SAPIENZA
UNIVERSITÀ DI ROMA

Knowledge Transfer in the Wild: Distill and Merge!

Master's Degree in Data Science
Faculty of Information Engineering, Informatics and Statistics

Cem Şirin
Matricola 2050640

External Advisor:
Prof. Hakan Akyüz
Erasmus University Rotterdam

Internal Advisor:
Prof. Ioannis Chatzigiannakis
Sapienza University of Rome

Academic Year 2023/2024

Abstract

In the rapidly evolving field of machine learning, the proliferation of deep neural networks has led to significant advancements across various applications. However, the increasing complexity and size of these models pose challenges in terms of efficient knowledge utilization and transfer. This thesis explores the concept of model merging, a technique aimed at integrating knowledge from multiple trained models into a single, more efficient model. By leveraging methods such as knowledge distillation, model pruning, and weight permutation, model merging seeks to retain the performance and accuracy of the original models while enhancing computational efficiency and reducing resource consumption. This work delves into the methodologies and implications of model merging, with a particular focus on developing architectures that facilitate this process. Key contributions include an exploration of Linear Mode Connectivity (LMC) and the Git Re-Basin method, as well as the introduction of a novel "Distill and Merge!" approach. Through a series of experiments and evaluations, this thesis demonstrates the potential of model merging to create compact, efficient models, thereby broadening the accessibility of advanced machine learning capabilities.

1 Introduction

Over the past several years, the machine learning domain has experienced a remarkable proliferation in the development and implementation of deep neural networks. These models, frequently trained on expansive datasets, have showcased exceptional capabilities across a broad spectrum of applications. Nevertheless, as the complexity and scale of these models continue to escalate, so too does the challenge of effectively leveraging and transferring the knowledge they harbor. It is in this context that the notion of model merging emerges as a crucial consideration (Brunet et al., 2006).

Model merging involves the integration of knowledge from multiple trained models into a single, more efficient model. This process not only aims to retain the performance and accuracy of the original models but also seeks to enhance computational efficiency and reduce resource consumption. The significance of model merging is highlighted by the growing need to deploy machine learning models in collaborative environments, where the ability to combine and transfer knowledge across models is crucial. Additionally, the capability to version control and merge models is vital for the iterative development and refinement of machine learning systems.

The popularity of model merging has been fueled by several recent advancements and research efforts. Techniques such as knowledge distillation, model pruning, and weight permutation have shown promising results in achieving effective model merging. For instance, the Git Re-Basin method Ainsworth et al. (2022) has introduced a novel framework for permuting model weights to achieve Linear Mode Connectivity (LMC), thereby facilitating the merging process. Additionally, the growing body of literature on model merging highlights its potential to address challenges related to model scalability, transferability, and robustness.

As the machine learning community continues to explore and refine these techniques, the relevance of model merging becomes increasingly apparent. By enabling the creation of more compact and efficient models, model merging holds the promise of making advanced machine learning capabilities accessible to a broader range of applications and devices. This thesis aims to delve deeper into the methodologies and implications of model merging, with a focus on developing architectures that are conducive to this process.

This paper is structured as follows: Section 2 outlines the research objectives, focusing on the exploration of model merging via linear mode connectivity. Section 3 reviews related work, including Linear Mode Connectivity (LMC), the Re-basin method, and the Sinkhorn operator. Section 4 discusses the development of a merging-friendly architecture, evaluating various convolution block designs. Section 5 presents an ablation study on different techniques used in the merging process. Section 6 introduces the "Distill and Merge!" approach, combining knowledge distillation and model merging. Section 7 details the experiments and results, comparing the performance of different architectures and techniques. Finally, Section 8 concludes the paper, summarizing the findings and suggesting directions for future research.

2 Research Objectives

The primary objective of this thesis is to explore the concept of model merging via linear mode connectivity in order to transfer knowledge between models. In our initial tests on using popular architectures such as ResNet (He et al., 2016), ConvNeXt (Liu et al., 2022), ResKAGNet (Drokin, 2024) and MobileNetV2

(Sandler et al., 2018), we observed that the merging process was not as effective as expected. This led us roll it back to the basics and build an architecture that is more merging-friendly. We hypothesize that the architecture of the model plays a crucial role in the merging process, and certain designs may be more conducive to effective knowledge transfer and integration.

In this paper, we aim to investigate the impact of different convolution block architectures on the merging process and identify the most merging-friendly designs. By evaluating various convolution block architectures, we seek to understand how architectural choices influence the effectiveness of model merging and knowledge transfer.

3 Related Work

Linear Mode Connectivity (LMC). Linear Mode Connectivity (LMC) is a property of neural networks that allows for the existence of a continuous path between any two minima in the loss landscape. Entezari et al. (2021) conjectured that most SGD solutions can be permuted into the same loss basin, which enables the linear interpolation of weights between two models. Loss barriers are defined as the difference between the loss of the interpolated model and the linear interpolation of the losses of the two models, formally the loss barrier $B(\theta_1, \theta_2)$ is defined as:

$$B\left(\theta_1, \theta_2\right) = \sup_{\alpha} \left[\mathcal{L}\left(\alpha\theta_1 + (1-\alpha)\theta_2\right)\right] - \left[\alpha\mathcal{L}\left(\theta_1\right) + (1-\alpha)\mathcal{L}\left(\theta_2\right)\right] \tag{1}$$

where θ_1 and θ_2 are the weights of two models, α is the interpolation factor, and $\mathcal{L}$ is the loss function. The LMC property has been leveraged in various model merging techniques to achieve efficient knowledge transfer and integration.

Re-basin. The Re-basin method introduced by Ainsworth et al. (2022) is a novel approach to find the optimal permutation of model weights that minimizes the loss barrier between two models. By permuting the weights of two models, the Re-basin method aims to align the loss basins of the models, thereby facilitating the merging process.

Sinkhorn Operator. The Sinkhorn operator is a differentiable approximation of the optimal transport problem, which has been used in various machine learning tasks, including domain adaptation, optimal transport, and model merging. Peña et al. (2023) proposed to use the sinkhorn operator so that the permutation matrices are differentiable and can be optimized using gradient descent. This approach also allows to customize the cost function for the optimal transport problem.

4 Building a Merging Friendly Architecture

Previous research has highlighted the critical role of model architecture in achieving Linear Mode Connectivity (LMC). Ainsworth et al. (2022) demonstrated that wider models are more likely to be permuted into the same loss basin. Unlike traditional ResNet models, modern convolutional architectures such as MobileNet and ConvNeXt employ depthwise separable convolutions and smaller kernels, which increase model width while keeping the parameter count low. Additionally, these contemporary models have shown superior performance compared to older architectures like ResNet and VGG. Therefore, we hypothesize that these newer models may be more conducive to the merging process.

Our primary objective is to design an architecture that is more merging-friendly. By merging-friendly, we mean an architecture that facilitates the combination of multiple trained models into a single model with minimal loss in performance. This is particularly important in scenarios where model ensembles or federated learning are employed, as it allows for more efficient and effective integration of diverse models.

Building on this insight, we evaluated five distinct convolution block architectures to understand their impact on the merging process. The architectures tested include: (1) Basic ResNet block, (2) Bottleneck ResNet block, (3) Inverted Bottleneck block, (4) Depthwise Separable Convolution block (inspired by ConvNeXt), and (5) MobileNetV2-inspired block. These blocks were assessed using the MNIST and CIFAR-10 datasets, which are common benchmarks for evaluating model performance. In figure 1, we provide a visual comparison of the different convolution block architectures.

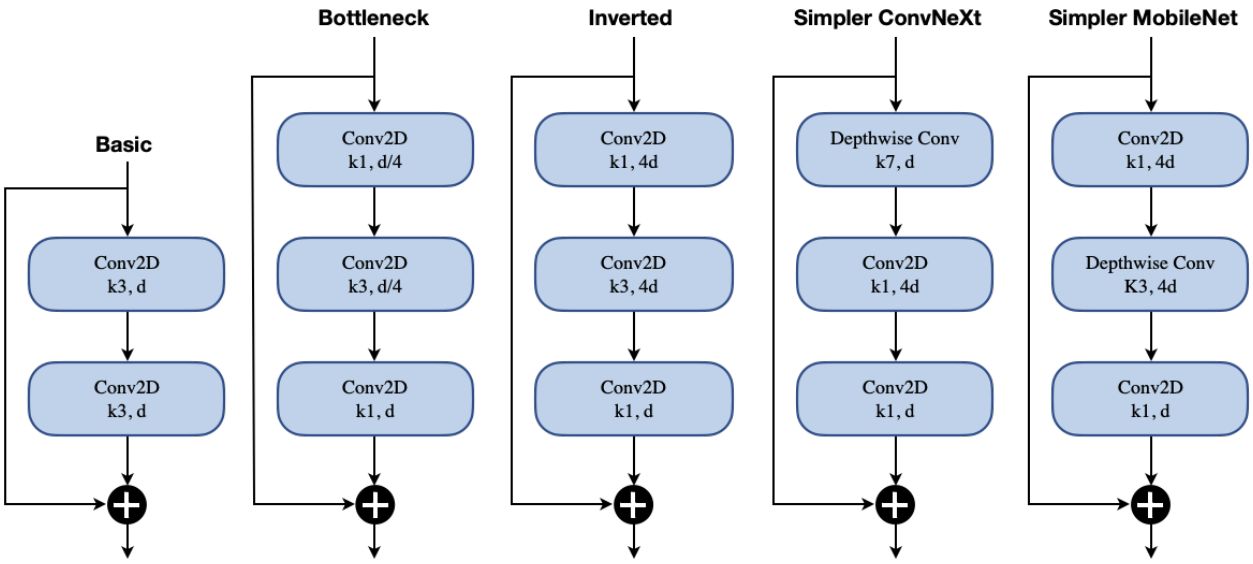

Figure 1: Comparison of different convolution block architectures.

Every convolution layer is followed by a batch normalization layer and a ReLU activation function. Note that this is not the original ConvNeXt and MobileNetV2 architecture, but a simplified version that retains the key characteristics of each block. Different normalization and activation functions will be tested in the following sections.

To fairly evaluate which block architecture is most suitable for merging, we constructed a 2-block model using each architecture and trained them on the MNIST datasets. We constrained the number of parameters to 16K, 64K, and 256K, and designed models with the maximum possible dimensions within these parameter limits. Each model was trained for 100 epochs, and the merging process was tested using the Git Re-Basin method.

5 Ablation Study on Tricks

We then did an ablation study for a couple of tricks that are used in the merging process. We found that the initialization of the weights is crucial for the merging process. We also found that the learning rate schedule is important for the merging process.

Permutation Layer. We have introduced a permutation layer to the residual connections where there is no parametric downsampling, i.e., the identity connection. Figure 2 illustrates the introduced permutation layer. When the connection is through an identity layer, a permutation matrix must be applied to subsequent blocks to match the permutation of the previous layer. The addition of the permutation layer ensures that a permutation matrix is applied to at most two neighboring blocks. As a result, there are also more permutation matrices for the identity connections than for the permutation connections. Refer to Tables 4 and 5 in the appendix for detailed examples of the permutation mappings for a 2-block convolutional model. Table 4 illustrates the mapping with an identity connection in block 0.

Normalization Fusing. Normalization fusing techniques potentially address the issue of variance collapse that can occur during the merging process. This technique involves combining the normalization layers of the models to be merged by calculating the mean and variance of these layers and applying the resulting statistics to the merged model (Ioffe, 2015). By aligning the normalization parameters, normalization fusing may potentially improve the merging process by reducing the discrepancy between the models.

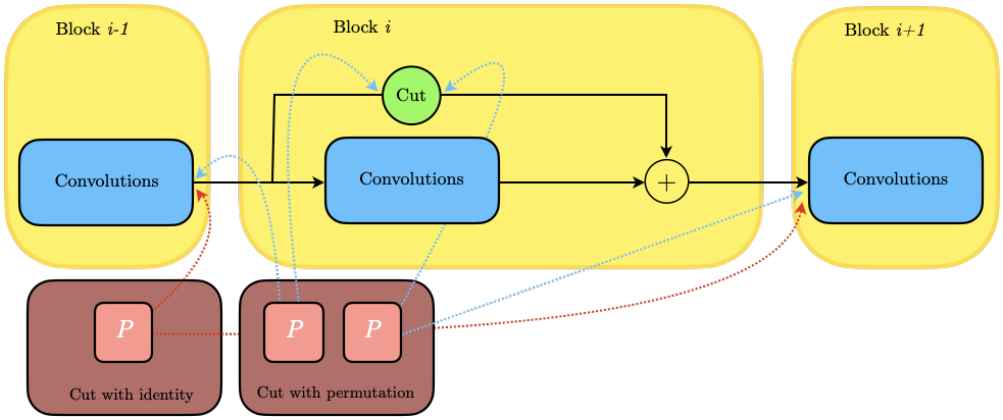

Figure 2: Illustration of the permutation layer introduced to residual connections. Red lines indicate cut connection with identity and blue lines indicate cut connection with permutation.

6 Ditill and Merge!

Distill and Merge! is an innovative concept in the field of machine learning that combines the principles of knowledge distillation and model merging to create a more efficient and effective model. The primary idea behind this approach is to distill the knowledge from multiple trained models into a single, compact model and then merge these distilled models to leverage the strengths of each.

Knowledge distillation, introduced by Hinton (2015), involves transferring the knowledge from a large, complex model (teacher) to a smaller, simpler model (student). This process aims to retain the performance and accuracy of the teacher model while reducing the computational complexity and resource requirements of the student model. By distilling the knowledge from multiple models, we can create a set of student models that encapsulate the collective knowledge of the original models.

Once the knowledge distillation process is complete, the next step is to merge the distilled models. Model merging involves integrating the knowledge from these distilled models into a single, unified model. This process aims to combine the strengths of each distilled model, resulting in a model that is not only efficient but also robust and capable of performing well across a variety of tasks.

The Distill and Merge! approach offers several advantages. First, it enhances computational efficiency by reducing the size and complexity of the models involved. This is particularly important in resource-constrained environments where computational resources are limited. Second, it enables the deployment of machine learning models in collaborative settings, where the ability to combine and transfer knowledge across models is crucial. Finally, it provides a framework for continuous learning and model refinement, allowing for the iterative development of more effective models.

In this paper, we explore the effectiveness of the Distill and Merge! approach by evaluating its performance on various benchmark datasets. We compare the results of the distilled and merged models with those of traditional models to demonstrate the benefits of this technique. Our findings highlight the potential of the Distill and Merge! approach in creating more compact, efficient, and robust models, making advanced machine learning capabilities accessible to a broader range of applications and devices.

7 Experiments and Results

To evaluate the performance of different convolution block architectures in the merging process, we conducted a series of experiments using the MNIST (LeCun et al., 2010) and CIFAR-10 (Krizhevsky, 2009) datasets. Our goal was to assess the accuracy and loss values of models before and after merging, thereby identifying which architectures are most conducive to the merging process. The results of these experiments are summarized in Table 1.

For the MNIST dataset, we constructed models with 2 convolutional blocks, while for the CIFAR-10 dataset, we used models with 3 convolutional blocks. Each model was designed to have the maximum

possible hidden dimensions within the specified parameter constraints. We trained these models for 100 epochs with a batch size of 128. The learning rate was set to 4e-3 and decayed using a cosine annealing schedule. The training was performed on Paperspace Gradient using various GPUs, including NVIDIA RTX 5000, P5000, and A4000, depending on availability.

The merging process involved applying the Sinkhorn operator to the permutation matrices and optimizing these matrices using gradient descent. This approach allowed us to align the loss basins of different models, facilitating an effective merging process.

Our experiments revealed several key insights:

1. **MobileNetV2-inspired Block**: This architecture consistently outperformed other block designs across different parameter sizes and datasets. Its superior performance suggests that the depthwise separable convolutions and efficient design of MobileNetV2 make it highly suitable for the merging process.

2. **Basic ResNet Block and Bottleneck ResNet Block**: Both of these architectures showed competitive performance, with the Basic ResNet block slightly outperforming the Bottleneck ResNet block. These results indicate that traditional ResNet designs are still effective for model merging, although they may not be as efficient as more modern architectures.

3. **ConvNeXt-inspired Block**: This architecture underperformed compared to others. Despite its modern design, it did not facilitate the merging process as effectively as the MobileNetV2-inspired block or the ResNet blocks.

4. **Inverted Bottleneck Block**: This block exhibited the lowest performance among the architectures tested. Its design, which is optimized for other tasks, may not be well-suited for the merging process, highlighting the importance of architectural choices in model merging.

In summary, our experiments demonstrate that the choice of convolution block architecture significantly impacts the effectiveness of the model merging process. The MobileNetV2-inspired block emerged as the most merging-friendly design, offering a promising direction for future research and development in this area.

	MNIST		CIFAR-10	
Block Architecture	Acc. (%)	Loss	Acc. (%)	Loss
Tiny	Capped at 16K		Capped at 64K	
Basic ResNet block	**99.1**	**0.0273**	73.0 (-0.12)	0.7692 (+0.0023)
Bottleneck ResNet block	**99.1**	0.0276	72.8 (+0.17)	0.7787 (+0.0021)
Inverted Bottleneck block	98.7	0.0471	68.6 (+0.20)	0.8896 (-0.0054)
ConvNeXt-inspiredblock	97.1	0.0532	71.1 (-0.29)	0.8123 (+0.0023)
MobileNetV2-inspired block	**99.1**	0.0274	**76.8** (+0.08)	**0.6629** (-0.0059)
Small	Capped at 64K		Capped at 256K	
Basic ResNet block	99.5	0.0162	77.7 (+0.20)	0.6393 (-0.0021)
Bottleneck ResNet block	99.3	0.0189	77.7 (+0.28)	0.6586 (+0.0016)
Inverted Bottleneck block	99.3	0.0230	75.3 (+0.12)	0.7119 (-0.0039)
ConvNeXt-inspired block	98.3	0.0144	73.5 (-0.24)	0.7632 (+0.0014)
MobileNetV2-inspired block	99.4	0.0191	80.4 (-0.11)	0.5963 (0.0000)
Medium	Capped at 256K		Capped at 512K	
Basic ResNet block	99.6	0.0135	79.7 (+0.25)	0.5962 (-0.0045)
Bottleneck ResNet block	99.5	0.0153	78.3 (+0.11)	0.6331 (-0.0019)
Inverted Bottleneck block	99.4	0.0159	77.5 (+0.01)	0.6545 (-0.0019)
ConvNeXt-inspired block	98.8	0.0147	76.2 (-0 .01)	0.6932 (+0.0029)
MobileNetV2-inspired block	99.4	0.0180	82.4 (-0.03)	0.5541 (0.0000)

Table 1: Performance comparison of different convolution block architectures. The accuracy and barrier values are reported for the MNIST and CIFAR-10 datasets. The values in parentheses indicate the change in performance after merging. The changes are color-coded, with purple indicating a decrease and cyan indicating an increase in performance.

7.1 Ablation Study on Tricks

Based on the results of the convolution block experiments, we conducted an ablation study using the MobileNetV2-inspired block to evaluate the impact of various tricks on the merging process. The tricks tested include increasing the number of blocks, using the GELU activation function, applying dropout, adding a permutation layer, and fusing normalization layers. The results of this ablation study are summarized in Table 2.

Trick	Params (K)	Acc. (%)	Loss
Baseline	535	82.4 (-0.03)	0.5541 (0.0000)
+ Increase Number of Blocks to 4	673	86.1 (-0.51)	0.3871 (+0.0059)
+ Increase in width to 1.5x	1432	87.1 (-0.21)	0.3600 (+0.0022)
+ GELU activation	1432	87.6 (+0.12)	0.3200 (-0.0001)
+ Dropout	1432	87.9 (-0.03)	0.2961 (+0.0001)
+ Permutation Layer	1432	90.2 (+0.29)	0.2322 (-0.0021)
+ Normalization Fusing	1432	89.9 (+0.28)	0.2419 (-0.0022)

Table 2: Ablation study results for different tricks used in the merging process.

Our ablation study revealed several key insights. Firstly, increasing the number of blocks in the model resulted in a decrease in merging capability, although it enhanced the model's performance. Secondly, expanding the model's width improved performance but similarly reduced merging capability. The use of the GELU activation function was found to enhance both the model's performance and its merging capability. Conversely, the application of dropout negatively impacted both performance and merging capability. Notably, the introduction of a permutation layer significantly boosted both the model's performance and its merging capability. Lastly, while normalization fusing did not markedly affect the model's performance, it did improve the merging capability.

7.2 Distill and Merge!

In our experiments, we exclusively utilized the MobileNetV2-inspired block, which we identified as the most merging-friendly architecture. Unfortunately, we could not benchmark against other architectures due to their failure in the merging process. Although Ainsworth et al. (2022) demonstrated that ResNet-20 with 32x width exhibits zero loss-barrier, we were unable to replicate these results across different datasets. Additionally, the model size constraints prevented us from conducting extensive experiments with larger models.

In this section, in addition to CIFAR-10, we also evaluated our approach on the Beans (Lab, 2020) and Food101 (Bossard et al., 2014) datasets. These datasets are among the most popular image classification datasets available on Hugging Face Datasets, providing numerous pre-trained models for distillation. We performed distillation from scratch, distillation from a pre-trained model, and merging of the distilled models. The pre-trained models were fine-tuned with weight decay from the pre-trained weights. The teacher models used were ViT fine-tuned on Food-101 (Ashaduzzaman, 2024), ViT fine-tuned on Beans (merve, 2023), and ViT fine-tuned on CIFAR-10 (nateraw, 2022). The results are summarized in Table 3.

	CIFAR-10		Beans		Food101	
Model	Acc. (%)	Loss	Acc. (%)	Loss	Acc. (%)	Loss
Simple Training (A)	90.2	0.2919	94.0	0.3401	84.0	2.0291
Distilled from scratch (B)	92.7	0.2122	96.5	0.3001	87.1	1.843
Merged (A and B)	90.5	0.2322	94.1	0.3440	85.9	1.978
Distilled from pretrained (C)	91.2	0.2298	95.2	0.4056	86.3	1.840
Merged (A and C)	90.8	0.2560	94.3	0.3440	85.9	1.970

Table 3: Performance distill and merge results for different models.

Our results indicate that the merging process is more effective when using distilled models that have been trained with weight decay. However, it is important to note that these distilled models exhibit slightly

lower performance compared to those distilled from scratch. This observation highlights a trade-off between merging capability and individual model performance.

Specifically, models distilled with weight decay tend to merge more seamlessly, likely due to the regularization effect of weight decay, which promotes smoother loss landscapes and better alignment of model parameters. This smoother alignment facilitates the merging process, resulting in a more cohesive and integrated final model. On the other hand, models distilled from scratch, while achieving higher individual performance, may have more complex and less aligned parameter spaces, making the merging process more challenging.

This trade-off is crucial for practitioners to consider when employing the distill and merge approach. In scenarios where the primary goal is to achieve the best possible performance from the merged model, it may be beneficial to prioritize models distilled from scratch. Conversely, in collaborative or resource-constrained environments where efficient merging is paramount, using models distilled with weight decay could be more advantageous.

Overall, our findings underscore the importance of balancing individual model performance with merging capability, and they provide valuable insights for optimizing the distill and merge process in various machine learning applications.

8 Conclusion

In this paper, we have explored the concept of model merging in machine learning and its implications for knowledge transfer and integration. By evaluating different convolution block architectures and their impact on the merging process, we have identified the MobileNetV2-inspired block as the most merging-friendly design. Our experiments on the MNIST and CIFAR-10 datasets have demonstrated the superior performance of this architecture in facilitating the merging process, highlighting the importance of architectural choices in model merging.

We have also conducted an ablation study to evaluate the impact of various tricks on the merging process. Our results indicate that the permutation layer, normalization fusing, and the GELU activation function significantly enhance the merging capability of the model. Conversely, increasing the number of blocks and applying dropout negatively affect the merging process. These findings provide valuable insights into the design and optimization of machine learning models for efficient knowledge transfer and integration.

Furthermore, this paper represents one of the pioneering studies that delve into the impact of architectural decisions on the model merging process. Our findings suggest that the trade-off between model performance and merging capability is a critical consideration in the design of future machine learning models. Models that exhibit high merging capability may be particularly advantageous in collaborative environments, continuous learning scenarios, and federated learning frameworks. By facilitating the seamless integration of knowledge from multiple sources, such models can enhance the efficiency and effectiveness of machine learning systems. We hope that this study will inspire further research in this area, ultimately contributing to the development of more sophisticated and versatile machine learning models that can adapt to a wide range of applications and deployment contexts.

It is important to note that this study did not incorporate augmentation techniques due to limited computational resources. Instead, images were preprocessed before training rather than during training. However, similar to dropout, batch normalization, and other regularization methods, augmentation techniques play a crucial role in enhancing model generalization. As previously discussed, techniques that improve generalization are advantageous for the merging process. Future research should explore the impact of augmentation techniques on model merging to further understand their benefits. Especially, the no pooling architecture may benefit from augmentation techniques as it may be more prone to overfitting positional information.

In the future, we plan to extend our research to explore the merging process in even more detailed fashion by investigating the impact of different activation functions, normalization layers, and optimization algorithms, via examining hidden layer activations and gradients. Additionally, we aim to explore the merging process in more complex architectures, such as transformers and graph neural networks, to understand how these models can be effectively merged. By delving deeper into the merging process and its implications, we hope to contribute to the development of more efficient, robust, and versatile machine learning models that can address a wide range of real-world challenges.

References

Ainsworth, S. K., Hayase, J., and Srinivasa, S. (2022). Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*.

Ashaduzzaman (2024). Vit fine-tuned on food-101. Available at: https://huggingface.co/ashaduzzaman/vit-finetuned-food101.

Bossard, L., Guillaumin, M., and Van Gool, L. (2014). Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*.

Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., and Sabetzadeh, M. (2006). A manifesto for model merging. In *Proceedings of the 2006 international workshop on Global integrated model management*, pages 5–12.

Drokin, I. (2024). Kolmogorov-arnold convolutions: Design principles and empirical studies. *arXiv preprint arXiv:2407.01092*.

Entezari, R., Sedghi, H., Saukh, O., and Neyshabur, B. (2021). The role of permutation invariance in linear mode connectivity of neural networks. *arXiv preprint arXiv:2110.06296*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hinton, G. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Ioffe, S. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Lab, M. A. (2020). Bean disease dataset.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. (2022). A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986.

merve (2023). Vit fine-tuned on beans. Available at: https://huggingface.co/merve/beans-vit-224.

nateraw (2022). Vit fine-tuned on cifar-10. Available at: https://huggingface.co/nateraw/vit-base-patch16-224-cifar10.

Peña, F. A. G., Medeiros, H. R., Dubail, T., Aminbeidokhti, M., Granger, E., and Pedersoli, M. (2023). Re-basin via implicit sinkhorn differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20237–20246.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

A Training Details

All models are trained using the AdamW optimizer with a learning rate of 4e-3 with momenentums 0.9 and 0.999, and weight decay 5e-2. The learning rate is decayed with cosine annealing. The models for MNIST, Beans, CIFAR-10, and Food101 are trained for 30, 30, 50 and 50 epochs, respectively. The training for the permutation matrices is done for 5 epochs with the same optimizer and learning rate. The batch size is set to 128 for all models. The models are trained on Paperspace Gradient with various GPUs, including NVIDIA RTX 5000, P5000, and A4000 based on availability.

B Change in Permutation Mapping After Adding Permutation Layer

Tables 4 and 5 illustrate the permutation mappings for a 2-block convolutional model with an identity connection in block 0 and a permutation layer in block 0, respectively. The addition of the permutation layer results in 6 permutation matrices for identity connections and 7 permutation matrices for permutation connections, highlighting the increased complexity introduced by the permutation layer.

Key	Shape	Perm ID	Shape	Prev Perm ID	Shape
embedder.conv.weight	[24, 1, 7, 7]	0	24		
embedder.norm.weight	[24]	0	24		
embedder.norm.bias	[24]	0	24		
encoder.0.conv.0.conv.weight	[96, 24, 1, 1]	1	96	0	24
encoder.0.conv.0.norm.weight	[96]	1	96		
encoder.0.conv.0.norm.bias	[96]	1	96		
encoder.0.conv.1.conv.weight	[96, 1, 3, 3]	1	96		
encoder.0.conv.1.norm.weight	[96]	1	96		
encoder.0.conv.1.norm.bias	[96]	1	96		
encoder.0.conv.2.conv.weight	[24, 96, 1, 1]	0	24	1	96
encoder.0.conv.2.norm.weight	[24]	0	24		
encoder.0.conv.2.norm.bias	[24]	0	24		
encoder.1.downsample.0.weight	[49, 24, 1, 1]	5	49	0	24
encoder.1.downsample.1.weight	[49]	5	49		
encoder.1.downsample.1.bias	[49]	5	49		
encoder.1.conv.0.conv.weight	[96, 24, 1, 1]	3	96	0	24
encoder.1.conv.0.norm.weight	[96]	3	96		
encoder.1.conv.0.norm.bias	[96]	3	96		
encoder.1.conv.1.conv.weight	[96, 1, 3, 3]	3	96		
encoder.1.conv.1.norm.weight	[96]	3	96		
encoder.1.conv.1.norm.bias	[96]	3	96		
encoder.1.conv.2.conv.weight	[49, 96, 1, 1]	5	49	3	96
encoder.1.conv.2.norm.weight	[49]	5	49		
encoder.1.conv.2.norm.bias	[49]	5	49		
classifier.weight	[10, 49]			5	49
classifier.bias	[10]				

Table 4: Permutation mapping for 2 block convolutional model with identity connection in block 0, in PyTorch format.

Key	Shape	Perm ID	Shape	Prev Perm ID	Shape
embedder.conv.weight	[24, 1, 7, 7]	0	24		
embedder.norm.weight	[24]	0	24		
embedder.norm.bias	[24]	0	24		
encoder.0.downsample.weight	[24, 24]	3	24	0	24
encoder.0.conv.0.conv.weight	[96, 24, 1, 1]	1	96	0	24
encoder.0.conv.0.norm.weight	[96]	1	96		
encoder.0.conv.0.norm.bias	[96]	1	96		
encoder.0.conv.1.conv.weight	[96, 1, 3, 3]	1	96		
encoder.0.conv.1.norm.weight	[96]	1	96		
encoder.0.conv.1.norm.bias	[96]	1	96		
encoder.0.conv.2.conv.weight	[24, 96, 1, 1]	3	24	1	96
encoder.0.conv.2.norm.weight	[24]	3	24		
encoder.0.conv.2.norm.bias	[24]	3	24		
encoder.1.downsample.0.weight	[49, 24, 1, 1]	6	49	3	24
encoder.1.downsample.1.weight	[49]	6	49		
encoder.1.downsample.1.bias	[49]	6	49		
encoder.1.conv.0.conv.weight	[96, 24, 1, 1]	4	96	3	24
encoder.1.conv.0.norm.weight	[96]	4	96		
encoder.1.conv.0.norm.bias	[96]	4	96		
encoder.1.conv.1.conv.weight	[96, 1, 3, 3]	4	96		
encoder.1.conv.1.norm.weight	[96]	4	96		
encoder.1.conv.1.norm.bias	[96]	4	96		
encoder.1.conv.2.conv.weight	[49, 96, 1, 1]	6	49	4	96
encoder.1.conv.2.norm.weight	[49]	6	49		
encoder.1.conv.2.norm.bias	[49]	6	49		
classifier.weight	[10, 49]			6	49
classifier.bias	[10]				

Table 5: Permutation mapping for 2 block convolutional model with permutation layer in block 0, in PyTorch format. Note that encoder.0.downsample.weight is the weight of the permutation layer.