Stitching Disparate Neural Network Layers with Complex Adapters and Spatial Rescaling

Neil Traft[1,2] **Nick Cheney**[1,2]

[1]Neurobotics Lab, University of Vermont
[2]Vermont Complex Systems Institute

Abstract Model stitching is a technique for assembling new neural networks from the parts of existing networks, without having to re-train or fine-tune the existing weights. It has shown promise for new forms of neural architecture search, decentralized training, and transfer learning. However, little investigation has gone into determining exactly what types of blocks can (or cannot) be stitched together, and how. In this short paper, we ask whether it is possible to stitch very low layers to very high layers, across different architectures. We find that it *is* possible to stitch such disparate layers, but requires some important modifications to the original stitching methods.[1]

1 Introduction

Model stitching was originally proposed as an interpretability method, for determining whether the internal representations of two different neural networks were functionally similar (Lenc and Vedaldi, 2015; Bansal et al., 2021). Thus, the original stitching papers only stitch identical layers from identical architectures, and only learn linear transformations.

Since then, many works have proposed to use stitching for the practical purpose of constructing new models, not just introspecting existing models. These works demonstrate the promise of stitching for diverse purposes. Ni et al. (2023) enables more distributed training of large models by training parts of the network separately and stitching them together later. Wang et al. (2023a) and Wang et al. (2023b) use stitching to enable the use of heterogenous models in federated learning. Other works select subsets of a number of models from a model zoo, and stitch them into a new model; this suggests a more efficient, flexible form of transfer learning compared to fine-tuning a single model (Yang et al., 2022; He et al., 2024; Liu et al., 2024; Teerapittayanon et al., 2023).

Some works even describe stitching diverse model architectures—e.g. stitching a CNN to a Transformer—which is far outside the scope of the original stitching methods (Yang et al., 2022; Pan et al., 2023; He et al., 2024; Liu et al., 2024). They describe assembling models from blocks pulled from a diverse model zoo. But they do not describe in detail how effective it is to stitch such diverse block types, nor any limitations on the process.

Thus, stitching has promise for a diverse range of AutoML use cases. It may enable new methods in neural architecture search, if it is possible to recombine architectural elements without having to train their weights from scratch. It may be paired with dynamic routing techniques to assemble large models from the pieces of smaller ones (Komatsuzaki et al., 2023; Muqeeth et al., 2024). In order to judge these promising directions, we probe what is possible with model stitching.

In this short paper we focus on one sub-question: is it possible to stitch at very different depths (e.g. low layers to high layers)? We find that it is, in fact, *not* possible with the original stitching methods. But with some modifications, listed in Section 3, it becomes possible and can achieve surprisingly good performance. This increases the scope of what can be stitched, potentially opening up new use cases.

[1]Our source code is available at `https://github.com/uvm-neurobotics-lab/stitching`.

2 Methods

For simplicity and reproducibility, we focus on stitching two models which were trained on the same dataset. In some cases, these might be two separate parts of the exact same model; in other cases, they might have entirely different architectures. Regardless, all the models were pretrained on ImageNet-1k, and all stitched models will be trained and evaluated on ImageNet-1k.

Given a source model A and a destination model B, we wish to know if we can stitch point A_i to point B_j, where $i \ll j$ and $i, j \in [0, 1]$ are fractions representing how far along in the feedforward computation graph of the model ($i = 0$ would be the input and $j = 1$ would be the output). If such a model can achieve a test accuracy close to the original models, this would be a success, since it has effectively "skipped" many intermediate layers.

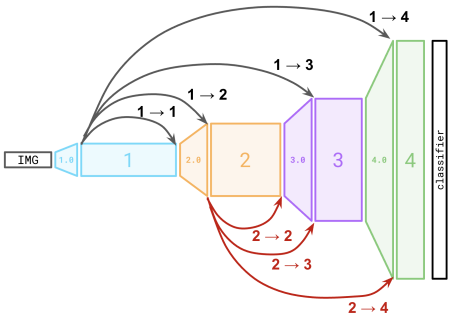


Figure 1: An illustration of the different gaps that may be stitched under our experimental protocol. For each arrow pictured here, we knock out the intermediate layers spanned by the arrow, and replace them with a particular type of adapter (or multiple adapters, in the case of bridging multiple scale-shifts). See Section 2 for a full description.

To simplify the possible permutations of this question, we will divide our candidate models into four stages. Many common vision models are already structured as four stages, or can be divided this way. For instance, ResNet-18 consists of four stages, with two blocks in each stage. ResNet-50 is four stages with {3, 4, 6, 4} blocks in each stage. Typically, each stage operates at a different receptive field scale. At the beginning of each stage, the model might transform the input as $[B, 2C, H/2, W/2]$—thus, **stitching across different stages involves stitching across different spatial scales**. We will simplify by choosing one stitching point in each stage, and testing all pairs where Stage $x \leq$ Stage y. When $x = y$, we use different points within the same stage. See Figure 1 for a diagram of the different gaps that we will stitch across.

We test our ability to stitch these gaps on four different scenarios: parts of the same ResNet-50[2], parts of the same Swin-Tiny[3], parts of the same MobileNetV3[4], and the beginning of ResNet-50 to the end of Swin-Tiny.

The stitching methods originally used in Bansal et al. (2021) were kept as strictly linear transforms for interpretability purposes (i.e., a 1x1 convolution). In our case, we are interested in finding out whatever stitching methods may be successful, even if nonlinear. Thus, **we test a number of different adapters of varying complexity**: Linear, Linear + ReLU, Conv3x3 + ReLU, ResNet BottleneckBlock (He et al., 2016).

In addition, we also test a scenario with no adapter (fine-tuning all weights). This baseline preserves the original downsample blocks from the top model, but deletes all intermediate blocks, and fine-tunes end-to-end. Schemes similar to this have been shown to be an effective form of warm-starting (Xu et al., 2023). However, this is only possible when stitching models with similar architecture, and cannot serve as a general replacement for stitching.

Further, we expect that we must rescale the feature maps to match the size expected at their destination. If we do not do this, we often run out of memory (see Section 3). This fact is not mentioned in "model zoo" approaches such as Yang et al. (2022); Pan et al. (2023); Liu et al. (2024). To our knowledge, StitchNet (Teerapittayanon et al., 2023) is unique in mentioning this issue, but only briefly. **We compare a few different methods for dealing with the spatial scaling problem**:

[2]Pretrained weights resnet50.a1_in1k from the Huggingface timm library (Wightman, 2019).
[3]Pretrained weights swin_t from the Torchvision library (Torchvision maintainers and contributors, 2016).
[4]Pretrained weights mobilenetv3_large_100.ra_in1k from the Huggingface timm library (Wightman, 2019).

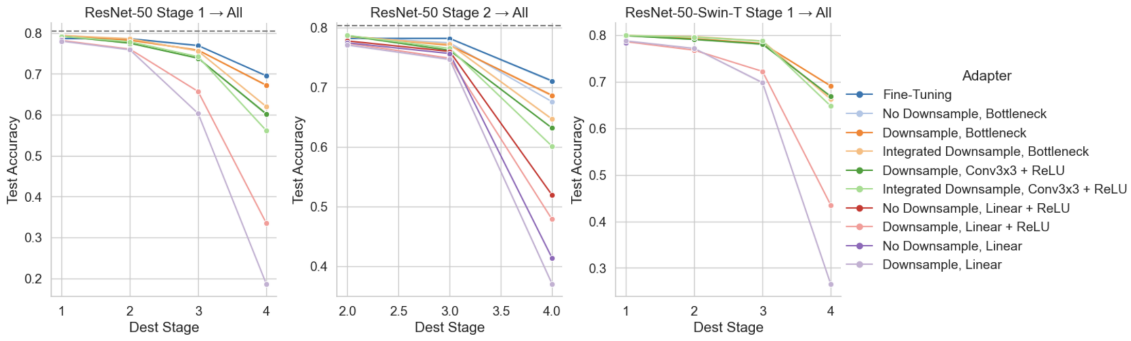


Figure 2: The test accuracy of various stitching methods in different scenarios. Downsampling and BottleneckBlocks can achieve performance close to fine-tuning, while being applicable in a range of scenarios far broader than fine-tuning. **Left**: Stitching from Stage 1 in ResNet-50. **Middle**: Stitching from Stage 2 in ResNet-50. **Right**: Stitching from ResNet-50 Stage 1 to other stages in Swin-T. Results for all other architectures and stages can be found in Appendix A.

- No Downsample — The method of prior literature.
- Downsample — Downsample just before each adapter using bilinear image interpolation.
- Integrated Downsample — Using a 3x3 convolution of stride 2.

We place BatchNorm (Ioffe and Szegedy, 2015) layers before and after each adapter to aid in optimization, as in Bansal et al. (2021). **All scenarios are optimized on ImageNet-1k for just 10 epochs**. For full details of optimization, see Appendix B.

3 Results

Here we will highlight four major findings:

1. It is possible to stitch across large gaps and across very different architectures, by interpolating the feature maps accordingly.
2. More complex adapters are needed to achieve good accuracy, rather than the traditional linear adapters.
3. Using a "bottleneck" style adapter is important for computational efficiency.
4. It is sufficient to use a simple bilinear image interpolation to match feature map sizes.

Stitching Across Large Gaps. We can stitch across very large gaps (e.g. Stage 1 → Stage 4), approaching the performance of full-parameter fine-tuning. Without spatial scaling, stitching across such gaps is often not possible (note that the *No Downsample* variants are missing from Figure 2 Left and Right, due to out-of-memory errors when attempting to push such large feature maps into later stages of the model).

Table 1: By stitching various parts of a ResNet-50, we can produce models which are similar to the smaller ResNet variants, with only 10 epochs of training.

Model	Accuracy	FLOPs
ResNet-18	**71.5**	1.81e09
ResNet-50 Stage 1 → 4	67.4	**1.67e09**
ResNet-34	76.4	3.66e09
ResNet-50 Stage 1 → 2	**78.3**	**3.58e09**
ResNet-50	80.4	4.09e09

As a baseline, we can also compare to smaller ResNet variants when trained from scratch. Table 1 shows that by stitching various gaps, we can turn a ResNet-50 into the rough equivalent of a ResNet-18 or ResNet-34, with just 10 epochs of adapter training. Although the objective of this method is not to be competitive on model compression, we find this to be an intriguing example of the efficacy of stitching.

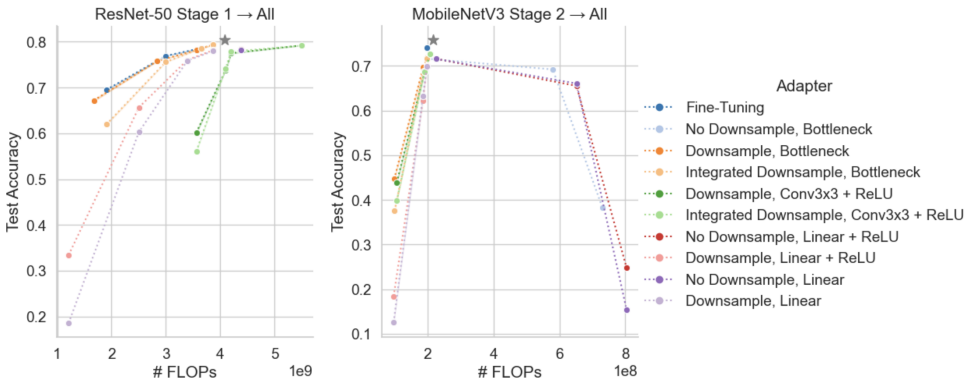


Figure 3: Accuracy vs. FLOP count, in two different scenarios. Up and to the left means better accuracy with less compute. Grey stars represent the original model. **Left**: ResNet-50 Stage 1. See how *Downsample, Bottleneck* matches the pareto curve of *Fine-Tuning*. Notice how the orange lines fully dominate the green lines (*Bottleneck* blocks are more efficient and performant than a full *Conv3x3*). **Right**: MobileNetV3 Stage 2. See how all *No Downsample* variants actually *grow* the computation relative to the original model, even though there are fewer layers.

Our method functions in cases where fine-tuning cannot, and can even stitch together such different architectures as a CNN and a Swin Transformer into a very capable model. Figure 2 shows that our best method (*Downsample, Bottleneck*) can stitch the ResNet Stage 1 layers into Swin Stage 3 (a ~33% reduction in FLOPs relative to Swin-T) with only a 3.3% drop in accuracy relative to the original Swin model. By stitching Stage 1 $\rightarrow$ Stage 4, we can even produce a ~65% reduction in FLOPs, with only ~12% lower accuracy than Swin-T.

More Complex Adapters. Using more sophisticated adapters greatly improves performance over linear adapters. In Figure 2, see that *Conv3x3* and *Bottleneck* adapters substantially outperform the original *Linear* and *Linear + ReLU* adapters used in all prior stitching work.

Bottleneck-shaped Adapters for Efficiency. This increased complexity must be balanced with efficiency. The ResNet BottleneckBlock (He et al., 2016) projects channels down to a smaller dimensionality before performing 3x3 convolution. Using a bottleneck rather than a full-channel convolution (*Conv3x3*) matches performance with a much smaller FLOP count. A similar effect was used in He et al. (2024), but it was based on LoRA (Hu et al., 2022) and limited to a linear transformation. Downsampling is also important for keeping the model tractable. See Figure 3.

Integrated Scaling is Not Needed. We also considered the possibility that interpolating from one receptive field scale to another would need to be learned as part of the adapter itself. To test this, we used adapters which incorporate a strided convolution, such that scaling and transformation happen simultaneously ("*Integrated Downsample*"). We find that this integrated downsampling is not necessary; a separate downsample step works just as well—in fact, better. In Figure 2, we see that *Downsample, Conv3x3* beats *Integrated Downsample, Conv3x3*, and similarly for the *Bottleneck* adapters. This replicates across all 12 scenarios in Appendix A.

4 Conclusion

We have determined two key modifications which greatly increase the range of layers that can be stitched together: (1) using more complex transformations, and (2) interpolating feature maps to their expected scale. We achieve accuracies on ImageNet-1k comparable to those of models trained from scratch, with only 10 epochs of adapter training. This reveals stitching as a promising technique for assembling novel architectures.

Acknowledgements. This material is based upon work supported by the National Science Foundation under Grants No. 2239691 and 2218063. Computations were performed on the Vermont Advanced Computing Core supported in part by NSF Award No. OAC-1827314.

References

Bansal, Y., Nakkiran, P., and Barak, B. (2021). Revisiting Model Stitching to Compare Neural Representations. *Advances in Neural Information Processing Systems*, 34:225–236.

He, H., Pan, Z., Liu, J., Cai, J., and Zhuang, B. (2024). Efficient Stitchable Task Adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28555–28565.

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.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. (2022). LoRA: Low-rank Adaptation of Large Language Models. *International Cnference on Learning Representations*, 1(2):3.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456. pmlr.

Komatsuzaki, A., Puigcerver, J., Lee-Thorp, J., Ruiz, C. R., Mustafa, B., Ainslie, J., Tay, Y., Dehghani, M., and Houlsby, N. (2023). Sparse Upcycling: Training Mixture-of-Experts from Dense Checkpoints. In *The Eleventh International Conference on Learning Representations*.

Lenc, K. and Vedaldi, A. (2015). Understanding Image Representations by Measuring Their Equivariance and Equivalence. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 991–999.

Liu, Z., Lu, Q., Zhao, Y., Zhao, Y., and Luo, J. (2024). Differentiable Neural Network for Assembling Blocks. In *ECAI 2024*, pages 2250–2257. IOS Press.

Loshchilov, I. and Hutter, F. (2019). Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

maintainers, T. and contributors (2016). TorchVision: PyTorch's Computer Vision library. `https://github.com/pytorch/vision`.

Muqeeth, M., Liu, H., Liu, Y., and Raffel, C. (2024). Learning to Route Among Specialized Experts for Zero-Shot Generalization. In *International Conference on Machine Learning*, pages 36829–36846. PMLR.

Ni, Z., Wang, Y., Yu, J., Jiang, H., Cao, Y., and Huang, G. (2023). Deep Incubation: Training Large Models by Divide-and-Conquering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17335–17345.

Pan, Z., Cai, J., and Zhuang, B. (2023). Stitchable Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16102–16112.

Teerapittayanon, S., Comiter, M., McDanel, B., and Kung, H. (2023). StitchNet: Composing Neural Networks from Pre-Trained Fragments. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pages 61–68. IEEE.

Wang, J., Cui, S., and Ma, F. (2023a). FedLEGO: Enabling Heterogenous Model Cooperation via Brick Reassembly in Federated Learning. In *International Workshop on Federated Learning for Distributed Data Mining*.

Wang, J., Yang, X., Cui, S., Che, L., Lyu, L., Xu, D. D., and Ma, F. (2023b). Towards Personalized Federated Learning via Heterogeneous Model Reassembly. *Advances in Neural Information Processing Systems*, 36:29515–29531.

Wightman, R. (2019). Pytorch image models. `https://github.com/rwightman/pytorch-image-models`.

Xu, Z., Chen, Y., Vishniakov, K., Yin, Y., Shen, Z., Darrell, T., Liu, L., and Liu, Z. (2023). Initializing Models with Larger Ones. *arXiv preprint arXiv:2311.18823*.

Yang, X., Zhou, D., Liu, S., Ye, J., and Wang, X. (2022). Deep Model Reassembly. *Advances in Neural Information Processing Systems*, 35:25739–25753.

A Full Results

See Figures 4 and 5 for results across all 12 scenarios.

B Details of Optimization

As mentioned in the main text, Section 2, all scenarios are optimized on ImageNet-1k image classification, using standard cross-entropy loss, for 10 epochs. We use AdamW (Loshchilov and Hutter, 2019) with an effective batch size of 512, learning rate 0.002, weight decay 0.05, and a learning rate schedule of cosine annealing to 0.0. We found that the fine-tuning baseline is more sensitive to learning rate, so in that case we ran with three separate learning rates (0.01, 0.002, 0.0002), and take the best result for each data point. Experiments justifying the learning rates are described in the next section (Appendix C). For each setting, our results are aggregated over two replicates with different random seeds.

C Learning Rate Study

We find that training stitching adapters is relatively stable with AdamW. Our chosen learning rate 0.002 and weight decay 0.05 seem to be optimal across the various adapters and architectures that we checked. See Figure 6.

However, for fine-tuning all weights, we find larger performance differences across different learning rates. We also find the best learning rate is not consistent across all scenarios. Due to these issues, we run separate trials for all three learning rates and report the best performer in each unique scenario (Appendix B).

Figure 4: Accuracy vs. stitching gap size, across three stages on all four architectures.

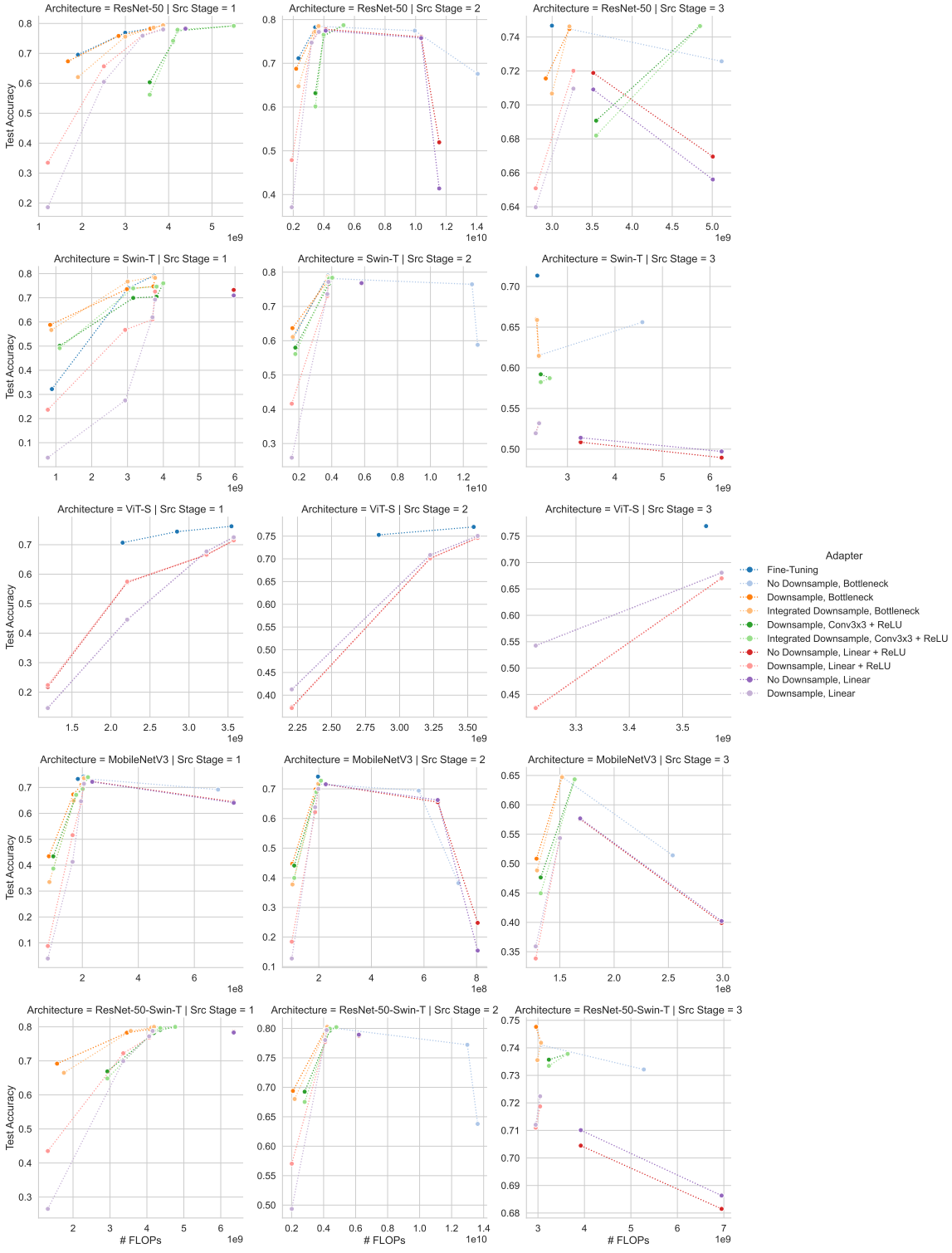
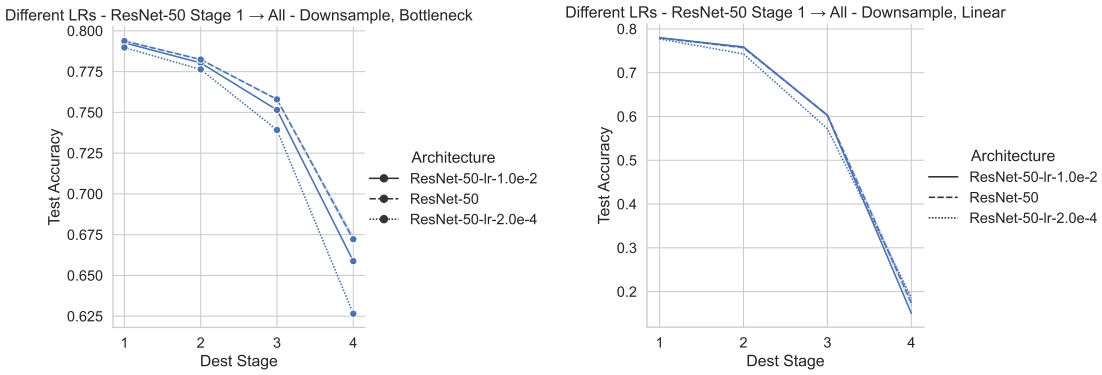


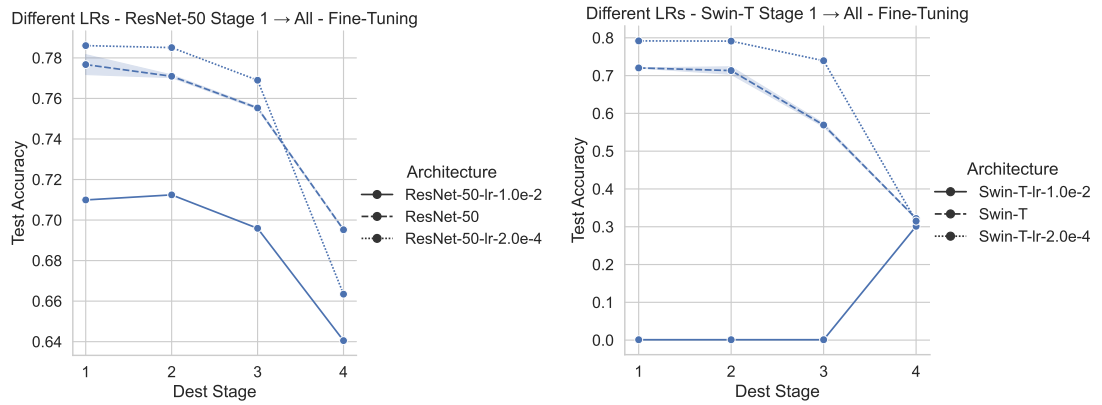
Figure 5: Accuracy vs. FLOP count of the resulting model, across three stages on all four architectures.



(a) BottleneckBlock adapter

(b) Linear adapter

Figure 6: Optimizing adapters using various learning rates, to stitch ResNet-50 Stage 1 to others. We find that the default learning rate (the dashed line) is generally optimal, with minor differences to other learning rates.



(a) ResNet-50

(b) Swin-Tiny

Figure 7: Fine-tuning all weights using various learning rates, on two different architectures. In contrast to the adapters, we find a larger performance difference across different learning rates, especially for Swin-T. We find that fine-tuning tends to prefer the smaller learning rate (the dotted line), but the best learning rate is not consistent across all scenarios. Due to these issues, we run separate trials for all three learning rates and report the best performer in each unique scenario.