was no instance in which allowing a partial overlap was detrimental to either test accuracy or test loss. Additionally, the benefits we can reap through even a small expansion (e.g. +20% width) are major, often reducing the remaining post-REPAIR barrier by half or more.

In typical ensembling use-cases, any performance drop is undesirable. It would be much wiser to simply use the best individual model, for no extra cost. However, specifically in those cases where ensembling is impossible, e.g. when doing federated learning, even the slightest barrier reduction could already be useful. If the performance drop that we get from (partially) merging the models from two clients is only half as big, unnecessary communication rounds can be avoided. Our partial merging method can easily be adapted to work for the strict data-agnostic federated learning regime by using weight matching instead of activation matching and using the data-free (yet slightly less performant) variant of REPAIR that approximates the necessary adjustments against variance collapse from the existing batch norm statistics alone [10]. If even the smallest resulting width increase is problematic, a possible solution could be to train the client models narrower than the desired final width in the first few rounds, then partially merge and repair them, and fall back to regular FedAvg training afterwards.

Another interesting property that we discovered but did not include in our experimental section is that narrower models can be merged into wider models with much greater ease than narrower models with each other, and even better than wider models with each other. This could enable federated learning scenarios in

5 Discussion

which the number of clients changes late in training, and a new narrow model from the additional client could be more easily integrated into a much wider existing model, potentially requiring no additional width expansion at all. As we did not evaluate partial merging of more than two models, we leave these questions for future research.

Likely the most relevant finding of our experiments is the independence of LAP-selected correlation coefficients and resulting barrier decrease. This finding has implications for all strategies that use these measures of similarity as decision criteria for which units to split up and which to keep together, such as PFNM and FedMA. While our benefit-factor-based method was just a proof-of-concept, it dramatically outperforms uniformly expanding across all layers or expanding layers with bad correlations more. We can think of immediately obvious improvements to our method. The first improvement would be to re-calculate the cost of expansion, that is the increase in parameters, after each expansion step. As the non-zero parameter increase depends on the shifting of the weight matrices in the direction of rows and columns, the shape of which is in return determined by preceding or following layers, the recalculation could make low-cost expansions more obvious. For example, when ResNet18 layer group $\{15, 17\}$ is already expanded by $\gamma = 1.0$, the expansion of layer 16 is practically free¹. This could improve the performance of our method considerably, especially for ResNets. As the re-calculation of parameter increases does not require any data, it will also not slow down our method significantly.

The second improvement to our proof-of-concept method is slightly more compute-intensive. Instead of expanding each layer completely from $\gamma = 0$ to $\gamma = 1$, we can use gradual expansions as intermediary steps. For example, it is visible in Figure 4.26 that the accuracy barrier decrease when expanding layer 3 by 50% is already 67% of the barrier decrease we get when expanding it by 100%, echoing the non-linear relationship observed in our forced buffer assignment experiments in single layers. It could thus be useful to expand each layer step-wise, re-evaluate the benefits, and then potentially choose a different layer to expand a bit more. The implementation of this improvement could be done using a greedy flood-fill

¹The expansion only results in a doubling of bias and batch norm parameters, but no changes to the number of non-zero weights in layer 16.

5 Discussion

algorithm without much changes to the existing code. However, the repeated recalculation of barrier decreases for each additional expansion is quite compute-heavy, as previous barrier decrease values cannot be reused.

Finally, one of the properties of CNNs not considered by our proof-of-concept method is the fact that kernels in earlier layers are applied more often than kernels in the later layers, due to the different x, y dimensions of the feature maps. Each max pooling layer reduces the number of expected kernel applications. However, this would only change our parameter increase to a weighted parameter increase, modifying our benefit factors, but change little about the overall distribution of costs and benefits. This adjustment to account for expected forward-pass FLOPs instead of parameters would optimize for the metric we actually care about, inference time, and give less weight to the absolute number of parameters.

As the method still need a good amount of parameters (approximately a quarter) to reach zero-barrier connectivity between the endpoint models, it only becomes useful compared to individual endpoints after that. However, often a performance almost as good as ensembling can be achieved with less than 100% parameter increase, for example in the VGG11s and ResNet18s trained on SVHN, for which the accuracy barrier reduction is more or less flat after $\gamma = 0.5$. If we are also okay with 80% of the performance increase we get from ensembling, we can thus save a large number of parameters compared to executing both models in parallel.

As stated in the original REPAIR paper, “any improvements towards reducing the obstacles to interpolation between trained models has the potential to lead to empirical progress” [10] in areas where the combination of multiple models parameters is of relevance. Our results are an incremental contribution to making the interpolation of models more efficient, and can represent a viable alternative to ensembling in some cases.

Further avenues for future research include pruning the models before partially merging them, which could result in a final partially merged model of the same with as the originally unpruned endpoint models. Furthermore, the combination of more advanced alignment methods such as Sinkhorn-Rebasin [35] for partial merging could boost the barrier decrease even further, just like it did for full merging.

6 Conclusion

In this thesis we explored several methods for merging models just in parts. Forced buffer assignment, which expands all layers equally, and adaptive buffer assignment, which expands each layer by a different factor depending on the measured correlation coefficients, performed approximately equally well. Both were able to gracefully bridge any occurring loss and accuracy barriers and converged to the performance of ensembling as γ approaches 1.0. Often a width increase by a factor of x results in a barrier reduction larger than x , which is especially true after applying REPAIR.

We were able to demonstrate that those layers that have lower correlation coefficients are not necessarily the layers that benefit the most from being expanded. Instead, the contribution of each layers expansion towards the reduction of the accuracy and loss barriers should be measured empirically. Doing this allows for selecting expansion orders which reach zero-barrier connectivity between endpoints earlier while at the same time incurring a smaller parameter increase in the partially merged model.

We hope that our results can be used as a starting point for constructing even more efficient merging methods, and move model merging further towards the performance-boosting regime that ensembling resides in.

A Additional tables and plots

		<i>MNIST</i>							
		MLP3	MLP4	MLP5	MLP6	MLP7	MLP8	MLP9	MLP10
0.125×		98.9%	98.7%	98.8%	98.7%	98.7%	98.7%	98.7%	98.7%
		0.040	0.047	0.047	0.048	0.047	0.051	0.053	0.053
0.25×		99.0%	99.0%	98.9%	98.9%	99.0%	99.0%	98.9%	99.0%
		0.036	0.039	0.041	0.045	0.043	0.043	0.046	0.047
0.5×		99.0%	99.1%	99.2%	99.1%	99.2%	99.2%	99.1%	99.1%
		0.032	0.032	0.033	0.037	0.036	0.038	0.046	0.045
1×		99.1%	99.2%	99.2%	99.2%	99.2%	99.1%	99.2%	99.1%
		0.029	0.028	0.030	0.035	0.035	0.039	0.037	0.044
2×		99.2%	99.2%	99.2%	99.3%	99.2%	99.2%	99.2%	99.2%
		0.028	0.028	0.030	0.034	0.034	0.037	0.039	0.039
4×		99.2%	99.3%	99.3%	99.2%	99.3%	99.2%	99.2%	99.2%
		0.028	0.029	0.031	0.036	0.035	0.040	0.039	0.040
8×		99.2%	99.3%	99.3%	99.2%	99.2%	99.2%	99.2%	99.3%
		0.029	0.030	0.034	0.038	0.042	0.042	0.041	0.040

Table A.1: Average test accuracy and test loss of both MLP endpoint models trained on MNIST for different depths and widths.

A Additional tables and plots

		<i>CIFAR10</i>				<i>SVHN</i>			
		VGG11	VGG13	VGG16	VGG19	VGG11	VGG13	VGG16	VGG19
0.25×		87.1%	–	–	–	95.6%	–	–	–
		0.460	–	–	–	0.196	–	–	–
0.5×		90.1%	–	–	–	96.1%	–	–	–
		0.408	–	–	–	0.175	–	–	–
1×		91.5%	93.3%	93.3%	92.9%	96.6%	96.8%	96.6%	96.7%
		0.336	0.277	0.305	0.347	0.146	0.144	0.167	0.181
2×		92.5%	–	–	–	96.7%	–	–	–
		0.299	–	–	–	0.141	–	–	–
4×		92.6%	–	–	–	96.5%	–	–	–
		0.280	–	–	–	0.141	–	–	–

Table A.2: Average test accuracy and test loss of both VGG endpoint models trained on CIFAR10 and SVHN for different depths and widths.

		<i>CIFAR10</i>	<i>SVHN</i>
		ResNet18	ResNet18
1×		92.0%	96.7%
		0.397	0.188
2×		93.6%	96.9%
		0.294	0.158
4×		94.8%	97.0%
		0.235	0.145
8×		95.3%	97.0%
		0.209	0.141

Table A.3: Average test accuracy and test loss of both ResNet18 endpoint models trained on CIFAR10 and SVHN for different widths.

A Additional tables and plots

		<i>SVHN</i>			
		VGG11	VGG13	VGG16	VGG19
0.25×		17.6%p			
		0.577			
		3.1%p (-82.7%)	–	–	–
		0.111 (-80.7%)			
		-0.7%p (-104.1%)			
0.5×		-0.037 (-106.5%)			
		9.5%p			
		0.406			
		1.2%p (-87.6%)	–	–	–
		0.043 (-89.3%)			
1×		-0.5%p (-104.9%)			
		-0.024 (-105.9%)			
		4.3%p	7.0%p	24.8%p	17.7%p
		0.236	0.292	0.828	0.655
		0.8%p (-81.0%)	0.7%p (-90.6%)	0.6%p (-97.5%)	0.6%p (-96.4%)
2×		0.023 (-90.1%)	0.017 (-94.0%)	0.018 (-97.8%)	0.016 (-97.6%)
		-0.3%p (-107.5%)	-0.3%p (-104.7%)	-0.4%p (-101.8%)	-0.5%p (-102.6%)
		-0.015 (-106.5%)	-0.016 (-105.6%)	-0.026 (-103.1%)	-0.034 (-105.2%)
		3.6%p			
		0.165			
4×		0.6%p (-84.5%)	–	–	–
		0.014 (-91.7%)			
		-0.2%p (-106.7%)			
		-0.011 (-107.0%)			
		1.6%p			
4×		0.098			
		0.1%p (-93.0%)	–	–	–
		0.001 (-99.9%)			
		-0.3%p (-118.8%)			
		-0.009 (-109.1%)			

Table A.4: Black: Absolute test accuracy barrier (top) and test loss barrier (bottom) of fully merged VGG models trained on SVHN for different depths and widths. Purple: The respective absolute barrier (and relative reduction to full merging) after additionally applying REPAIR. Grey: The respective absolute barrier (and relative reduction to full merging) when using ensembling instead.

A Additional tables and plots

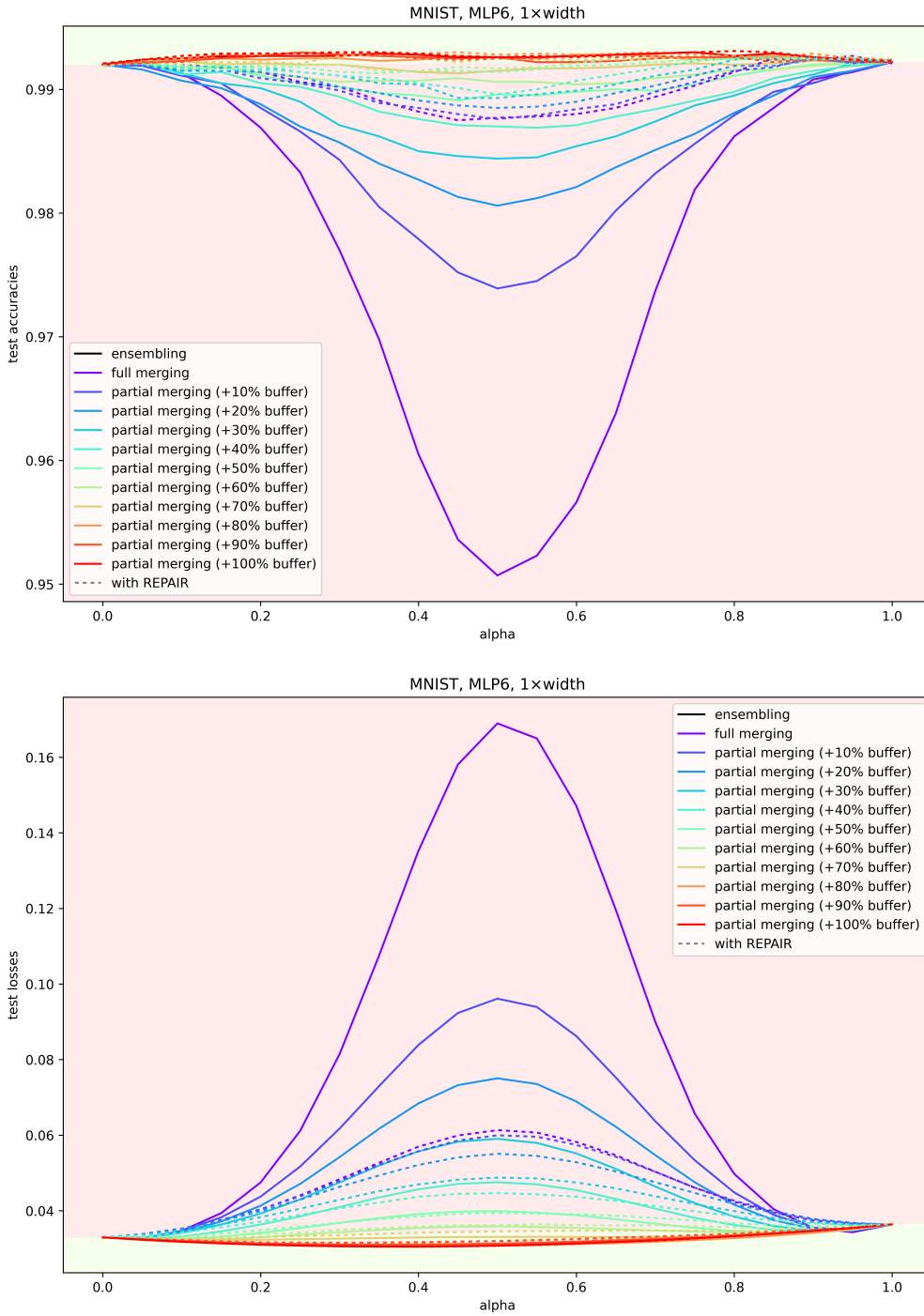


Figure A.1: Test accuracies and losses with and without REPAIR for a regular-width MLP with 6 layers. An improvement in accuracy and loss compared to the endpoint model average (green area) is achieved beyond an addition of 90% buffer neurons to the original layer width and beyond 70% when using REPAIR.

A Additional tables and plots

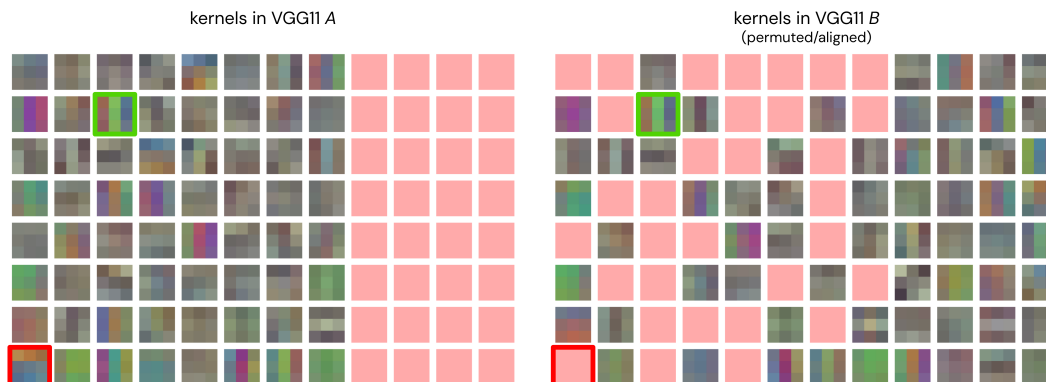


Figure A.2: Aligned first-layer kernels of two VGG11 models trained on CIFAR10, identical to Figure 3.1. In green: a good match. In red: a (previously) bad match that is now separated and will not be interpolated. Kernels shaded in red indicate buffer kernels whose parameters are all zero. Kernel saturation increased for better visibility.

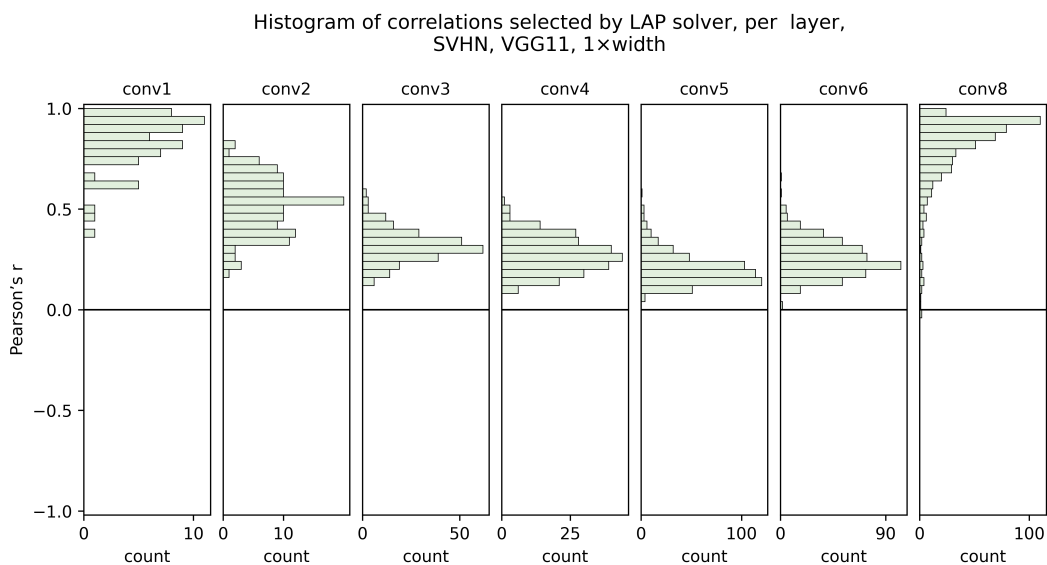


Figure A.3: The activation correlations chosen by the LAP solver when fully merging two VGG11s trained on SVHN, per convolutional layer.

A Additional tables and plots

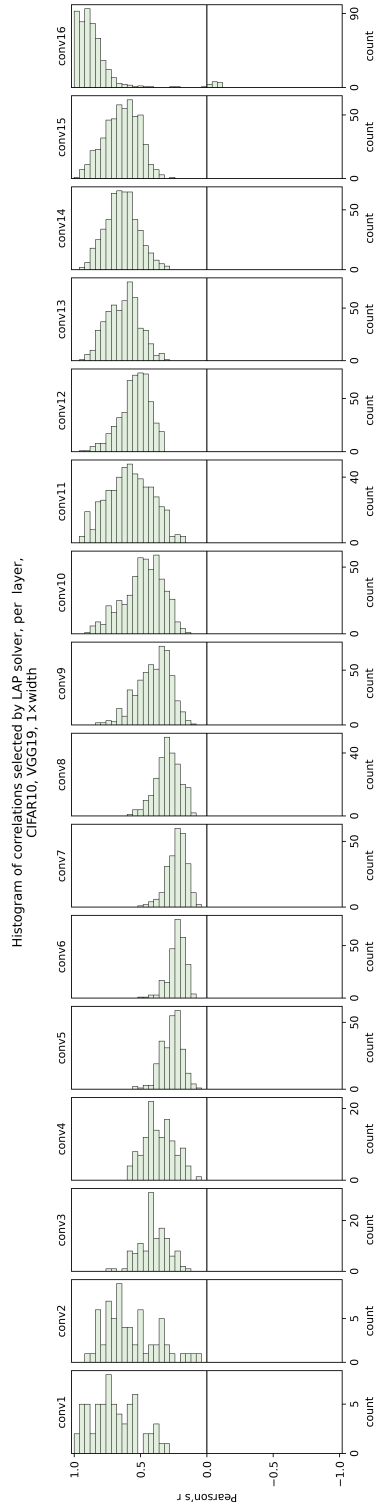


Figure A.4: The activation correlations chosen by the LAP solver when fully merging two VGG19s trained on CIFAR10, per convolutional layer.

A Additional tables and plots

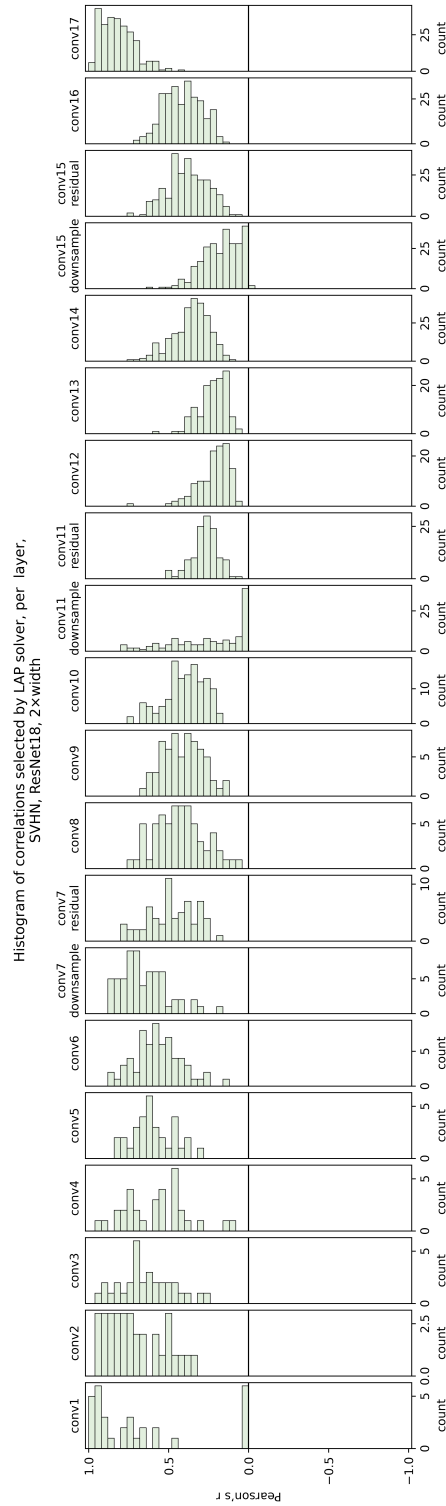


Figure A.5: The activation correlations chosen by the LAP solver when fully merging two $2\times$ width ResNet18s trained on SVHN, per convolutional layer. The correlations of the downsampling layers and residual layers are shown separately in this plot, but only their sum is used for activation matching. Bad correlations for downsampling activations imply that their magnitude was low, and that they therefore had little influence on the summed up activations used by the LAP solver.

A Additional tables and plots

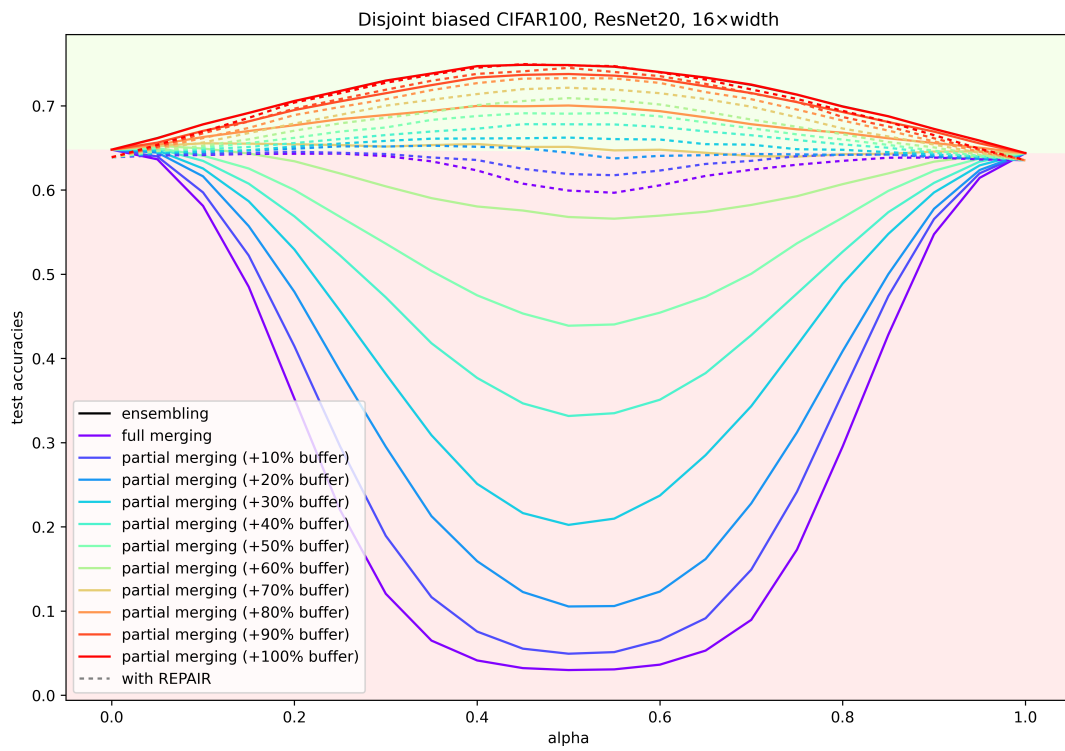


Figure A.6: The resulting test accuracies from the interpolation of the biased endpoint models.

A Additional tables and plots

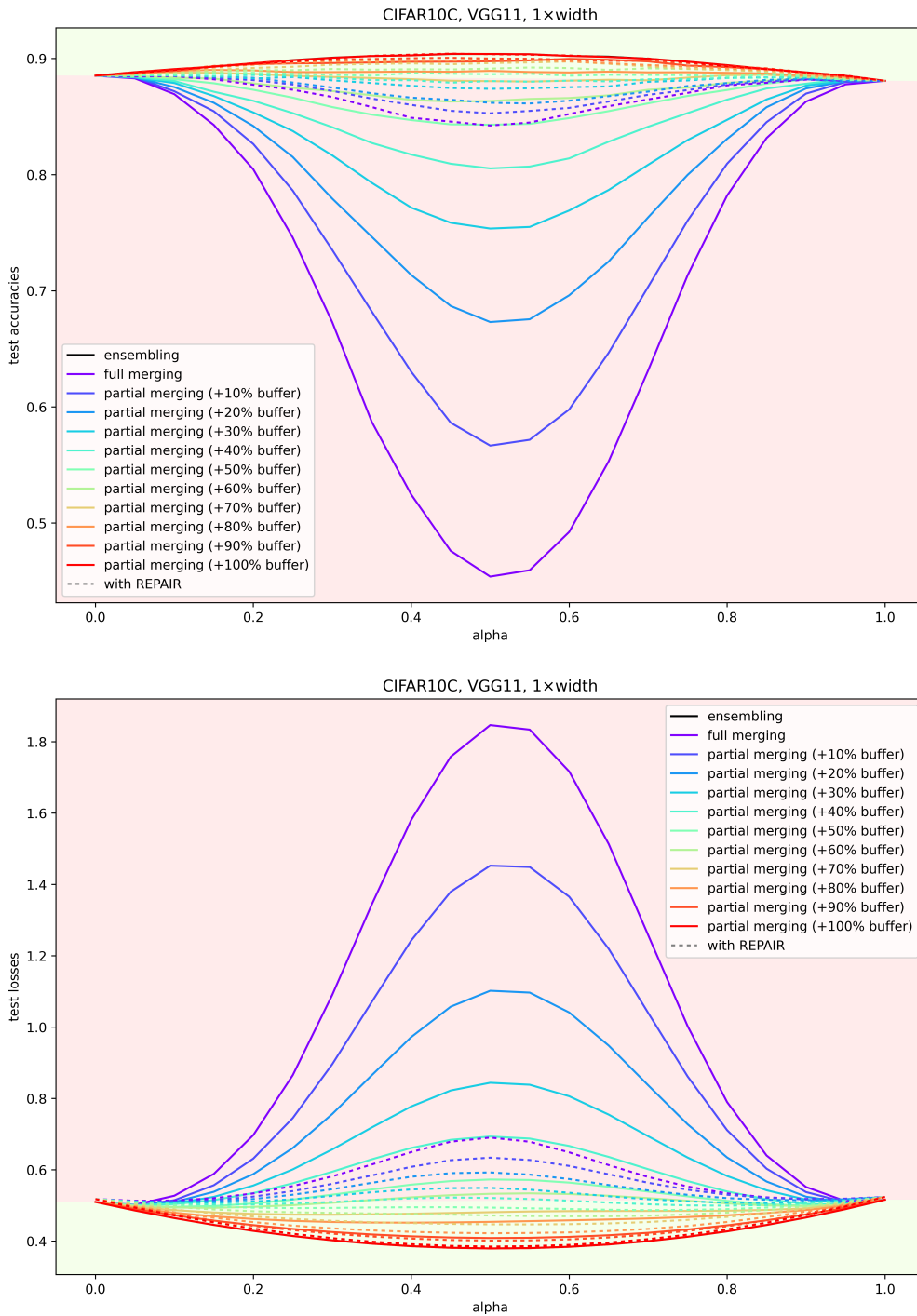


Figure A.7: Test accuracies and losses with and without REPAIR for two regular-width VGG11s trained on unbiased disjoint subsets of CIFAR10.

A Additional tables and plots

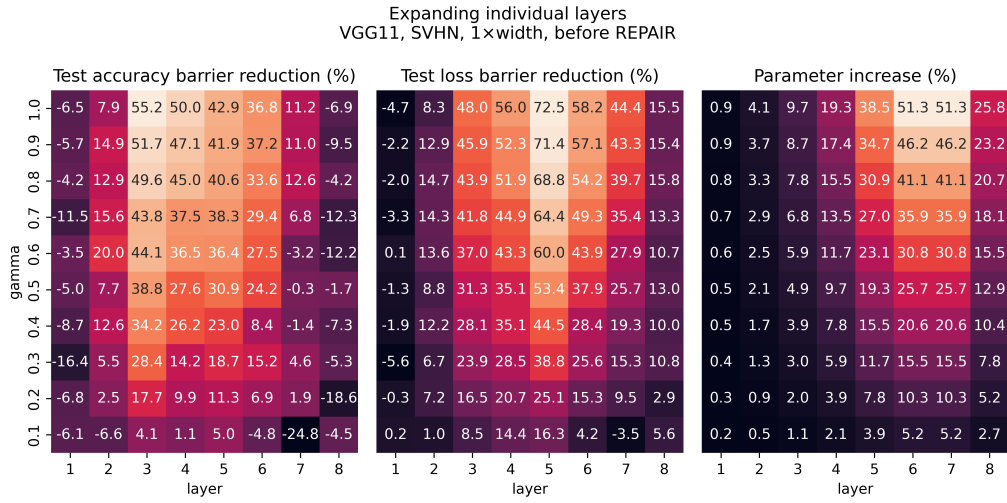


Figure A.8: The test accuracy, test loss, and (non-zero) parameter increase when only one layer is partially merged on a regular-width VGG11 trained on SVHN.

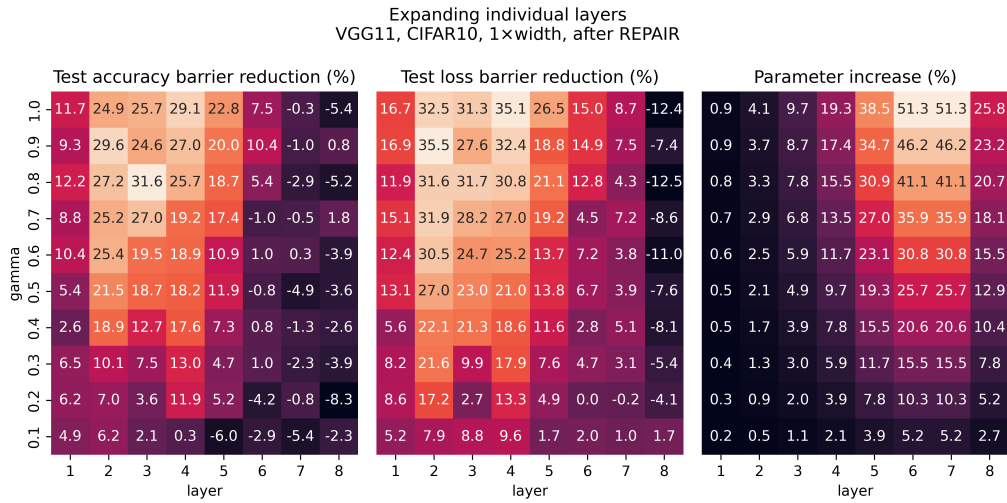


Figure A.9: The test accuracy, test loss, and (non-zero) parameter increase when only one layer is partially merged on a regular-width VGG11 trained on CIFAR10, recorded after applying REPAIR.

A Additional tables and plots

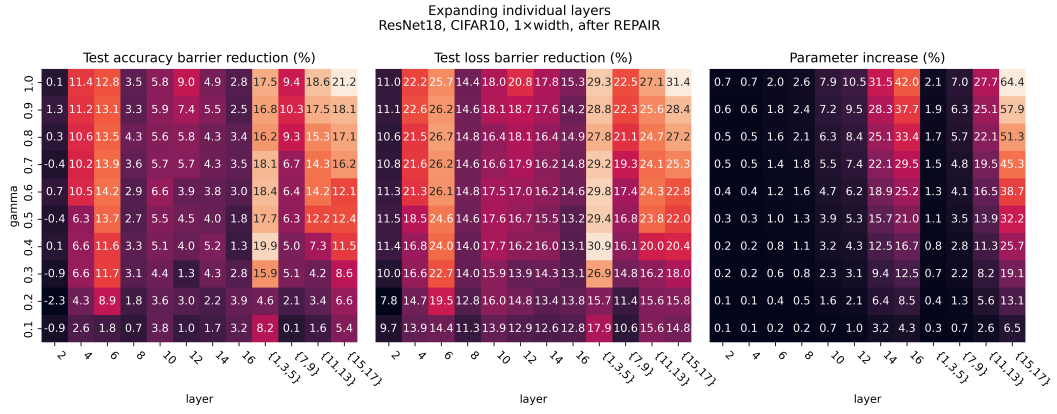


Figure A.10: The test accuracy, test loss, and (non-zero) parameter increase when only one layer is partially merged on a regular-width ResNet18 trained on CIFAR10, recorded after applying REPAIR.

A Additional tables and plots

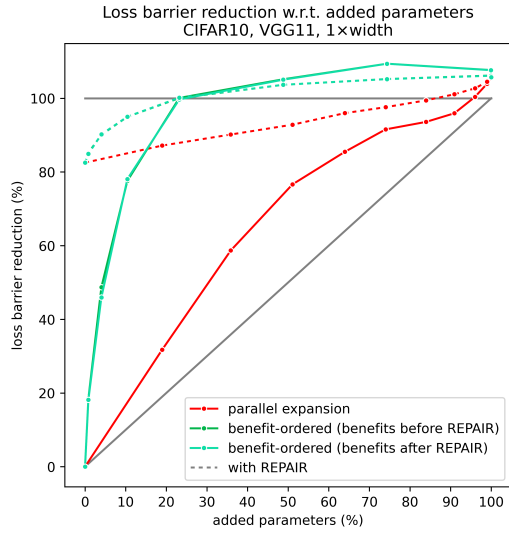


Figure A.11: Parallel expansion compared to the strategies using the pre-REPAIR and post-REPAIR benefit factors w.r.t. loss, on VGG11s trained on CIFAR10.

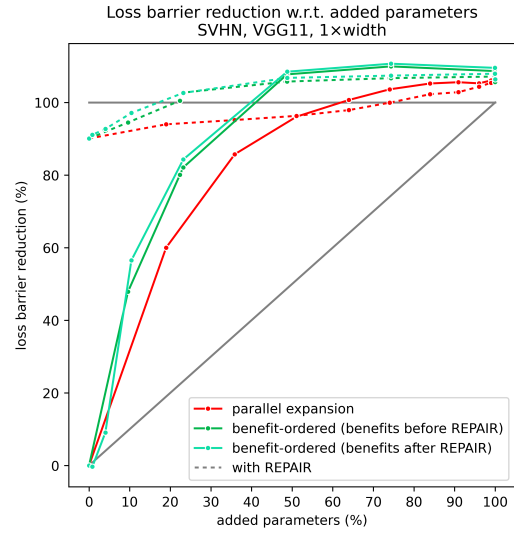


Figure A.12: Parallel expansion compared to the strategies using the pre-REPAIR and post-REPAIR benefit factors w.r.t. loss, on VGG11s trained on SVHN.

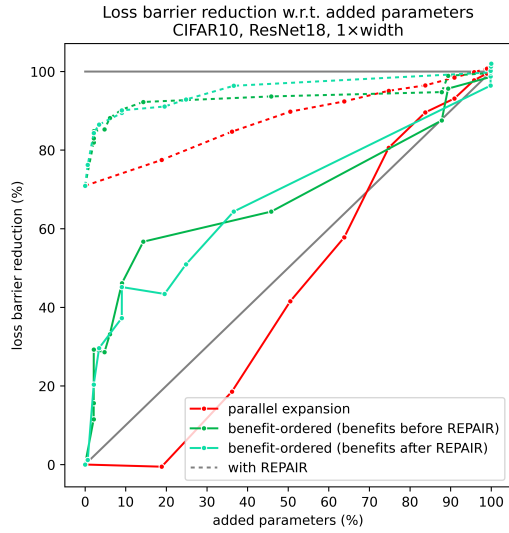


Figure A.13: Parallel expansion compared to the strategies using the pre-REPAIR and post-REPAIR benefit factors w.r.t. loss, on ResNet18s trained on CIFAR10.

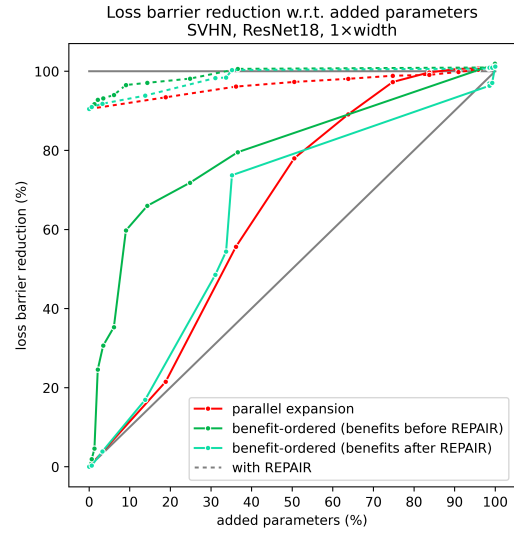


Figure A.14: Parallel expansion compared to the strategies using the pre-REPAIR and post-REPAIR benefit factors w.r.t. loss, on ResNet18 trained on SVHN.

B Weight matrix partitioning

As seen in Figure 3.4, the partial merging of originally dense weight matrices introduces sparsity in the output weight matrix, ranging from 0% (full merging) to 50% (100% width increase) sparsity or zero-weights¹. However, whenever we talk about induced parameter increase in this thesis, we are speaking of the increase of non-zero parameters only, and ignore the added zeros. Is this a fair approach? In this appendix we show why the added zero parameters are harmless, both in terms of model storage size and floating-point operations (FLOPs) per kernel multiplication, and can be ignored.

If no additional steps are undertaken and the resulting sparse weight matrices are kept as-is, partial model merging results in a unnecessary high storage cost and computation effort. While efficient implementations for multiplying sparse matrices for popular tensor frameworks like PyTorch are currently being developed², we can also manually modify our model architecture in such a way that the introduced zero parameters do not result in any additional overhead. We achieve this through subsequent permutation of the final merged model in such a way that all interpolated input and output units are clustered together³. Next, the weight matrix is partitioned into three dense blocks (visible in Figure B.1), and the zero parameters are discarded.

The partitioning of the weight matrices necessitates an adapted forward pass procedure, which takes the shapes of the weight matrix blocks into account. When before, we had a weight matrix $\mathbf{W}_\ell^{(\alpha)}$ of shape $n + \lceil \gamma n \rceil \times m + \lceil \gamma m \rceil$, with the

¹This sparsity occurs exclusively in the weight matrices, and not in bias or batch norm parameters, which are simple vectors.

²See <https://pytorch.org/docs/stable/sparse.html>.

³Note that the bias and batch norm parameters must be permuted appropriately using the same permutation or its transpose.

B Weight matrix partitioning

	in ₄	in ₂	in ₅	in ₆	in ₃	in ₁	in ₄	in ₅	in ₂
out ₃	-0.32	-0.70	-0.32	-0.42	-0.64	0.16	0	0	0
out ₂	0.58	-0.24	0.62	0.78	-0.26	0.42	0	0	0
out ₁	0.26	-0.84	0.30	-0.39	0.16	0.62	-0.90	0.96	-0.46
out ₂	0.58	-0.84	0.54	0.40	0.58	0.56	-0.80	0.20	0.94
out ₄	-0.22	-0.68	-0.86	-0.64	-0.18	0.07	0.92	0.08	-0.98
out ₃	0	0	0	0.40	0.22	-0.92	-0.48	0.52	-0.64
out ₅	0	0	0	0.18	0.46	0.96	-0.72	0.08	-0.04

Figure B.1: The sparse weight matrix from Figure 3.4, permuted and partitioned into three smaller dense weight matrix blocks outlined in blue.

following forward pass:

$$\mathbf{z}_{\ell+1} = \sigma \left(\mathbf{W}_{\ell}^{(\alpha)} \mathbf{z}_{\ell} + \mathbf{b}_{\ell} \right) \quad (\text{B.1})$$

we now have the following forward pass using our partitioned matrices:

$$\mathbf{z}_{\ell+1} = \sigma \left(\mathbf{W}_{\ell}^{([\gamma n]:m)^{(\alpha)}} \mathbf{z}_{\ell} + \mathbf{b}_{\ell} \oplus \mathbf{W}_{\ell}^{([\gamma m]:n)^{(\alpha)}} \mathbf{z}_{\ell} + \mathbf{b}_{\ell} \oplus \mathbf{W}_{\ell}^{(n;[\gamma m])^{(\alpha)}} \mathbf{z}_{\ell} + \mathbf{b}_{\ell} \oplus \right) \quad (\text{B.2})$$

where superscripts of type $a : b, c : d$ denote partitions from row a (excluding) to row b (including) and from column c (excluding) to column d (including), superscripts of type $a : b$ denote the equivalent logic for vectors, partitions from the beginning or to the end are expressed as ellipses (e.g. $:m$), and \oplus denotes concatenation. Note that for ease of notation, we assume that $\mathbf{W}_{\ell}^{(\alpha)}$, \mathbf{z}_{ℓ} , and \mathbf{b}_{ℓ} have already been permuted as described in the previous paragraph and visible in Figure B.1.

The partitioning procedure eliminates the need for storing the zero parameters and limits the FLOPs to only those operations absolutely necessary, i.e. those

B Weight matrix partitioning

involving non-zero parameters. Please note that it is not a novel contribution, but directly inspired by the optimization done in [47], where different parts of a network can be switched “on” or “off”, depending on the currently executed task.

Bibliography

- [1] B. Dasarathy and B. Sheela, ‘A composite classifier system design: Concepts and methodology’, *Proceedings of the IEEE*, vol. 67, no. 5, pp. 708–713, 1979. DOI: 10.1109/PROC.1979.11321 (cit. on p. 1).
- [2] Mistral AI, *Mixtral of experts: A high quality sparse mixture-of-experts*, Accessed: December 20th, 2023, Dec. 2023. [Online]. Available: <https://mistral.ai/news/mixtral-of-experts/> (cit. on p. 1).
- [3] Z. Yang, L. Li, X. Xu, B. Kailkhura, T. Xie and B. Li, ‘On the certified robustness for ensemble models and beyond’, in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022. [Online]. Available: <https://openreview.net/forum?id=tUa4REjGjTf> (cit. on p. 1).
- [4] S. Rezaei, Z. Shafiq and X. Liu, ‘Accuracy-privacy trade-off in deep ensemble: A membership inference perspective’, in *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, IEEE, 2023, pp. 364–381. DOI: 10.1109/SP46215.2023.10179463. [Online]. Available: <https://doi.org/10.1109/SP46215.2023.10179463> (cit. on p. 1).
- [5] L. Bourtole, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie and N. Papernot, ‘Machine unlearning’, in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, IEEE, 2021, pp. 141–159. DOI: 10.1109/SP40001.2021.00019. [Online]. Available: <https://doi.org/10.1109/SP40001.2021.00019> (cit. on p. 2).
- [6] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. G. Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith and L. Schmidt, ‘Model soups: Averaging weights of multiple fine-tuned models improves ac-

Bibliography

- curacy without increasing inference time’, in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 23 965–23 998. [Online]. Available: <https://proceedings.mlr.press/v162/wortsman22a.html> (cit. on pp. 2, 5, 24, 54).
- [7] S. K. Ainsworth, J. Hayase and S. S. Srinivasa, ‘Git Re-Basin: Merging models modulo permutation symmetries’, *CoRR*, vol. abs/2209.04836, 2022. DOI: 10.48550/arXiv.2209.04836. arXiv: 2209.04836. [Online]. Available: <https://doi.org/10.48550/arXiv.2209.04836> (cit. on pp. 2, 5–7, 16–18, 21, 22, 30, 31, 51).
- [8] S. P. Singh and M. Jaggi, ‘Model fusion via optimal transport’, in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, Eds., 2020. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/hash/fb2697869f56484404c8ceee2985b01d-Abstract.html> (cit. on pp. 2, 17).
- [9] W. Li, Y. Peng, M. Zhang, L. Ding, H. Hu and L. Shen, ‘Deep model fusion: A survey’, *CoRR*, vol. abs/2309.15698, 2023. DOI: 10.48550/ARXIV.2309.15698. arXiv: 2309.15698. [Online]. Available: <https://doi.org/10.48550/arXiv.2309.15698> (cit. on p. 4).
- [10] K. Jordan, H. Sedghi, O. Saukh, R. Entezari and B. Neyshabur, ‘REPAIR: renormalizing permuted activations for interpolation repair’, *CoRR*, vol. abs/2211.08403, 2022. DOI: 10.48550/arXiv.2211.08403. arXiv: 2211.08403. [Online]. Available: <https://doi.org/10.48550/arXiv.2211.08403> (cit. on pp. 5, 16–19, 21, 22, 30, 31, 51, 60, 62).
- [11] M. Matena and C. Raffel, ‘Merging models with fisher-weighted averaging’, in *NeurIPS*, 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/70c26937fbf3d4600b69a129031b66ec-Abstract-Conference.html (cit. on p. 5).

Bibliography

- [12] X. Jin, X. Ren, D. Preotiuc-Pietro and P. Cheng, ‘Dataless knowledge fusion by merging weights of language models’, in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, OpenReview.net, 2023. [Online]. Available: <https://openreview.net/pdf?id=FCnohuR6AnM> (cit. on p. 5).
- [13] T. Garipov, P. Izmailov, D. Podoprikin, D. P. Vetrov and A. G. Wilson, ‘Loss surfaces, mode connectivity, and fast ensembling of dnns’, in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, Eds., 2018, pp. 8803–8812. [Online]. Available: <https://proceedings.neurips.cc/paper/2018/hash/be3087e74e9100d4bc4c6268cdbe8456-Abstract.html> (cit. on pp. 5, 13, 14, 18).
- [14] A. Jolicoeur-Martineau, E. Gervais, K. Fatras, Y. Zhang and S. Lacoste-Julien, ‘Population parameter averaging (PAPA)’, *CoRR*, vol. abs/2304.03094, 2023. DOI: 10.48550/ARXIV.2304.03094. arXiv: 2304.03094. [Online]. Available: <https://doi.org/10.48550/arXiv.2304.03094> (cit. on pp. 5, 6).
- [15] B. T. Polyak and A. B. Juditsky, ‘Acceleration of stochastic approximation by averaging’, *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992. DOI: 10.1137/0330046. [Online]. Available: <https://doi.org/10.1137/0330046> (cit. on p. 5).
- [16] P. Izmailov, D. Podoprikin, T. Garipov, D. P. Vetrov and A. G. Wilson, ‘Averaging weights leads to wider optima and better generalization’, in *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, A. Globerson and R. Silva, Eds., AUAI Press, 2018, pp. 876–885. [Online]. Available: <http://auai.org/uai2018/proceedings/papers/313.pdf> (cit. on p. 5).
- [17] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft and K. Q. Weinberger, ‘Snapshot ensembles: Train 1, get M for free’, in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017. [Online]. Available: <https://openreview.net/forum?id=BJYwwY911> (cit. on p. 5).

Bibliography

- [18] J. Cha, S. Chun, K. Lee, H. Cho, S. Park, Y. Lee and S. Park, ‘SWAD: domain generalization by seeking flat minima’, in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang and J. W. Vaughan, Eds., 2021, pp. 22 405–22 418. [Online]. Available: <https://proceedings.neurips.cc/paper/2021/hash/bcb41ccdc4363c6848a1d760f26c28a0-Abstract.html> (cit. on p. 5).
- [19] H. Guo, J. Jin and B. Liu, ‘Stochastic weight averaging revisited’, *Applied Sciences*, vol. 13, no. 5, 2023, ISSN: 2076-3417. DOI: 10.3390/app13052935. [Online]. Available: <https://www.mdpi.com/2076-3417/13/5/2935> (cit. on p. 5).
- [20] G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson and C. D. Sa, ‘SWALP : Stochastic weight averaging in low precision training’, in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 7015–7024. [Online]. Available: <http://proceedings.mlr.press/v97/yang19d.html> (cit. on p. 5).
- [21] M. Zimmer, C. Spiegel and S. Pokutta, ‘Sparse model soups: A recipe for improved pruning via model averaging’, *CoRR*, vol. abs/2306.16788, 2023. DOI: 10.48550/ARXIV.2306.16788. arXiv: 2306.16788. [Online]. Available: <https://doi.org/10.48550/arXiv.2306.16788> (cit. on p. 6).
- [22] A. Ramé, M. Kirchmeyer, T. Rahier, A. Rakotomamonjy, P. Gallinari and M. Cord, ‘Diverse weight averaging for out-of-distribution generalization’, in *NeurIPS*, 2022. [Online]. Available: http://papers.nips.cc/paper%5C_files/paper/2022/hash/46108d807b50ad4144eb353b5d0e8851-Abstract-Conference.html (cit. on p. 6).
- [23] J. Frankle, G. K. Dziugaite, D. M. Roy and M. Carbin, ‘Linear mode connectivity and the lottery ticket hypothesis’, in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119,

Bibliography

- PMLR, 2020, pp. 3259–3269. [Online]. Available: <http://proceedings.mlr.press/v119/frankle20a.html> (cit. on p. 6).
- [24] R. Hecht-Nielsen, ‘On the algebraic structure of feedforward network weight spaces’, *Advanced Neural Computers*, 1990 (cit. on p. 6).
- [25] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd. Cambridge; New York: Cambridge University Press, 2013, ISBN: 9780521839402 (cit. on p. 8).
- [26] K. Simonyan and A. Zisserman, ‘Very deep convolutional networks for large-scale image recognition’, in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: <http://arxiv.org/abs/1409.1556> (cit. on pp. 11, 21).
- [27] K. He, X. Zhang, S. Ren and J. Sun, ‘Deep residual learning for image recognition’, in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, IEEE Computer Society, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. [Online]. Available: <https://doi.org/10.1109/CVPR.2016.90> (cit. on pp. 11, 21).
- [28] N. J. Tatro, P. Chen, P. Das, I. Melnyk, P. Sattigeri and R. Lai, ‘Optimizing mode connectivity via neuron alignment’, in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, Eds., 2020. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/hash/aecad42329922dfc97eee948606e1f8e-Abstract.html> (cit. on pp. 12–14, 16, 22).
- [29] A. E. Orhan and X. Pitkow, ‘Skip connections eliminate singularities’, in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018. [Online]. Available: <https://openreview.net/forum?id=HkwBEMWCZ> (cit. on pp. 13, 22).
- [30] F. Draxler, K. Veschgini, M. Salmhofer and F. A. Hamprecht, ‘Essentially no barriers in neural network energy landscape’, in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholms-*

Bibliography

- mässan, Stockholm, Sweden, July 10-15, 2018*, J. G. Dy and A. Krause, Eds., ser. *Proceedings of Machine Learning Research*, vol. 80, PMLR, 2018, pp. 1308–1317. [Online]. Available: <http://proceedings.mlr.press/v80/draxler18a.html> (cit. on pp. 13, 14, 19).
- [31] R. Entezari, H. Sedghi, O. Saukh and B. Neyshabur, ‘The role of permutation invariance in linear mode connectivity of neural networks’, in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022. [Online]. Available: <https://openreview.net/forum?id=dNigytemkL> (cit. on pp. 14, 15).
- [32] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. New York: Wiley, 1949 (cit. on p. 16).
- [33] Y. Li, J. Yosinski, J. Clune, H. Lipson and J. E. Hopcroft, ‘Convergent learning: Do different neural networks learn the same representations?’, in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: <http://arxiv.org/abs/1511.07543> (cit. on p. 16).
- [34] H. W. Kuhn, ‘The hungarian method for the assignment problem’, *Naval Research Logistics Quarterly*, vol. 2, no. 1–2, pp. 83–97, Mar. 1955. DOI: 10.1002/nav.3800020109 (cit. on p. 17).
- [35] F. A. Guerrero-Peña, H. R. Medeiros, T. Dubail, M. Aminbeidokhti, E. Granger and M. Pedersoli, ‘Re-basin via implicit sinkhorn differentiation’, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, IEEE, 2023, pp. 20 237–20 246. DOI: 10.1109/CVPR52729.2023.01938. [Online]. Available: <https://doi.org/10.1109/CVPR52729.2023.01938> (cit. on pp. 17, 31, 62).
- [36] N. Markuš, *Fusing batch normalization and convolution in runtime*, Accessed: November 7th, 2023, May 2018. [Online]. Available: <https://nenadmarkus.com/p/fusing-batchnorm-and-conv> (cit. on p. 19).

Bibliography

- [37] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas, ‘Communication-efficient learning of deep networks from decentralized data’, in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, A. Singh and X. (Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282. [Online]. Available: <http://proceedings.mlr.press/v54/mcmahan17a.html> (cit. on pp. 20, 46).
- [38] M. Yurochkin, M. Agarwal, S. Ghosh, K. H. Greenewald, T. N. Hoang and Y. Khazaeni, ‘Bayesian nonparametric federated learning of neural networks’, in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 7252–7261. [Online]. Available: <http://proceedings.mlr.press/v97/yurochkin19a.html> (cit. on p. 20).
- [39] H. Wang, M. Yurochkin, Y. Sun, D. S. Papailiopoulos and Y. Khazaeni, ‘Federated learning with matched averaging’, in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020. [Online]. Available: <https://openreview.net/forum?id=BkluqlSFDS> (cit. on p. 21).
- [40] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, ‘Gradient-based learning applied to document recognition’, *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791. [Online]. Available: <https://doi.org/10.1109/5.726791> (cit. on p. 21).
- [41] O. Ronneberger, P. Fischer and T. Brox, ‘U-Net: Convolutional networks for biomedical image segmentation’, in *Medical Image Computing and Computer-Assisted Intervention*, N. Navab, J. Hornegger, W. M. W. III and A. F. Frangi, Eds., ser. Lecture Notes in Computer Science, vol. 9351, Springer, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28. [Online]. Available: https://doi.org/10.1007/978-3-319-24574-4_28 (cit. on p. 21).
- [42] Z. Liu, H. Mao, C. Wu, C. Feichtenhofer, T. Darrell and S. Xie, ‘A convnet for the 2020s’, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, IEEE, 2022, pp. 11 966–11 976. DOI: 10.1109/CVPR52688.2022.01167. [Online].

Bibliography

- Available: <https://doi.org/10.1109/CVPR52688.2022.01167> (cit. on p. 21).
- [43] S. Hu, Q. Li and B. He, ‘Communication-efficient generalized neuron matching for federated learning’, in *Proceedings of the 52nd International Conference on Parallel Processing, ICPP 2023, Salt Lake City, UT, USA, August 7-10, 2023*, ACM, 2023, pp. 254–263. DOI: 10.1145/3605573.3605726. [Online]. Available: <https://doi.org/10.1145/3605573.3605726> (cit. on pp. 21, 22).
- [44] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, ‘Gradient-based learning applied to document recognition’, *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791 (cit. on pp. 30, 31).
- [45] A. Krizhevsky, ‘Learning multiple layers of features from tiny images’, pp. 32–33, 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf> (cit. on pp. 30, 31, 51).
- [46] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, ‘Reading digits in natural images with unsupervised feature learning’, *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf (cit. on pp. 30, 31).
- [47] X. He, Z. Zhou and L. Thiele, ‘Multi-task zipping via layer-wise neuron sharing’, in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, Eds., 2018, pp. 6019–6029. [Online]. Available: <https://proceedings.neurips.cc/paper/2018/hash/ad8e88c0f76fa4fc8e5474384142a00a-Abstract.html> (cit. on p. 78).

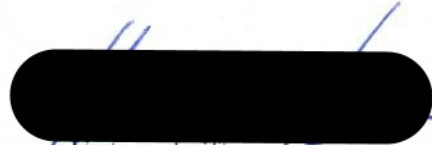
Bibliography

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Master Intelligent Adaptive Systems selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg,
21.12.2023

Ort, Datum


Unterschrift

Erklärung zu Bibliothek

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek eingestellt wird.

Hamburg,
21.12.2023
Ort, Datum


Unterschrift