Mimetic Initialization for Deep Neural Networks

Asher James Trockman

CMU-CS-25-114

May 2025

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
J. Zico Kolter, Chair
Albert Gu
Aditi Raghunathan
Sébastien Bubeck (OpenAI)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science.

This research was sponsored by the Air Force Research Laboratory under award number FA87501720152; Robert Bosch GMBH under award number 0087016732PCRPO0087023984; Robert Bosch LLC under award number OSP00009188; and the Defense Advanced Research Projects Agency under award numbers HR00112020006 and HR00112320029. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Deep Learning, Computer Vision, Convolutional Neural Networks, Vision Transformers, Self-Attention, State Space Models, Multilayer Perceptrons, Initialization

For Mom and Dad —
and for Aunt Marsha

Abstract

While neural network weights are typically initialized randomly from univariate distributions, pre-trained weights often have visually-discernible multivariate structure. We propose a technique called "mimetic initialization" that aims to replicate such structures when initializing convolutional networks (CNNs), Transformers, and State Space Models (SSMs). For CNNs, we handcraft a class of multivariate Gaussian distributions to initialize filters for depthwise convolutional layers; for Transformers, we initialize the query and key weights for self-attention layers such that their product approximates the identity; and for SSMs, we initialize layers to approximate simple linear attention. Mimetic initialization substantially reduces training time and increases final accuracy on various common small-scale benchmarks. Our technique enables us to almost close the gap between untrained and pre-trained Vision Transformers on small datasets like CIFAR-10, achieving up to a 6% gain in accuracy through initialization alone. For convolutional networks like ConvMixer and ConvNeXt, we observe improvements in accuracy and reductions in training time, even when convolutional filters are frozen (untrained) after initialization. For SSMs, mimetic initialization substantially improves generalization abilities on synthetic language tasks like copying and associative recall. Overall, our findings suggest that some of the benefits of pre-training may be explained by it serving as a good initialization, whose structure is simple enough to (at least partially) capture by hand in closed form.

Acknowledgments

Foreword to the acknowledgment: It feels funny to write an acknowledgment for such a modest contribution to the science of deep learning. After all, I just did the research that seemed the most fun at the time. So my acknowledgment is not just for the work presented in this document, but rather for the past nearly six years of graduate school. Even when life got harder, as it did for a substantial part of those six years, my time at Carnegie Mellon remained a consistent source of enjoyment.

That said, I would first like to thank my mom, Lisa Trockman, for her endless support, advice, and reassurance. The past six years were hard for both of us, but we have gotten through them together. And I thank my dad, James (Jim) Trockman, for inspiring in me the thing that makes someone want to spend six years doing research in graduate school— appreciation of beauty, curiosity, language, meditative work... And I thank my aunt, Marsha Trockman, for encouraging me to explore and experience the world, even when it's not comfortable. I also want to thank Aunt Jane, Uncle Barry, and my sisters: Jami, Joni, and Jordyn.

I also want to thank the many good friends I have made here and in related endeavors—you all made this a great time. My friends from my cohort: Lucio Dery, Justin Whitehouse, Praneeth Kacham, Siddharth Prasad. And from around Carnegie Mellon: David Widder, Rijnard van Tonder, Ian Waudby-Smith, Likhitha Chintareddy, Sang Choe, Courtney Miller, Tanya Marwah. My friends and collaborators from my lab: Gaurav Manek, Jeremy Cohen, Avi Schwarzschild, Yash Savani, Joshua Williams, Zhili Feng, Alex Robey, Marc Finzi. And from internships: Nicholas Roberts, Jessica Yin, Chris Choi, Bingbin Liu, Changming Xu. And from conferences: Felix Petersen, Kelsey Doerksen. Thank you all, I've enjoyed all the good times.

I was also fortunate to be advised by Zico Kolter for the past six years; it has been a pleasure to be immersed in some of the coolest ideas in machine learning. I appreciated your support and understanding in the more difficult years, and the freedom to choose my own direction in the easier years. I've enjoyed working on our many projects, even the ones that didn't work. I was also fortunate to be mentored by Bogdan Vasilescu and Christian Kästner in Carnegie Mellon's REUSE program in the summer of 2017, two years before starting graduate school; that summer has been paying dividends for eight years! Thanks, Bogdan, for our many chats over the years.

Between the aforementioned summer and now, it feels like I've been here forever. I'm sad to go, but I didn't want to take up the Computer Science Department on its maximum offer of ten years. I'm looking forward to what's next.

Contents

List of Figures

List of Tables

Chapter 1

Initializing deep neural networks

1.1 Introduction

Initialization was at one point crucial for training very deep neural networks. Early works in this area attempted to address the problems of vanishing and exploding gradients, initializing the weights so that the variance of activations and gradients was bounded from layer to layer (He et al., 2015; Glorot and Bengio, 2010). Historically, the weights of deep neural networks are initialized at random from *univariate* distributions with variance dependent on the hidden dimension of the layer at hand. In particular, most work initializes weights $W \in \mathbb{R}^{m \times n}$ in order to preserve the variance of inputs $x \in \mathbb{R}^n$. For illustrative purposes, we could say that these weights W for a particular layer are drawn from a multivariate Gaussian with fixed variance $\alpha \propto \mathsf{Hidden\ Dimension}$, i.e., with a diagonal covariance matrix:

$$\mathsf{vec}(W) \sim \mathcal{N}(0, \alpha I). \tag{1.1}$$

This formulation captures the most common strategies for initialization of deep networks, which are typically only concerned with the particular setting of α given the shape of W; in particular, it is common to set $\alpha = n^{-\frac{1}{2}}$, which preserves the variance of the inputs. This comes from the following fact, where we assume $x, W \sim \mathcal{N}(0, 1)$ for the purpose of illustration:

$$\mathsf{Var}(n^{-\frac{1}{2}} W x) = \mathsf{Var}(x). \tag{1.2}$$

Kaiming initialization (He et al., 2015), the predominant strategy in deep learning libraries such as PyTorch, further accounts for the effect of the ReLU activation function which, intuitively speaking, halves the variance of its inputs, setting $\alpha = 2^{\frac{1}{2}} n^{-\frac{1}{2}}$.

While initialization using such heuristics is still important, techniques such as adaptive optimizers and normalization layers have made it less critical to the trainability of deep neural networks Kingma and Ba (2017); Ioffe and Szegedy (2015). These methods ensure that gradients and activations do not explode or vanish even for sub-optimal choices of α.

As another perspective on initialization, the most common paradigm in deep learning currently involves *pretraining* neural networks on large, general datasets and then *finetuning* them on downstream tasks (Kolesnikov et al., 2020; Bommasani et al., 2021)—that is, pretraining

serves as a kind of smart initialization. This leads to much better performance than directly training on the downstream tasks: for language, learning tasks such as copying and recall within state space models seems to be near-impossible without pretraining (Jelassi et al., 2024), and in vision we see especially large improvements for more flexible architectures such as Vision Transformers (Dosovitskiy et al., 2020), which are notoriously difficult to train from scratch on small-scale datasets. This seems to be the case even when the downstream task is substantially different from the pretraining dataset (Kolesnikov et al., 2020), especially for larger architectures (Raghu et al., 2019). In fact, recent work has suggested that one should "never train from scratch" (Amos et al., 2023).

The prevailing wisdom is that pretraining is about storing transferrable knowledge or learning representations that can be leveraged on downstream tasks (Bommasani et al., 2021). However, some work has hypothesized that some of the benefits of pretraining may be due to the fact that it serves as a *good initialization*, in that pretrained networks are simply easier to optimize than their untrained counterparts (Zhang et al., 2022; Neyshabur et al., 2020).

We hypothesized that it would be possible to extract this *good initialization* component from pretraining in *closed-form* through studies of pretrained neural networks, rather than through mathematical first principles as in work on signal propagation (He et al., 2023; Xiao et al., 2018; Martens et al., 2021). From such studies, perhaps we can obtain new initialization schemes that significantly improve training even with the full suite of tricks in modern deep learning such as BatchNorm and Adam and variants.

As a first step towards investigating our hypothesis, we propose a simple ansatz of the form of the *good initialization* component of pretraining which is more flexible than the formulation in Eq. 1.1:

$$\mathsf{vec}(W) \sim \mathcal{N}(0, \Sigma), \tag{1.3}$$

for a specifically-structured or even unconstrained covariance matrix Σ, whose structure we set through inspiration from studies of pretrained networks. In some cases, we study the joint distribution of related or adjacent layers W_a, W_b instead:

$$[\mathsf{vec}(W_a); \mathsf{vec}(W_b)] \sim \mathcal{N}(0, \Sigma_{ab}). \tag{1.4}$$

In the work that follows, we show that by using this more general initialization scheme that allows for high-dimensional statistical dependencies between weights, we can considerably improve neural network performance in a variety of settings. We conclude by presenting some observations of the covariance structure of the *entire weight space* of small convolutional networks.

1.1.1 Thesis overview

We begin by proposing a novel and extremely simple convolutional neural network architecture called ConvMixer, whose simplicity will lead us to propose the first mimetic initialization for convolutional layers. Unlike previous convolutional networks, ConvMixer separates convolutional layers into *depthwise* and *pointwise* convolutions, where *depthwise* convolutions mix over the spatial dimension with a single independent filter per channel and *pointwise* convolutions mix over channel dimensions (equivalently to linear layers). Also unlike previous convolutional

networks, ConvMixer uses large 9×9 filters rather than stacks of smaller 3×3 filters. In our initial study on ConvMixer, we notice that these filters contain interesting and perhaps interpretable structures after training.

Inspired by the structures we observed in ConvMixer filters, we propose the first "mimetic initialization" for convolutional filters, which decreases training time, increases data efficiency, and results in higher final accuracy compared to traditional univariate initializations. We then extend our method to self-attention layers in vision transformers, which successfully addresses the gap in data efficiency between vision transformers and convolutional networks. Then, leveraging the connection between self-attention layers and state space layers, we demonstrate that mimetic initialization allows state space models (namely Mamba version 1 and 2) to more easily learn to do tasks such as copying and associative recall. Finally, we attempt to extend mimetic initialization beyond spatial mixing layers (convolution layers, attention layers, state space layers) to channel mixing layers, and we propose a very simple mimetic initialization scheme for multilayer perceptrons that can be combined with the previous spatial mixing layer mimetic initialization techniques.

Chapter 2

ConvMixer, the simple CNN

2.1 Patches are all you need? 🤷

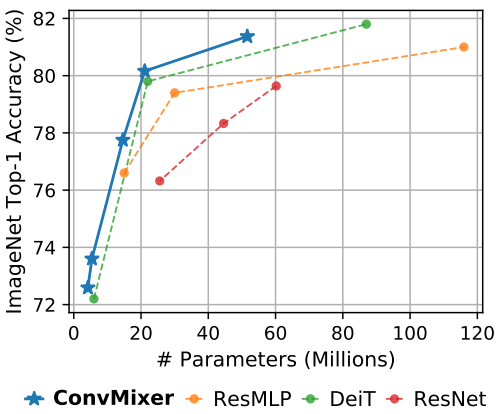


Figure 2.1: Accuracy *vs.* parameters, trained and evaluated on ImageNet-1k.

For many years, convolutional neural networks have been the dominant architecture for deep learning systems applied to computer vision tasks. But recently, architectures based upon *Transformer* models, *e.g.*, the so-called Vision Transformer architecture (Dosovitskiy et al., 2020), have demonstrated compelling performance in many of these tasks, often outperforming classical convolutional architectures, especially for large data sets. An understandable assumption, then, is that it is only a matter of time before Transformers become the dominant architecture for vision domains, just as they have for language processing. In order to apply Transformers to images, however, the representation had to be changed: because the computational cost of the self-attention layers used in Transformers would scale quadratically with the number of pixels per image if applied naively at the per-pixel level, the compromise was to first split the image into multiple "patches", linearly embed them, and then apply the transformer directly to this collection of patches.

In this work, we explore the question of whether, fundamentally, the strong performance of vision transformers may result more from this patch-based representation than from the Transformer architecture itself. We develop a very simple convolutional architecture which we dub the "ConvMixer" due to its similarity to the recently-proposed MLP-Mixer (Tolstikhin et al., 2021). This architecture is similar to the Vision Transformer (and MLP-Mixer) in many respects: it directly operates on patches, it maintains an equal-resolution-and-size representation throughout all layers, it does no downsampling of the representation at successive layers, and it separates "channel-wise mixing" from the "spatial mixing" of information. But unlike the Vision Transformer and MLP-Mixer, our architecture does all these operations via only standard convolutions.

The chief result we show in this paper is that this ConvMixer architecture, despite its extreme simplicity (it can be implemented in ≈ 6 lines of dense PyTorch code), outperforms both "standard" computer vision models such as ResNets of similar parameter counts *and* some corresponding Vision Transformer and MLP-Mixer variants, even with a slate of additions intended to make those architectures more performant on smaller data sets. Importantly, this is despite the fact that *we did not design our experiments to maximize accuracy nor speed, in contrast to the models we compared against. Our results suggest that,* at least to some extent, the *patch representation itself* may be a critical component to the "superior" performance of newer architectures like Vision Transformers. While these results are naturally just a snapshot, and more experiments are required to exactly disentangle the effect of patch embeddings from other factors, we believe that this provides a strong "convolutional-but-patch-based" baseline to compare against for more advanced architectures in the future.

2.1.1 Introducing ConvMixer: a simple convolutional network

Our model, dubbed *ConvMixer*, consists of a patch embedding layer followed by repeated applications of a simple fully-convolutional block. We maintain the spatial structure of the patch embeddings, as illustrated in Fig. 2.2. Patch embeddings with patch size p and embedding dimension h can be implemented as convolution with c_{in} input channels, h output channels, kernel size p, and stride p:

$$z_0 = \mathsf{BN}\left(\sigma\{\texttt{Conv}_{c_{\text{in}}\to h}(X, \texttt{stride=}p, \texttt{kernel\_size=}p)\}\right) \tag{2.1}$$

The ConvMixer block itself consists of depthwise convolution (i.e., grouped convolution with groups equal to the number of channels, h) followed by pointwise (i.e., kernel size 1×1) convolution. As we will explain in Sec. 2.1.2, ConvMixers work best with unusually large kernel sizes for the depthwise convolution. Each of the convolutions is followed by an activation and post-activation BatchNorm:

$$z'_l = \mathsf{BN}\left(\sigma\{\texttt{ConvDepthwise}(z_{l-1})\}\right) + z_{l-1} \tag{2.2}$$

$$z_{l+1} = \mathsf{BN}\left(\sigma\{\texttt{ConvPointwise}(z'_l)\}\right) \tag{2.3}$$

After many applications of this block, we perform global pooling to get a feature vector of size h, which we pass to a softmax classifier. See Fig. 2.3 for an implementation of ConvMixer in PyTorch.

Design parameters. An instantiation of ConvMixer depends on four parameters: (1) the "width" or hidden dimension h (i.e., the dimension of the patch embeddings), (2) the depth d, or the number of repetitions of the ConvMixer layer, (3) the patch size p which controls the internal resolution of the model, (4) the kernel size k of the depthwise convolutional layer. We name ConvMixers after their hidden dimension and depth, like ConvMixer-h/d. We refer to the original input size n divided by the patch size p as the *internal resolution*; note, however, that ConvMixers support variable-sized inputs.

Motivation. Our architecture is based on the idea of *mixing*, as in Tolstikhin et al. (2021). In particular, we chose depthwise convolution to mix *spatial locations* and pointwise convolution to mix *channel locations*. A key idea from previous work is that MLPs and self-attention can

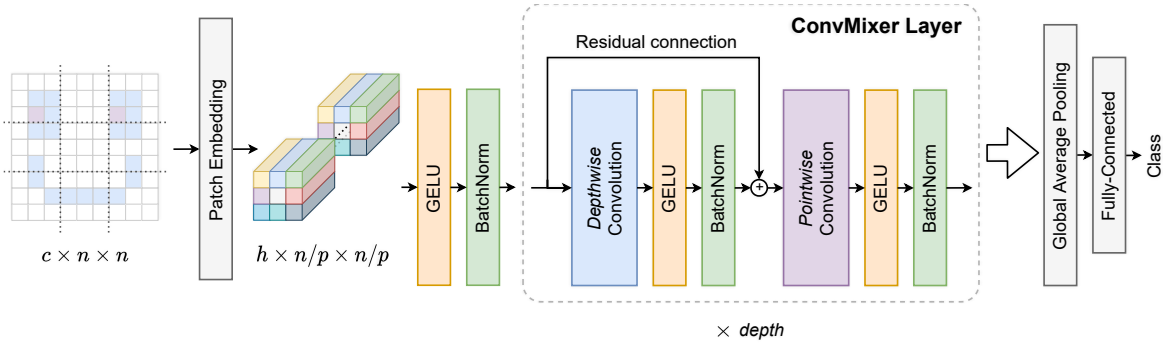


Figure 2.2: ConvMixer uses "tensor layout" patch embeddings to preserve locality, and then applies d copies of a simple fully-convolutional block consisting of *large-kernel* depthwise convolution followed by pointwise convolution, before finishing with global pooling and a simple linear classifier.

```
def ConvMixer(h, depth, kernel_size=9, patch_size=7, n_classes=1000):
    Seq, ActBn = nn.Sequential, lambda x: Seq(x, nn.GELU(), nn.BatchNorm2d(h))
    Residual = type('Residual', (Seq,), {'forward': lambda self, x: self[0](x) + x})
    return Seq(ActBn(nn.Conv2d(3, h, patch_size, stride=patch_size)),
        *[Seq(Residual(ActBn(nn.Conv2d(h, h, kernel_size, groups=h, padding="same"))),
          ActBn(nn.Conv2d(h, h, 1))) for i in range(depth)],
         nn.AdaptiveAvgPool2d((1,1)), nn.Flatten(), nn.Linear(h, n_classes))
```

Figure 2.3: Implementation of ConvMixer in PyTorch; see Appendix A.1 for more implementations.

mix distant spatial locations, i.e., they can have an arbitrarily large receptive field. Consequently, we used convolutions with an unusually large kernel size to mix distant spatial locations.

While self-attention and MLPs are theoretically more flexible, allowing for large receptive fields and content-aware behavior, the inductive bias of convolution is well-suited to vision tasks and leads to high data efficiency. By using such a standard operation, we also get a glimpse into the effect of the patch representation itself in contrast to the conventional pyramid-shaped, progressively-downsampling design of convolutional networks.

2.1.2 ImageNet experiments on ConvMixer

Training setup. We primarily evaluate ConvMixers on ImageNet-1k classification without any pretraining or additional data. We added ConvMixer to the `timm` framework (Wightman, 2019) and trained it with nearly-standard settings: we used RandAugment (Cubuk et al., 2020), mixup (Zhang et al., 2017), CutMix (Yun et al., 2019), random erasing (Zhong et al., 2020), and gradient norm clipping in addition to default `timm` augmentation. We used the AdamW (Loshchilov and Hutter, 2018) optimizer and a simple triangular learning rate schedule. Due to limited compute, *we did absolutely no hyperparameter tuning* on ImageNet and trained for fewer epochs than competitors. Consequently, our models could be over- or under-regularized, and the accuracies we report likely underestimate the capabilities of our model.

Results. A ConvMixer-1536/20 with 52M parameters can achieve 81.4% top-1 accuracy on ImageNet, and a ConvMixer-768/32 with 21M parameters 80.2% (see Table 2.1). Wider

Current "Most Interesting" **ConvMixer** Configurations *vs.* Other Simple Models							
Network	**Patch Size**	**Kernel Size**	**# Params** ($\times 10^6$)	**T.hput (img/sec)**	**Act. Fn.**	**# Epochs**	**ImNet top-1 (%)**
ConvMixer-1536/20	7	9	51.6	134	G	150	81.37
ConvMixer-768/32	7	7	21.1	206	R	300	80.16
ResNet-152	–	3	60.2	828	R	150	79.64
DeiT-B	16	–	86	792	G	300	81.8
ResMLP-B24/8	8	–	129	181	G	400	81.0

Table 2.1: Models trained and evaluated on 224×224 ImageNet-1k only. See more in Appendix 2.1.4.

ConvMixers seem to converge in fewer epochs, but are memory- and compute-hungry. They also work best with large kernel sizes: ConvMixer-1536/20 lost $\approx 1\%$ accuracy when reducing the kernel size from $k = 9$ to $k = 3$ (we discuss kernel sizes more in Section 2.1.4 & 2.1.5). ConvMixers with smaller patches are substantially better in our experiments, similarly to Sandler et al. (2019); we believe larger patches require deeper ConvMixers. With everything held equal except increasing the patch size from 7 to 14, ConvMixer-1536/20 achieves 78.9% top-1 accuracy but is around $4\times$ faster. We trained one model with ReLU to demonstrate that GELU (Hendrycks and Gimpel, 2016), which is popular in recent isotropic models, isn't necessary.

Comparisons. Our model and ImageNet1k-only training setup closely resemble that of recent patch-based models like DeiT (Touvron et al., 2021b). Due to ConvMixer's simplicity, we focus on comparing to only the most basic isotropic patch-based architectures adapted to the ImageNet-1k setting, namely DeiT and ResMLP. Attempting a fair comparison with a standard baseline, we trained ResNets using exactly the same parameters as ConvMixers; while this choice of parameters is suboptimal (Wightman et al., 2021), it is likely also suboptimal for ConvMixers, since we did *no hyperparameter tuning*.

Looking at Table 2.1 and Fig. 2.1, ConvMixers achieve competitive accuracies for a given parameter budget: ConvMixer-1536/20 outperforms both ResNet-152 and ResMLP-B24 despite having substantially fewer parameters and is competitive with DeiT-B. ConvMixer-768/32 uses just a third of the parameters of ResNet-152, but is similarly accurate. Note that unlike ConvMixer, the DeiT and ResMLP results involved hyperparameter tuning, and when substantial resources are dedicated to tuning ResNets, including training for twice as many epochs, they only outperform an equivalently-sized ConvMixer by $\approx 0.2\%$ (Wightman et al., 2021). However, ConvMixers are substantially slower at inference than the competitors, likely due to their smaller patch size; hyperparameter tuning and optimizations could narrow this gap. For more discussion and comparisons, see Table 2.2 and Section 2.1.4.

Hyperparameters. For experiments presented in the main text, we used only one set of "common sense" parameters for the regularization methods. Recently, we adapted parameters

from the A1 procedure in Wightman et al. (2021), published after our work, which were better than our initial guess, *e.g.*, giving $+0.8\%$ for ConvMixer-1536/20, or 82.2% top-1 accuracy (see Section 2.1.4).

CIFAR-10 Experiments. We also performed smaller-scale experiments on CIFAR-10, where ConvMixers achieve over 96% accuracy with as few as 0.7M parameters, demonstrating the data efficiency of the convolutional inductive bias. Details of these experiments are presented in Section 2.1.5.

2.1.3 Visual recognition architecture background

Isotropic architectures. Vision transformers have inspired a new paradigm of "isotropic" architectures, i.e., those with equal size and shape throughout the network, which use patch embeddings for the first layer. These models look similar to repeated transformer-encoder blocks (Vaswani, 2017) with different operations replacing the self-attention and MLP operations. For example, MLP-Mixer (Tolstikhin et al., 2021) replaces them both with MLPs applied across different dimensions (i.e., spatial and channel location mixing); ResMLP (Touvron et al., 2021a) is a data-efficient variation on this theme. CycleMLP (Chen et al., 2021), gMLP (Liu et al., 2021a), and vision permutator (Hou et al., 2021), replace one or both blocks with various novel operations. These are all quite performant, which is typically attributed to the novel choice of operations. In contrast, Melas-Kyriazi (2021) proposed an MLP-based isotropic vision model, and also hypothesized patch embeddings could be behind its performance. ResMLP tried replacing its linear interaction layer with (small-kernel) convolution and achieved good performance, but kept its MLP-based cross-channel layer and did not explore convolutions further. As our investigation of ConvMixers suggests, these works may conflate the effect of the new operations (like self-attention and MLPs) with the effect of the use of patch embeddings and the resulting isotropic architecture.

A study predating vision transformers investigate isotropic (or "isometric") MobileNets (Sandler et al., 2019), and even implements patch embeddings under another name. Their architecture simply repeats an isotropic MobileNetv3 block. They identify a tradeoff between patch size and accuracy that matches our experience, and train similarly performant models (see Section 2.1.4, Table 2.2). However, their block is substantially more complex than ours; simplicity and motivation sets our work apart.

Patches aren't all you need. Several papers have increased vision transformer performance by replacing standard patch embeddings with a different stem: Xiao et al. (2021) and Yuan et al. (2021a) use a standard convolutional stem, while Yuan et al. (2021b) repeatedly combines nearby patch embeddings. However, this conflates the effect of using patch embeddings with the effect of adding convolution or similar inductive biases *e.g.*, locality. We attempt to focus on the use of patches.

CNNs meet ViTs. Many efforts have been made to incorporate features of convolutional networks into vision transformers and vice versa. Self-attention can emulate convolution (Cordonnier et al., 2019) and can be initialized or regularized to be like it (d'Ascoli et al., 2021); other works simply add convolution operations to transformers (Dai et al., 2021; Guo et al., 2021), or include downsampling to be more like traditional pyramid-shaped convolutional networks (Wang et al., 2021). Conversely, self-attention or attention-like operations can supplement

or replace convolution in ResNet-style models (Bello et al., 2019; Ramachandran et al., 2019; Bello, 2021). While all of these attempts have been successful in one way or another, they are orthogonal to this work, which aims to emphasize the effect of the architecture common to most ViTs by showcasing it with a less-expressive operation.

2.1.4 Comparison to related architectures

Experiment overview. We did not design our experiments to maximize accuracy: We chose "common sense" parameters for `timm` and its augmentation settings, found that it worked well for a ConvMixer-1024/12, and stuck with them for the proceeding experiments. We admit this is not an optimal strategy, however, we were aware from our early experiments on CIFAR-10 that results seemed robust to various small changes. We did not have access to sufficient compute to attempt to tune hyperparameters for each model: *e.g.*, larger ConvMixers could probably benefit from more regularization than we chose, and smaller ones from less regularization. Keeping the parameters the same across ConvMixer instances seemed more reasonable than guessing for each.

However, to some extent, we changed the number of epochs per model: for earlier experiments, we merely wanted a "proof of concept", and used only 90–100 epochs. Once we saw potential, we increased this to 150 epochs and trained some larger models, namely ConvMixer-1024/20 with $p = 14$ patches and ConvMixer-1536/20 with $p = 7$ patches. Then, believing that we should explore deeper-but-less-wide ConvMixers, and knowing from CIFAR-10 that the deeper models converged more slowly, we trained ConvMixer-768/32s with $p = 14$ and $p = 7$ for 300 epochs. Of course, training time was a consideration: ConvMixer-1536/20 took about 9 days to train (on $10\times$ RTX8000s) 150 epochs, and ConvMixer-768/32 is over twice as fast, making 300 epochs more feasible.

If anything, we believe that in the worst case, the lack of parameter tuning in our experiments resulted in underestimating the accuracies of ConvMixers. Further, due to our limited compute and the fact that large models (particularly ConvMixers) are expensive to train on large data sets, we generally trained our models for fewer epochs than competition like DeiT and ResMLP (see Table 2.2).

In this revision, we have added some additional results (denoted with a ★ in Table 2.2) using hyperparameters loosely based on the precisely-crafted "A1 training procedure" from Wightman et al. (2021). In particular, we adjusted parameters for RandAug, Mixup, CutMix, Random Erasing, and weight decay to match those in the procedure. Importantly, we still only trained for 150 epochs, rather than the 600 epochs used in Wightman et al. (2021), and we did not use binary cross-entropy loss nor repeated augmentation. While we do not think optimal hyperparameters for ResNet would also be optimal for ConvMixer, these settings are significantly better than the ones we initially chose. This further highlights the capabilities of ConvMixers, and we are optimistic that further tuning could lead to still-better performance. Throughout the paper, we still refer to ConvMixers trained using our initial "one shot" selection of hyperparameters.

A note on throughput. We measured throughput using batches of 64 images in half precision on a single RTX8000 GPU, averaged over 20 such batches. In particular, we measured CUDA execution time rather than "wall-clock" time. We noticed discrepancies in the relative throughputs of models, *e.g.*, Touvron et al. (2021b) reports that ResNet-152 is $2\times$ faster than DeiT-B,

Comparison with other simple models trained on **ImageNet-1k only** with input size 224.

Network	Patch Size	Kernel Size	# Params ($\times 10^6$)	Throughput (img/sec)	Act. Fn.	# Epochs	ImNet top-1 (%)
ConvMixer-1536/20★	7	9	51.6	134	G	150	82.20
ConvMixer-1536/20 ●	7	9	51.6	134	G	150	81.37
ConvMixer-1536/20★	7	3	49.4	246	G	150	81.60
ConvMixer-1536/20	7	3	49.4	246	G	150	80.43
ConvMixer-1536/20	14	9	52.3	538	G	150	78.92
ConvMixer-1536/24★	14	9	62.3	447	G	150	80.21
ConvMixer-768/32 ●	7	7	21.1	206	R	300	80.16
ConvMixer-1024/16	7	9	19.4	244	G	100	79.45
ConvMixer-1024/12	7	8	14.6	358	G	90	77.75
ConvMixer-512/16	7	8	5.4	599	G	90	73.76
ConvMixer-512/12 ●	7	8	4.2	798	G	90	72.59
ConvMixer-768/32	14	3	20.2	1235	R	300	74.93
ConvMixer-1024/20 ●	14	9	24.4	750	G	150	76.94
ResNet-152 ●	–	3	60.2	828	R	150	79.64
ResNet-101 ●	–	3	44.6	1187	R	150	78.33
ResNet-50	–	3	25.6	1739	R	150	76.32
DeiT-B†	7	–	86.7	83	G	–	–
DeiT-S†	7	–	22.1	174	G	–	–
DeiT-Ti†	7	–	5.7	336	G	–	–
DeiT-B ●	16	–	86	792	G	300	81.8
DeiT-S ●	16	–	22	1610	G	300	79.8
DeiT-Ti ●	16	–	5.7	2603	G	300	72.2
ResMLP-S12/8 ●	8	–	22.1	872	G	400	79.1
ResMLP-B24/8 ●	8	–	129	181	G	400	81.0
ResMLP-B24	16	–	116	1597	G	400	81.0
Swin-S ●	4	–	50	576	G	300	83.0
Swin-T ●	4	–	29	878	G	300	81.3
ViT-B/16 ●	16	–	86	789	G	300	77.9
Mixer-B/16 ●	16	–	59	1025	G	300	76.44
Isotropic MobileNetv3 ●	8	3	20	–	R	–	80.6
Isotropic MobileNetv3 ●	16	3	20	–	R	–	77.6

Table 2.2: Throughputs measured on an RTX8000 GPU using batch size 64 and fp16. ConvMixers and ResNets trained ourselves. Other statistics: DeiT (Touvron et al., 2021b), ResMLP (Touvron et al., 2021a), Swin (Liu et al., 2021c), ViT (Dosovitskiy et al., 2020), MLP-Mixer (Tolstikhin et al., 2021), Isotropic MobileNets (Sandler et al., 2019). We think models with matching colored dots (●) are informative to compare with each other. †Throughput tested, but not trained. Activations: **R**eLU, **G**ELU.
★Using new regularization hyperparameters based on Wightman et al. (2021)'s A1 procedure.

but our measurements show that the two models have nearly the same throughput. We therefore speculate that our throughputs may underestimate the performance of ResNets and ConvMixers relative to the transformers. The difference may be due to using RTX8000 rather than V100 GPUs, or other low-level differences. Our throughputs were similar for batch sizes 32 and 128.

ResNets. As a simple baseline to which to compare ConvMixers, we trained three standard ResNets using exactly the same training setup and parameters as ConvMixer-1536/20. Despite having fewer parameters and being architecturally much simpler, ConvMixers substantially outperform these ResNets in terms of accuracy. A possible confounding factor is that ConvMixers use GELU, which may boost performance, while ResNets use ReLU. In an attempt to rule out this confound, we used ReLU in a later ConvMixer-768/32 experiment and found that it still achieved competitive accuracy. We also note that the choice of ReLU *vs.* GELU was not important on CIFAR-10 experiments (see Table 2.3). However, ConvMixers do have substantially less throughput.

DeiTs. We believe that DeiT is the most reasonable comparison in terms of vision transformers: It only adds additional regularization, as opposed to architectural additions in the case of CaiT (Touvron et al., 2021c), and is then essentially a "vanilla" ViT modulo the distillation token (we don't consider distilled architectures). In terms of a fixed parameter budget, ConvMixers generally outperform DeiTs. For example, ConvMixer-1536/20 is only 0.43% less accurate than DeiT-B despite having over 30M fewer parameters; ConvMixer-768/32 is 0.36% more accurate than DeiT-S despite having 0.9M fewer parameters; and ConvMixer-512/16 is 0.39% more accurate than DeiT-Ti for nearly the same number of parameters. Admittedly, none of the ConvMixers are very competitive in terms of throughput, with the closest being the ConvMixer-512/16 which is $4\times$ slower than DeiT-Ti.

A confounding factor is the difference in patch size between DeiT and ConvMixer; DeiT uses $p = 16$ while ConvMixer uses $p = 7$. This means DeiT is substantially faster. However, ConvMixers using larger patches are not as competitive. While we were not able to train DeiTs with larger patch sizes, it is possible that they would outperform ConvMixers on the parameter count *vs.* accuracy curve; however, we tested their throughput for $p = 7$, and they are even slower than ConvMixers. Given the difference between convolution and self-attention, we are not sure it is salient to control for patch size differences.

DeiTs were subject to more hyperparameter tuning than ConvMixers, as well as longer training times. They also used stochastic depth while we did not, which can in some cases contribute percent differences in model accuracy (Touvron et al., 2021a). It is therefore possible that further hyperparameter tuning and more epochs for ConvMixers could close the gap between the two architectures for large patches, *e.g.*, $p = 16$.

ResMLPs. Similarly to DeiT for ViT, we believe that ResMLP is the most relevant MLP-Mixer variant to compare against. Unlike DeiT, we can compare against instances of ResMLP with similar patch size: ResMLP-B24/8 has $p = 8$ patches, and underperforms ConvMixer-1536/20 by 0.37%, despite having over twice the number of parameters; it also has similarly low throughput. ConvMixer-768/32 also outperforms ResMLP-S12/8 for millions fewer parameters, but $3\times$ less throughput.

ResMLP did not significantly improve in terms of accuracy for halving the patch size from 16 to 8, which shows that smaller patches do not always lead to better accuracy for a fixed architecture and regularization strategy (*e.g.*, training a $p = 8$ DeiT may be challenging).

Swin Transformers. While we intend to focus on the most basic isotropic, patch-based architectures for fair comparisons with ConvMixer, it is also interesting to compare to a more complicated model that is closer to state-of-the-art. For a similar parameter budget, ConvMixer is around 1.2-1.6% less accurate than the Swin Transformer, while also being 4-6$\times$ slower. However, considering we did not attempt to tune or optimize our model in any way, we find it surprising that an exceedingly simple patch-based model that uses only plain convolution does not lag too far behind Swin Transformer.

Isotropic MobileNets. These models are closest in design to ours, despite using a repeating block that is substantially more complex than the ConvMixer one. Despite this, for a similar number of parameters, we can get similar performance. Notably, isotropic MobileNets seem to suffer less from larger patch sizes than ConvMixers, which makes us optimistic that sufficient parameter tuning could lead to more performant large-patch ConvMixers.

Other models. We included ViT and MLP-Mixer instances in our table, though they are not competitive with ConvMixer, DeiT, or ResMLP, even though MLP-Mixer has comparable regularization to ConvMixer. That is, ConvMixer seems to outperform MLP-Mixer and ViT, while being closer to complexity to them in terms of design and training regime than the other competitors, DeiT and ResMLP.

Kernel size. While we found some evidence that larger kernels are better on CIFAR-10, we wanted to see if this finding transferred to ImageNet. Consequently, we trained our best-performing model, ConvMixer-1536/20, with kernel size $k = 3$ rather than $k = 9$. This resulted in a decrease of 0.94% top-1 accuracy, which we believe is quite significant relative to the mere 2.2M additional parameters. However, $k = 3$ is substantially faster than $k = 9$ for spatial-domain convolution; we speculate that low-level optimizations could close the performance gap to some extent, *e.g.*, by using implicit instead of explicit padding. Since large-kernel convolutions throughout a model are unconventional, there has likely been low demand for such optimizations.

2.1.5 Small-scale experiments on ConvMixer

Residual connections. We experimented with leaving out one, the other, or both residual connections before settling on the current configuration, and consequently chose to leave out the second residual connection. Our baseline model without the connection achieves 95.88% accuracy, while including the connection reduces it to 94.78%. Surprisingly, we see only a 0.31% decrease in accuracy for *removing all residual connections*. We acknowledge that these findings for residual connections may not generalize to deeper ConvMixers trained on larger data sets.

Normalization. Our model is conceptually similar to the vision transformer and MLP-Mixer, both of which use LayerNorm instead of BatchNorm. We attempted to use LayerNorm instead, and saw a decrease in performance of around 1% as well as slower convergence (see Table 2.3). However, this was for a relatively shallow model, and we cannot guarantee that LayerNorm would not hinder ImageNet-scale models to an even larger degree. We note that the authors of ResMLP also saw a relatively small increase in accuracy for replacing LayerNorm with BatchNorm, but for a larger-scale experiment (Touvron et al., 2021a). We conclude that BatchNorm is no more crucial to our architecture than other regularizations or parameter settings (*e.g.*, kernel size).

Having settled on an architecture, we proceeded to adjust its parameters h, d, p, k as well

Ablation of ConvMixer-256/8 on CIFAR-10	
Ablation	**CIFAR-10 Acc. (%)**
Baseline	95.88
– Residual in Eq. 2.2	95.57
+ Residual in Eq. 2.3	94.78
BatchNorm $\rightarrow$ LayerNorm	94.44
GELU $\rightarrow$ ReLU	95.51
– Mixup and CutMix	95.92
– Random Erasing	95.24
– RandAug	92.86
– Random Scaling	86.24
– Gradient Norm Clipping	86.33

Table 2.3: Small ablation study of training a ConvMixer-256/8 on CIFAR-10.

as weight decay on CIFAR-10 experiments. (Initially, we took the unconventional approach of excluding weight decay since we were already using strong regularization in the form of RandAug and mixup.) We acknowledge that tuning our architecture on CIFAR-10 does not necessarily generalize to performance on larger data sets, and that this is a limitation of our study.

Results. ConvMixers are quite performant on CIFAR-10, easily achieving $> 91\%$ accuracy for as little as $100,000$ parameters, or $> 96\%$ accuracy for only $887,000$ parameters (see Table 2.4). With additional refinements *e.g.*, a more expressive classifier or bottlenecks, we think that ConvMixer could be even more competitive. For all experiments, we trained for 200 epochs on CIFAR-10 with RandAug, mixup, cutmix, random erasing, gradient norm clipping, and the standard augmentations in `timm`. We remove some of these augmentations in Table 2.3, finding that RandAug and random scaling ("default" in `timm`) are very important, each accounting for over 3% of the accuracy.

Scaling ConvMixer. We adjusted the hidden dimension h and the depth d, finding that deeper networks take longer to converge while wider networks converge faster. That said, increasing the width or the depth is an effective way to increase accuracy; a doubling of depth incurs less compute than a doubling of width. The number of parameters in a ConvMixer is given exactly by:

$$\#\mathsf{params} = h[d(k^2 + h + 6) + c_{\mathsf{in}}p^2 + n_{\mathsf{classes}} + 3] + n_{\mathsf{classes}}, \tag{2.4}$$

including affine scaling parameters in BatchNorm layers, convolutional kernels, and the classifier.

Kernel size. We initially hypothesized that large kernels would be important for ConvMixers, as they would allow the mixing of distant spatial information similarly to unconstrained MLPs

or self-attention layers. We tried to investigate the effect of kernel size on CIFAR-10: we fixed the model to be a ConvMixer-256/8, and increased the kernel size by 2s from 3 to 15.

Using a kernel size of 3, the ConvMixer only achieves 93.61% accuracy. Simply increasing it to 5 gives an additional 1.50% accuracy, and further to 7 an additional 0.61%. The gains afterwards are relatively marginal, with kernel size 15 giving an additional 0.28% accuracy. It could be that with more training iterations or more regularization, the effect of larger kernels would be more pronounced. Nonetheless, we concluded that ConvMixers benefit from larger-than-usual kernels, and thus used kernel sizes 7 or 9 in most of our later experiments.

It is conventional wisdom that large-kernel convolutions can be "decomposed" into stacked small-kernel convolutions with activations between them, and it is therefore standard practice to use $k = 3$ convolutions, stacking more of them to increase the receptive field size with additional benefits from nonlinearities. This raises a question: is the benefit of larger kernels in ConvMixer actually better than simply increasing the depth with small kernels? First, we note that deeper networks are generally harder to train, so by increasing the kernel size independently of the depth, we may recover some of the benefits of depth without making it harder for signals to "propagate back" through the network. To test this, we trained a ConvMixer-256/10 with $k = 3$ (698K parameters) in the same setting as a ConvMixer-256/8 with $k = 9$ (707K parameters), i.e., we increased depth in a small-kernel model to roughly match the parameters of a large-kernel model. The ConvMixer-256/10 achieved 94.29% accuracy (1.5% less), which provides more evidence for the importance of larger kernels in ConvMixers. Next, instead of fixing the parameter budget, we tripled the depth (using the intuition that 3 stacked $k = 3$ convolutions have the receptive field of a $k = 9$ convolution), giving a ConvMixer-256/24 with 1670K parameters, and got 95.16% accuracy, i.e., still less.

Patch size. CIFAR-10 inputs are so small that we initially only used $p = 1$, i.e., the patch embedding layer does little more than compute h linear combinations of the input image. Using $p = 2$, we see a reduction in accuracy of about 0.80%; this is a worthy tradeoff in terms of training and inference time. Further increasing the patch size leads to rapid decreases in accuracy, with only 92.61% for $p = 4$.

Since the "internal resolution" is decreased by a factor of p when increasing the patch size, we assumed that larger kernels would be less important for larger p. We investigated this by again increasing the kernel size from 3 to 11 for ConvMixer-256/8 with $p = 2$: however, this time, the improvement going from 3 to 5 is only 1.13%, and larger kernels than 5 provide only marginal benefit.

Weight decay. We did many of our initial experiments with minimal weight decay. However, this was not optimal: by tuning weight decay, we can get an additional 0.15% of accuracy for no cost. Consequently, we used weight decay (without tuning) for our larger-scale experiments on ImageNet.

2.1.6 Inspecting the structure of ConvMixer's weights

Tiny **ConvMixers** trained on CIFAR-10.						
Width h	**Depth** d	**Patch Size** p	**Kernel Size** k	**# Params** ($\times 10^3$)	**Weight Decay**	**CIFAR-10 Acc. (%)**
128	4	1	8	103	0	91.26
128	8	1	8	205	0	93.83
128	12	1	8	306	0	94.83
256	4	1	8	338	0	93.37
256	8	1	8	672	0	95.60
256	12	1	8	1006	0	96.39
256	16	1	8	1339	0	96.74
256	20	1	8	1673	0	96.67
↓ Kernel adjustments						
256	8	1	3	559	0	93.61
256	8	1	5	592	0	95.19
256	8	1	7	641	0	95.80
256	8	1	9	707	0	95.88
256	8	1	11	788	0	95.70
256	8	1	13	887	0	96.04
256	8	1	15	1001	0	96.08
↓ Patch adjustments						
256	8	2	9	709	0	95.00
256	8	4	9	718	0	92.61
256	8	8	9	755	0	85.57
↓ Weight decay adjustments						
256	8	1	9	707	1×10^{-1}	95.88
256	8	1	9	707	1×10^{-2}	96.03
256	8	1	9	707	1×10^{-3}	95.76
256	8	1	9	707	1×10^{-4}	95.63
256	8	1	9	707	1×10^{-5}	95.88
↓ Kernel size adjustments when $p = 2$						
256	8	2	3	561	0	94.08
256	8	2	5	594	0	95.21
256	8	2	7	643	0	95.35
256	8	2	9	709	0	95.00
256	8	2	11	791	0	95.14
↓ Adding weight decay to the above						
256	8	2	3	561	1×10^{-2}	94.69
256	8	2	5	594	1×10^{-2}	95.26
256	8	2	7	643	1×10^{-2}	95.25
256	8	2	9	709	1×10^{-2}	95.06
256	8	2	11	791	1×10^{-2}	95.17

Table 2.4: Investigation of ConvMixer design parameters h, d, p, k and weight decay on CIFAR-10

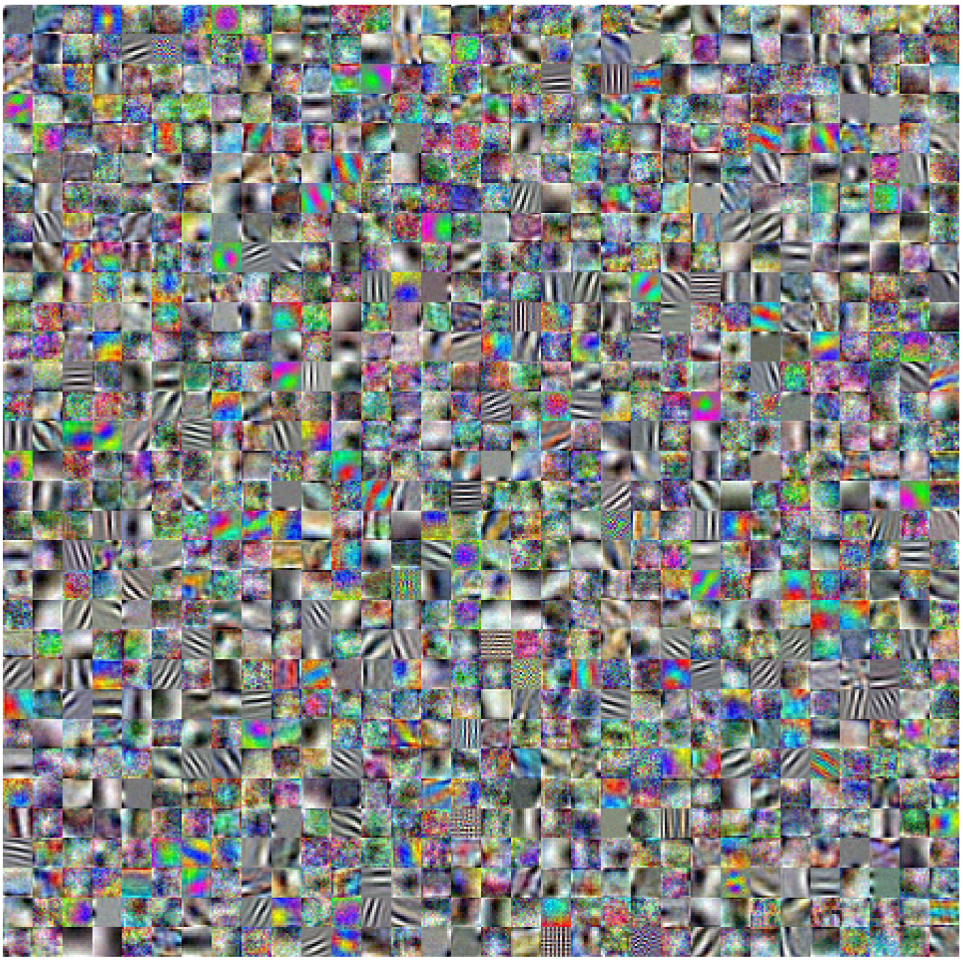


Figure 2.4: Patch embedding weights for a ConvMixer-1024/20 with patch size 14 (see Table 2.2).

In Figure 2.4 and 2.5, we visualize the (complete) weights of the patch embedding layers of a ConvMixer-1536/20 with $p = 14$ and a ConvMixer-768/32 with $p = 7$, respectively. Much like Sandler et al. (2019), the layer consists of Gabor-like filters as well as "colorful globs" or rough edge detectors. The filters seem to be more structured than those learned by MLP-Mixer (Tolstikhin et al., 2021); also unlike MLP-Mixer, the weights look much the same going from $p = 14$ to $p = 7$: the latter simply looks like a downsampled version of the former. It is unclear, then, why we see such a drop in accuracy for larger patches. However, some of the filters essentially look like noise, maybe suggesting a need for more regularization or longer training, or even more data. Ultimately, we cannot read too much into the learned representations here.

In Figure 2.6, we plot the hidden convolutional kernels for successive layers of a ConvMixer. Initially, the kernels seem to be relatively small, but make use of their allowed full size in later layers; there is a clear hierarchy of features as one would expect from a standard convolutional architecture. Interestingly, Touvron et al. (2021a) saw a similar effect for ResMLP, where earlier layers look like small-kernel convolution, while later layers were more diffuse, despite these

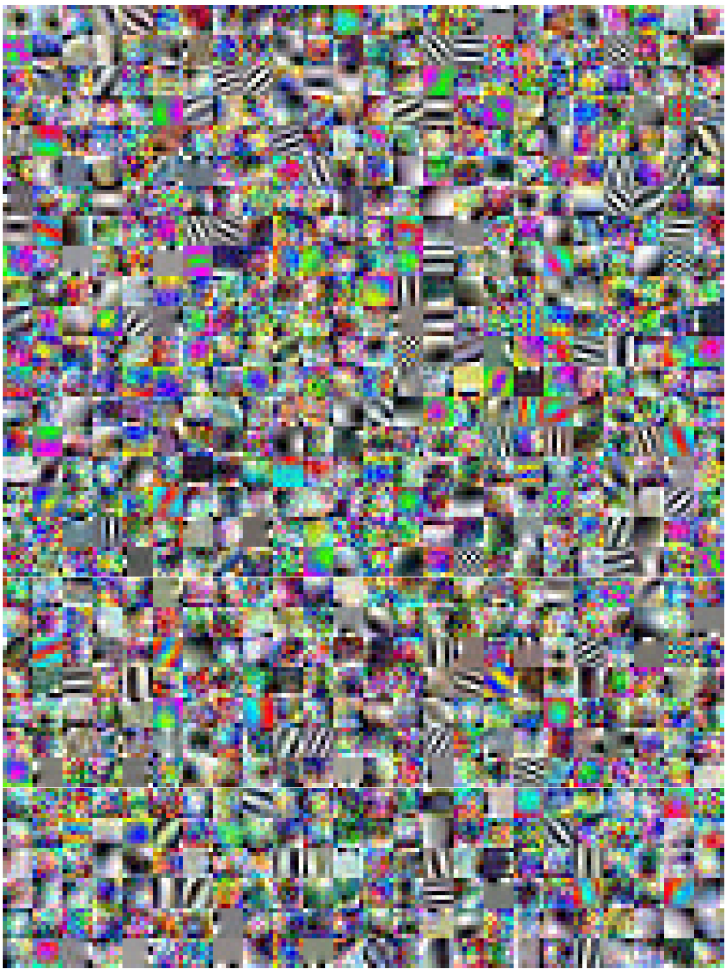


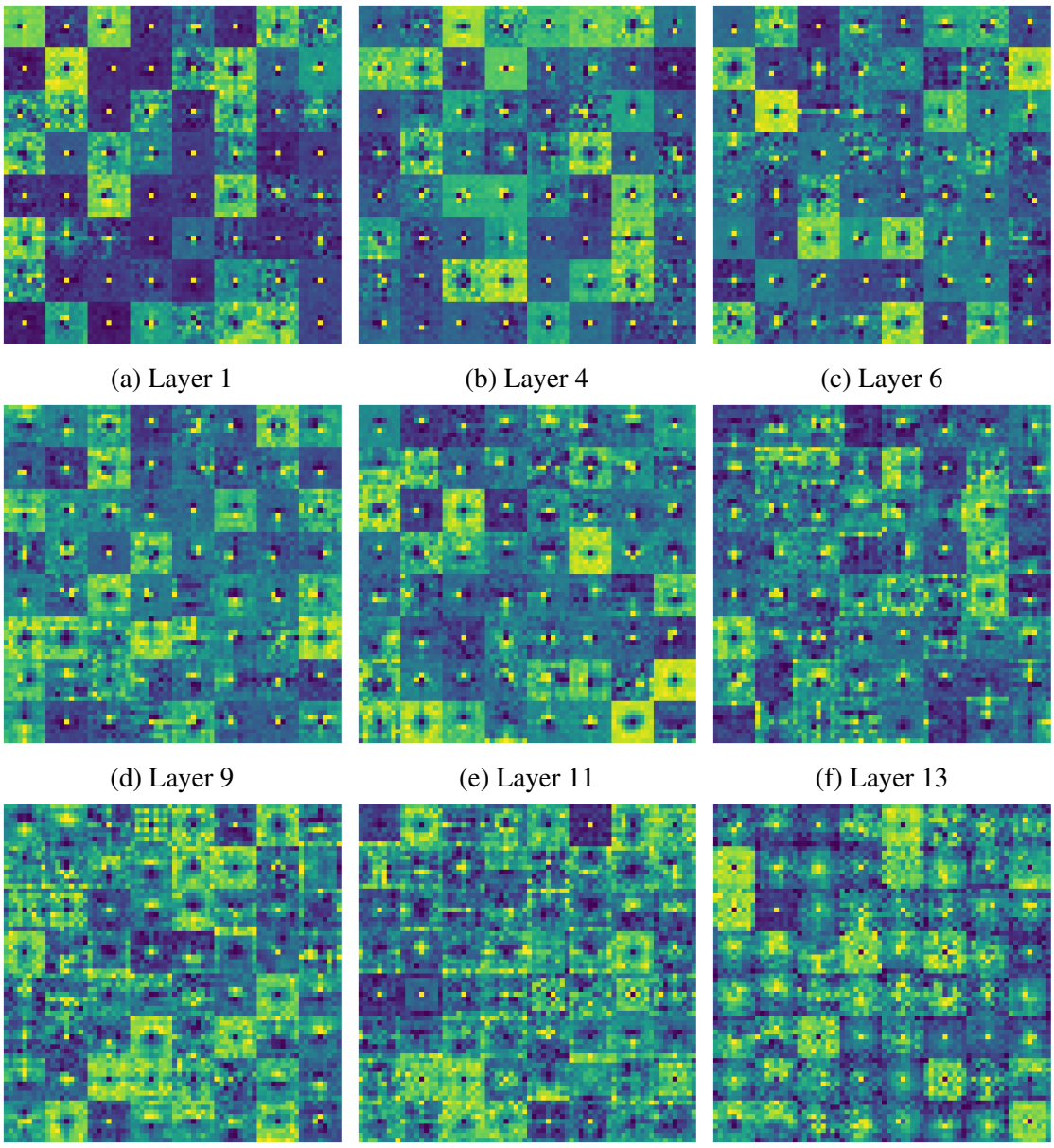
Figure 2.5: Patch embedding weights for a ConvMixer-768/32 with patch size 7 (see Table 2.2).

layers being representated by an unconstrained matrix multiplication rather than convolution.

2.1.7 Summary of contributions

We presented ConvMixers, an extremely simple class of models that independently mixes the spatial and channel locations of patch embeddings using only standard convolutions. We also highlighted that using large kernel sizes, inspired by the large receptive fields of ViTs and MLP-Mixers, provides a substantial performance boost. While neither our model nor our experiments were designed to maximize accuracy or speed, i.e., we did not search for good hyperparameters, ConvMixers outperform the Vision Transformer and MLP-Mixer, and are competitive with ResNets, DeiTs, and ResMLPs.

We provided evidence that the increasingly common "isotropic" architecture with a simple patch embedding stem is itself a powerful template for deep learning. Patch embeddings allow all the downsampling to happen at once, immediately decreasing the internal resolution and thus increasing the effective receptive field size, making it easier to mix distant spatial information.



(a) Layer 1 (b) Layer 4 (c) Layer 6

(d) Layer 9 (e) Layer 11 (f) Layer 13

(g) Layer 15 (h) Layer 17 (i) Layer 19

Figure 2.6: Random subsets of 64 depthwise convolutional kernels from progressively deeper layers of ConvMixer-1536/20 (see Table 2.1).

Our title, while an exaggeration, points out that attention isn't the only export from language processing into computer vision: tokenizing inputs, i.e., using patch embeddings, is also a powerful and important takeaway.

While our model is not state-of-the-art, we find its simple patch-mixing design to be compelling. We hope that ConvMixers can serve as a baseline for future patch-based architectures with novel operations, or that they can provide a basic template for new conceptually simple and performant models.

Future work. We are optimistic that a deeper ConvMixer with larger patches could reach a desirable tradeoff between accuracy, parameters, and throughput after longer training and more regularization and hyperparameter tuning, similarly to how Wightman et al. (2021) enhanced ResNet performance through carefully-designed training regimens. Low-level optimization of large-kernel depthwise convolution could substantially increase throughput, and small enhancements to our architecture like the addition of bottlenecks or a more expressive classifier could trade simplicity for performance.

Due to its large internal resolution and isotropic design, ConvMixer may be especially well-suited for semantic segmentation, and it would be useful to run experiments on this task with a ConvMixer-like model and on other tasks such as object detection. More experiments could be designed to more clearly extricate the effect of patch embeddings from other architectural choices. In particular, for a more in-depth comparison to ViTs and MLP-Mixers, which excel when trained on very large data sets, it is important to investigate the performance of ConvMixers in the regime of large-scale pre-training.

Chapter 3

Mimetic initialization

3.1 Understanding the Covariance Structure of Convolutional Filters

Early work in deep learning for vision demonstrated that the convolutional filters in trained neural networks are often highly-structured, in some cases being qualitatively similar to filters known from classical computer vision (Krizhevsky et al., 2017). However, for many years it became standard to replace large-filter convolutions with stacked small-filter convolutions, which have less room for any notable amount of structure. But in the past year, this trend has changed with inspiration from the long-range spatial mixing abilities of vision transformers. Some of the most prominent new convolutional neural networks, such as ConvNeXt and ConvMixer, once again use large-filter convolutions. These new models also completely separate the processing of the channel and spatial dimensions, meaning that the now-single-channel filters are, in some sense, more independent from each other than in previous models such as ResNets. This presents an opportunity to investigate the structure of convolutional filters.

In particular, we seek to understand the *statistical structure* of convolutional filters, with the goal of more effectively initializing them. Most initialization strategies for neural networks focus simply on controlling the *variance* of weights, as in Kaiming (He et al., 2015) and Xavier (Glorot and Bengio, 2010) initialization, which neglect the fact that many layers in neural networks are highly-structured, with interdependencies between weights, particularly after training. Consequently, we study the *covariance* matrices of the parameters of convolutional filters, which we find to have a large degree of perhaps-interpretable structure. We observe that the covariance of filters calculated from pre-trained models can be used to effectively initialize new convolutions by sampling filters from the corresponding multivariate Gaussian distribution.

We then propose a closed-form and completely learning-free construction of covariance matrices for randomly initializing convolutional filters from Gaussian distributions. Our initialization is *highly effective*, especially for larger filters, deeper models, and shorter training times; it usually outperforms both standard uniform initialization techniques *and* our baseline technique of initializing by sampling from the distributions of pre-trained filters, both in terms of final accuracy and time-to-convergence. Models using our initialization often see gains of over 1% accuracy on CIFAR-10 and short-training ImageNet classification; it also leads to small but sig-

nificant performance gains on full-scale, $\approx 80\%$-accuracy ImageNet training. Indeed, in some cases our initialization works so well that it outperforms uniform initialization *even when the filters aren't trained at all*. And our initialization is almost completely *free to compute.*

Related work Saxe et al. (2013) proposed to replace random *i.i.d.* Gaussian weights with random orthogonal matrices, a constraint in which weights depend on each other and are thus, in some sense, "multivariate"; Xiao et al. (2018) also proposed an orthogonal initialization for convolutions. Similarly to these works, our initialization greatly improves the trainability of deep (depthwise) convolutional networks, but is much simpler and constraint-free, being just a random sample from a multivariate Gaussian distribution. Martens et al. (2021) uses "Gaussian Delta initialization" for convolutions; while largely unrelated to our technique both in form and motivation, this is similar to our initialization as applied in the first layer (*i.e.,* the lowest-variance case). Zhang et al. (2022) suggests that the main purpose of pre-training may be to find a good initialization, and crafts a *mimicking initialization* based on observed, desirable information transfer patterns. We similarly initialize convolutional filters to be closer to those found in pre-trained models, but do so in a completely random and simpler manner. Romero et al. (2021) proposes an analytic parameterization of variable-size convolutions, based in part on Gaussian filters; while our covariance construction is also analytic and built upon Gaussian filters, we use them to specify the *distribution* of filters.

Our contribution is most advantageous for large-filter convolutions, which have become prevalent in recent work: ConvNeXt (Liu et al., 2022b) uses 7×7 convolutions, and ConvMixer (Trockman and Kolter, 2022) uses 9×9; taking the trend a step further, Ding et al. (2022) uses 31×31, and Liu et al. (2022a) uses 51×51 sparse convolutions. Many other works argue for large-filter convolutions (Wang et al., 2022; Chen et al., 2022; Han et al., 2021).

Preliminaries This work is concerned with depthwise convolutional filters, each of which is parametrized by a $k \times k$ matrix, where k (generally odd) denotes the filter's size. Our aim is to study distributions that arise from convolutional filters in pretrained networks, and to explore properties of distributions whose samples produce strong initial parameters for convolutional layers. More specifically, we hope to understand the covariance among pairs of filter parameters for fixed filter size k. This is intuitively expressed as a covariance matrix $\Sigma \in \mathbb{R}^{k^2 \times k^2}$ with block structure: Σ has $k \times k$ blocks, where each block $[\Sigma_{i,j}] \in \mathbb{R}^{k \times k}$ corresponds to the covariance between filter pixel i, j and all other $k^2 - 1$ filter pixels. That is, $[\Sigma_{i,j}]_{\ell,m} = [\Sigma_{\ell,m}]_{i,j}$ gives the covariance of pixels i, j and ℓ, m.

In practice, we restrict our study to multivariate Gaussian distributions, which by convention are considered as distributions over n-dimensional *vectors* rather than matrices, where the distribution $\mathcal{N}(\mu, \Sigma')$ has a covariance matrix $\Sigma' \in \mathbb{S}_+^n$ where $\Sigma'_{i,j} = \Sigma'_{j,i}$ represents the covariance between vector elements i and j. To align with this convention when sampling filters, we convert from our original block covariance matrix representation to the representation above by simple reassignment of matrix entries, given by

$$\Sigma'_{ki+j,k\ell+m} := [\Sigma_{i,j}]_{\ell,m} \text{ for } 1 \leq i, j, \ell, m \leq k \tag{3.1}$$

or, equivalently,

$$\Sigma'_{ki+j,:} := \text{vec}\left([\Sigma_{i,j}]\right) \text{ for } 1 \leq i, j \leq k. \tag{3.2}$$

In this form, we may now generate a filter $F \in \mathbb{R}^{k \times k}$ by drawing a sample $f \in \mathbb{R}^{k^2}$ from $\mathcal{N}(\mu, \Sigma')$ and assigning $F_{i,j} := f_{ki+j}$. Throughout the paper, we assume covariance matrices are in the block form unless we are sampling from a distribution, where the conversion between forms is assumed.

Scope We restricted our study to networks made by stacking simple blocks which each have a single *depthwise* convolutional layer (that is, filters in the layer act on each input channel separately, rather than summing features over input channels), plus other operations such as pointwise convolutions or MLPs; the *depth* of networks throughout the paper is synonymous with the number of depthwise convolutional layers, though this is not the case for neural networks more generally. All networks investigated use a fixed filter size throughout the network, though the methods we present could easily be extended to the non-uniform case. Further, all methods presented do not concern the biases of convolutional layers.

3.1.1 The empirical covariances of trained convolutional filters

In this section, we propose a simple starting point in our investigation of convolutional filter covariance structure: using the distribution of filters from *pre-trained models* to initialize filters in new models, a process we term *covariance transfer*. In the simplest case, we use a pre-trained model with exactly the same architecture as the model to be initialized; we then show that we can actually transfer filter covariances across very different models.

Basic method. We use $i \in 1, \ldots, D$ to denote the i^{th} depthwise convolutional layer of a model with D layers. We denote the $j \in 1, \ldots, H$ filters of the i^{th} pre-trained layer of the model by F_{ij} for a model with H convolutional filters in a particular layer (*i.e.,* hidden dimension H) and F' to denote the filters of a new, untrained model. Then the empirical covariance of the filters in layer i is

$$\Sigma_i = \text{Cov}[\text{vec}(F_{i1}), \ldots, \text{vec}(F_{iH})], \tag{3.3}$$

with the mean μ_i computed similarly. Then the new model can be initialized by drawing filters from the multivariate Gaussian distribution with parameters μ_i, Σ_i:

$$F'_{ij} \sim \mathcal{N}(\mu_i, \Sigma_i) \quad \text{for } j \in 1, \ldots, H, i \in 1, \ldots, D \tag{3.4}$$

Note that in this section, we use the means of the filters in addition to the covariances to define the distributions from which to initialize. However, we found that the mean can be assumed to be zero with little change in performance, and we focus solely on the covariance in later sections.

Experiment design. We test our initialization methods primarily on ConvMixer since it is simple and exceptionally easy to train on CIFAR-10. We use FFCV (Leclerc et al., 2022) for fast data loading using our own implementations of fast depthwise convolution and RandAugment (Cubuk et al., 2020). To demonstrate the performance of our methods across a variety of training times, we train for 20, 50, or 200 epochs with a batch size of 512, and we repeat all experiments with three random seeds. For all experiments, we use a simple triangular learning rate schedule with

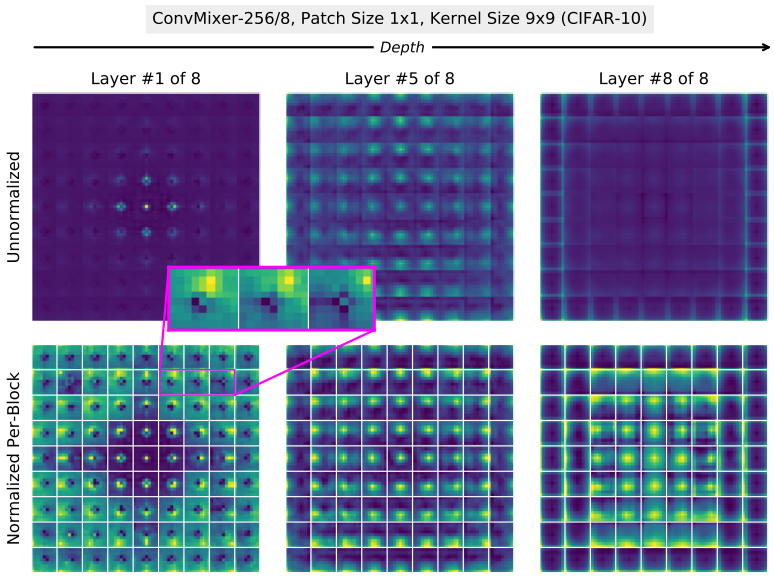


Figure 3.1: In pre-trained models, the covariance matrices of convolutional filters are *highly-structured.* Filters in earlier layers tend to be focused, becoming more diffuse as depth increases. Observing the structure of each sub-block, we note that there is often a static, centered negative component and a dynamic positive component that moves according to the block's position. Often, covariances are higher towards the center of the filters.

the AdamW optimizer, a learning rate of .01, and weight decay of .01 as in Trockman and Kolter (2022).

Most of our CIFAR experiments use a ConvMixer-256/8 with either patch size 1 or 2; a ConvMixer-H/D has precisely D depthwise convolutional layers with H filters each, ideal for testing our initial covariance transfer techniques. We train ConvMixers using popular filter sizes 3, 7, and 9, as well as 15 (see Figure 3.13 for 5). We also test our methods on ConvNeXt (Liu et al., 2022b), which includes downsampling unlike ConvMixer; we use a patch size of 1 or 2 with ConvNeXt rather than the default 4 to accomodate relatively small CIFAR-10 images, and the default 7×7 filters.

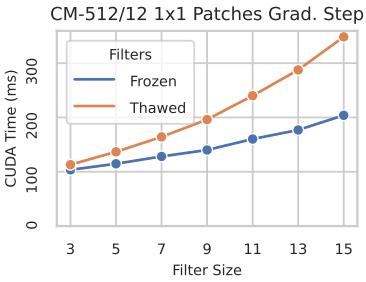


Figure 3.2: The backward pass is faster with frozen filters.

For most experiments, we provide two baselines for comparison: standard uniform initialization, the standard in PyTorch (He et al., 2015), as well as *directly* transferring the learned filters from a pre-trained model to the new model. In most cases, we expect new random initializations to fall between the performance of uniform and direct transfer initializations. For our covariance transfer experiments, we trained a variety of reference models from which to compute covariances; these are all trained for the full 200 epochs using the same settings as above.

Frozen filters. Cazenavette et al. noticed that ConvMixers with 3×3 filters perform well even when the filters are *frozen*; that is, the filter weights remain unchanged over the course of training, receiving no gradient updates. As we are initializing filters from the distribution of trained filters, we suspect that additional training may not be completely necessary. Consequently, in all experiments we investigate both models with *thawed* filters as well as their *frozen* counterparts. Freezing filters removes one of the two gradient calculations from depthwise convolution,

resulting in substantial training speedups as kernel size increases (see Figure 3.2). ConvMixer-512/12 with kernel size 9×9 is around 20% faster, while 15×15 is around 40% faster. Further, good performance in the frozen filter setting suggests that an initialization technique is highly effective.

Covariance transfer across width, depth, patch size, and kernel size

The simplest case of covariance transfer (from exactly the same architecture) is a fairly effective initialization scheme for convolutional filters. In Fig. 3.3, note that this case of covariance transfer (group **B**) results in somewhat higher accuracies than uniform initialization (group **A**), particularly for 20-epoch training; it also substantially improves the case for frozen filters. Across all trials, the effect of using this initialization is higher for larger kernel sizes. In Fig. 3.10, we show that covariance transfer (*gold*) initially increases convergence, but the advantange over uniform initialization quickly fades. As expected, covariance transfer tends to fall between the performance of *direct transfer*, where we directly initialize using the filters of the pre-trained model, and default uniform initialization (see group **D** in Fig. 3.3 and the *green* curves in Fig. 3.10).

However, we acknowledge that it is not appealing to pre-train models just for an initialization technique with rather marginal gains, so we explore the feasibility of covariance transfer from *smaller* models, both in terms of width and depth.

Narrower models. We first see if it's possible to train a narrower reference model to calculate filter covariances to initialize a wider model; for example, using a ConvMixer-32/8 to initialize a ConvMixer-256/8. In Figure 3.4, we show that the optimal performance surprisingly *comes from the covariances of a smaller model*. For filter sizes sizes greater than 3, the covariance transfer performance increases with width until width 32, and then decreases for width 256 for both the thawed and frozen cases. We plot this method in Fig. 3.3 (group **C**), and note that it almost uniformly exceeds the performance of covariance transfer from the same-sized model. Note that the method does not change; the covariances are simply calculated from a smaller sample of filters.

Shallower models. Covariance transfer from a shallow model to a deeper model is somewhat more complicated, as there is no longer a one-to-one mapping between layers. Instead, we *linearly interpolate* the covariance matrices to the desired depth. Surprisingly, we find that this technique is also highly effective: for example, for a 32-layer-deep ConvMixer, the optimal covariance transfer result is from an 8-layer-deep ConvMixer, and 2- and 4-deep models are also quite effective (see Figure 3.4).

Different patch sizes. Similarly, it is straightforward to transfer covariances between models with different patch sizes. We find that initializing ConvMixers with 1×1 patches from filter covariances of ConvMixers with 2×2 patches leads to a decrease in performance relative to using a reference model of the correct patch size; however, using the filters of a 1×1 patch size ConvMixer to initialize a 2×2 patch size ConvMixer increases performance (see group **b** vs. group **B** in Fig. 3.11). Yet, in both cases, the performance is better than uniform initialization.

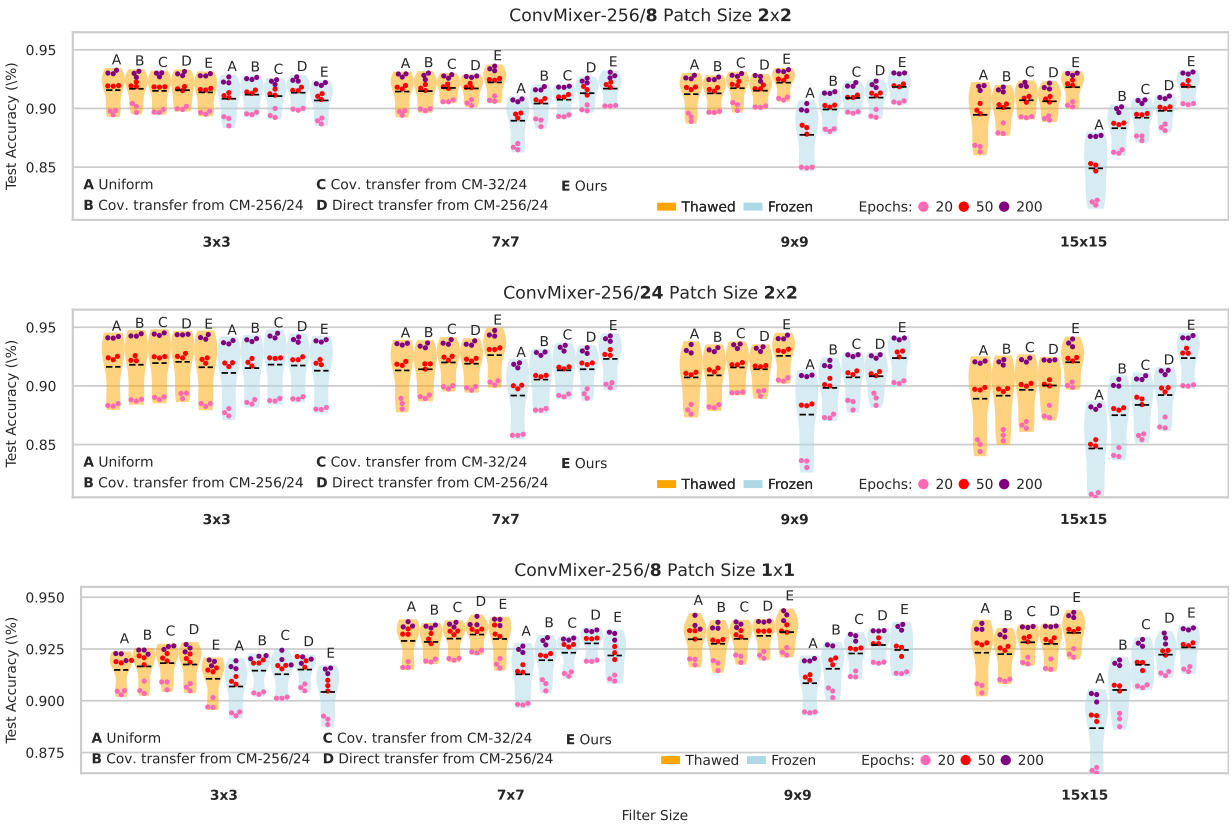


Figure 3.3: CIFAR-10 accuracy for uniform initialization (**A**), baseline covariance transfer (**B-D**), and our custom initialization results (**E**).

Different filter sizes. Covariance transfer between models with different filter sizes is more challenging, as the covariance matrices have different sizes. In the block form, we mean-pad or clip each block to the target filter size, and then bilinearly interpolate over the blocks to reach a correctly-sized covariance matrix. This technique is still better than uniform initialization for filter sizes larger than 3 (which naturally has very little structure to transfer), especially in the frozen case (see Fig. 3.11)

Kronecker-factored covariance structure. As a first step towards modeling the structure of filter covariances, we replaced covariances with their Kronecker-factorized counterparts using the rearranged form of the covariance matrix defined in Eq. (3.1), *i.e.,* $\Sigma = A \otimes A$ where $A \in \mathbb{R}^{k \times k}$. Surprisingly, this slightly improved performance over unfactorized covariance transfer (see Fig. 3.5), suggesting that filter covariances are not only eminently transferrable for initialization, but that their core structure may be simpler than meets the eye. Kronecker factorizations were computed via gradient descent minimizing the mean squared error.

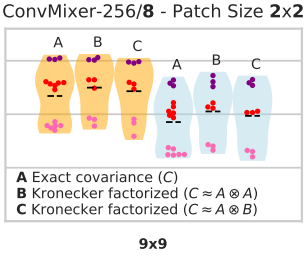


Figure 3.5: Kronecker-factorized covariances.

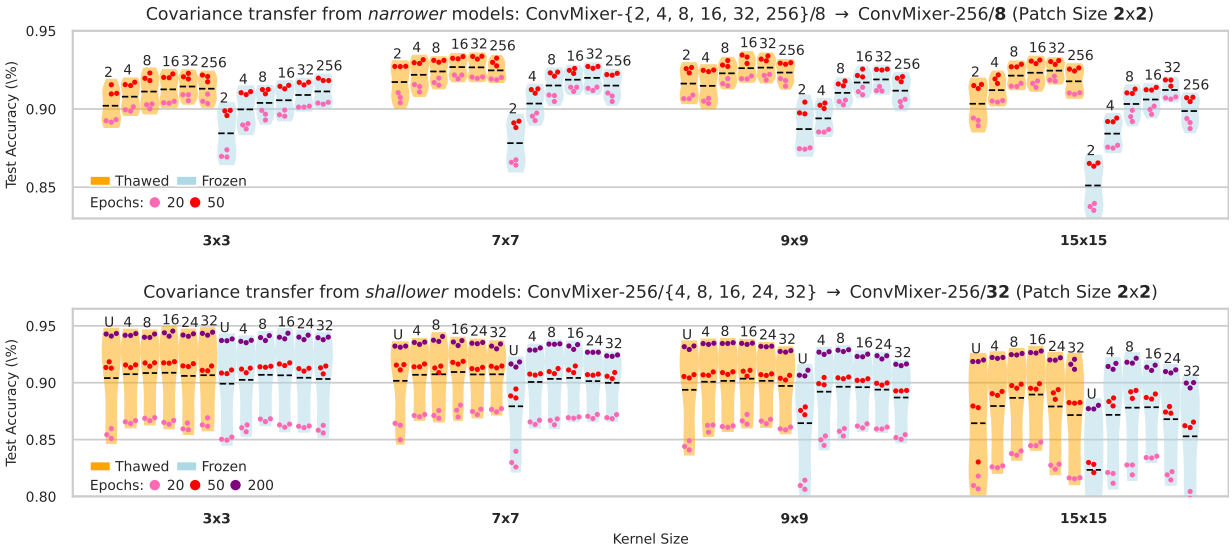


Figure 3.4: CIFAR-10 experimental results from initializing via convariances from narrower *(top)* and shallower *(bottom)* models. The numeric annotations represent the width *(top)* and depth *(bottom)* of the pre-trained model we use to intialize. **U** represents uniform initialization.

Discussion. We have demonstrated that it is possible to initialize filters from the covariances of pre-trained models of different widths, depths, patch sizes, and kernel sizes; while some of these techniques perform better than others, they are almost all better than uniform initialization. Our observations indicate that the optimal choice of reference model is narrower or shallower, and perhaps with a smaller patch size or kernel size. We also found that covariance transfer from ConvMixers trained on ImageNet led to greater performance still (Figure 3.12). This suggests that the best covariances for filter initialization may be quite unrelated to the target model, *i.e.,* model independent.

3.1.2 D.I.Y. Filter Covariances

Ultimately, the above methods for initializing convolutional filters via transfer are limited by the necessity of a trained network from which to form a filter distribution, which must be accessible at initialization. We thus use observations on the structure of filter covariance matrices to construct our own covariance matrices from scratch. Using our construction, we propose a depth-dependent but simple initialization strategy for convolutional filters that greatly outperforms previous techniques.

Visual observations. Filter covariance matrices in pre-trained ConvMixers and ConvNeXts have a great deal of structure, which we observe across models with different patch sizes, architectures, and data sets; see Fig. 3.1 and 3.15 for examples. In both the block and rearranged forms of the covariance matrices, we noticed clear repetitive structure, which led to an initial investigation on modeling covariances via Kronecker factorizations; see Figure 3.5 for experimental results. Beyond this, we first note that the overall variance of filters tends to increase with depth,

until breaking down towards the last layer. Second, we note that the sub-blocks of the covariances often have a *static* negative component in the center, with a *dynamic* positive component whose position mirrors that of the block itself. Finally, the covariance of filter parameters is greater in their center, *i.e.,* covariance matrices are at first centrally-focused and become more diffuse with depth. These observations agree with intuition about the structure of convolutional filters: most filters have the greatest weight towards their center, and their parameters are correlated with their neighbors.

Constructing covariances. With these observations in mind, we propose a construction of covariance matrices. We fix the (odd) filter size $k \in \mathbb{N}^+$, let $\mathbf{1} \in \mathbb{R}^{k\times k}$ be the all-ones matrix, and, as a building block for our initialization, use unnormalized Gaussian-like filters $Z_\sigma \in \mathbb{R}^{k\times k}$ with a single variance parameter σ, defined elementwise by

$$(Z_\sigma)_{i,j} := \exp\left(-\frac{(i-\lfloor\frac{k}{2}\rfloor)^2+(j-\lfloor\frac{k}{2}\rfloor)^2}{2\sigma}\right) \text{ for } 1 \le i,j, \le k. \tag{3.5}$$

Such a construction produces filters similar to those observed in the blocks of the Layer #5 covariance matrix in Fig. 3.1.

To capture the *dynamic* component that moves according to the position of its block, we define the block matrix $C \in \mathbb{R}^{k^2\times k^2}$ with $k \times k$ blocks by

$$[C_{i,j}] = \text{Shift}(Z_\sigma, i-\lfloor\tfrac{k}{2}\rfloor, j-\lfloor\tfrac{k}{2}\rfloor) \tag{3.6}$$

where the Shift operation translates each element of the matrix i and j positions forward in their respective dimensions, wrapping around when elements overflow; see Appendix B.2 for details. We then define two additional components, both constructed from Gaussian filters: a *static* component $S = \mathbf{1}\otimes Z_\sigma \in \mathbb{R}^{k^2\times k^2}$ and a blockwise mask component $M = Z_\sigma \otimes \mathbf{1} \in \mathbb{R}^{k^2\times k^2}$, which encodes higher variance as pixels approach the the center of the filter.

Using these components and our intuition, we first consider $\hat{\Sigma} = M \odot (C - \frac{1}{2}S)$, where $\odot$ is an elementwise product. While this adequately represents what we view to be the important structural components of filter covariance matrices, it does not satisfy the property $[\Sigma_{i,j}]_{\ell,m} = [\Sigma_{\ell,m}]_{i,j}$ (*i.e.,* covariance matrices must be symmetric, accounting for our block representation). Consequently, we instead calculate its symmetric part, using the notation as follows to denote a "block-transpose":

$$\Sigma^B = \Sigma' \iff [\Sigma_{i,j}]_{\ell,m} = \left[\Sigma'_{\ell,m}\right]_{i,j} \text{ for } 1 \le i,j,\ell,m \le k. \tag{3.7}$$

Equivalently, this is the perfect shuffle permutation such that $(X\otimes Y)^B = Y \otimes X$ with $X, Y \in \mathbb{R}^{k\times k}$. First, we note that $C^B = C$ due to the definition of the shift operation used in Eq. 3.6 (see Appendix B.2). Then, noting that $S^B = M$ and $M^B = S$ by the previous rule, we define our construction of Σ to be the symmetric part of $\hat{\Sigma}$ (where C, S, M are implicitly parameterized by σ, similarly to Z_σ):

$$\Sigma = \tfrac{1}{2}(\hat{\Sigma} + \hat{\Sigma}^B) = \tfrac{1}{2}\left[M \odot (C - \tfrac{1}{2}S) + (M \odot (C - \tfrac{1}{2}S))^B\right] \tag{3.8}$$

$$= \tfrac{1}{2}\left[M \odot (C - \tfrac{1}{2}S) + (M^B \odot (C^B - \tfrac{1}{2}S^B))\right] = M \odot (C - \tfrac{1}{2}S) + S \odot (C - \tfrac{1}{2}M) \tag{3.9}$$

$$= \tfrac{1}{2}\left[M \odot (C - S) + S \odot C\right]. \tag{3.10}$$

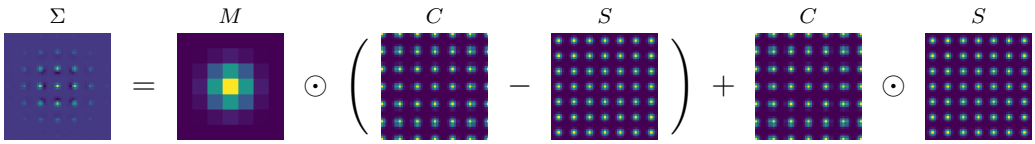


Figure 3.6: Our convolutional covariance matrix construction with $\sigma = \pi/2$.

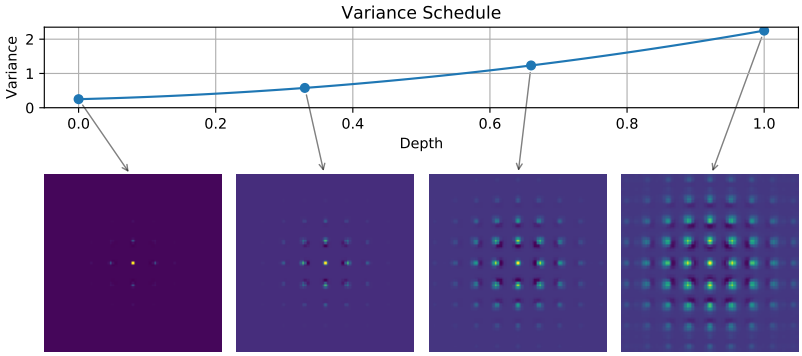


Figure 3.7: How our initialization changes with depth. Variance increases quadratically with depth according to a schedule which can be chosen through visual inspection of pre-trained models or through grid search. Here we use the parameters $\sigma_0 = .5$, $v_\sigma = .5$, $a_\sigma = 3$.

While Σ is now symmetric (in the rearranged form of Eq. 3.1), it is not positive semi-definite, but can easily be projected to $\mathbb{S}_+^{k^2}$, as is often done automatically by multivariate Gaussian procedures. We illustrate our construction in Fig. 3.6, and provide an implementation in Fig. A.3.

Completing the initialization. As explained in Fig. 3.1, we observed that in pre-trained models, the filters become more "diffuse" as depth increases; we capture this fact in our construction by increasing the parameter σ with depth according to a simple quadratic schedule; let d be the percentage depth, i.e., $d = \frac{i-1}{D-1}$ for the i^{th} convolutional layer of a model with D total such layers. Then for layer i, we parameterize our covariance construction by a *variance schedule*:

$$\sigma(d) = \sigma_0 + v_\sigma d + \tfrac{1}{2} a_\sigma d^2 \tag{3.11}$$

where $\sigma_0, v_\sigma, a_\sigma$ jointly describe how the covariance evolves with depth. Then, for each layer $i \in 1, \ldots, D$, we compute $d = \frac{i-1}{D-1}$ and initialize the filters as $F_{i,j} \sim \mathcal{N}(0, \Sigma'_{\sigma(d)})$ for $j \in 1, \ldots, H$. We illustrate our complete initialization scheme in Figure 3.7.

3.1.3 Initializing using our filter covariance structure

In this section, we present the performance of our initialization within ConvMixer and ConvNeXt on CIFAR-10 and ImageNet classification, finding it to be highly effective, particularly for deep models with large filters. Our new initialization overshadows our previous covariance transfer results.

Settings of initialization hyperparameters σ_0, v_σ, and a_σ were found and fixed for CIFAR-10 experiments, while two such settings were used for ImageNet experiments. Appendix B.1 contains full details on our (relatively small) hyperparameter searches and experimental setups, as well as empirical evidence that our method is *robust to a large swath of hyperparameter settings.*

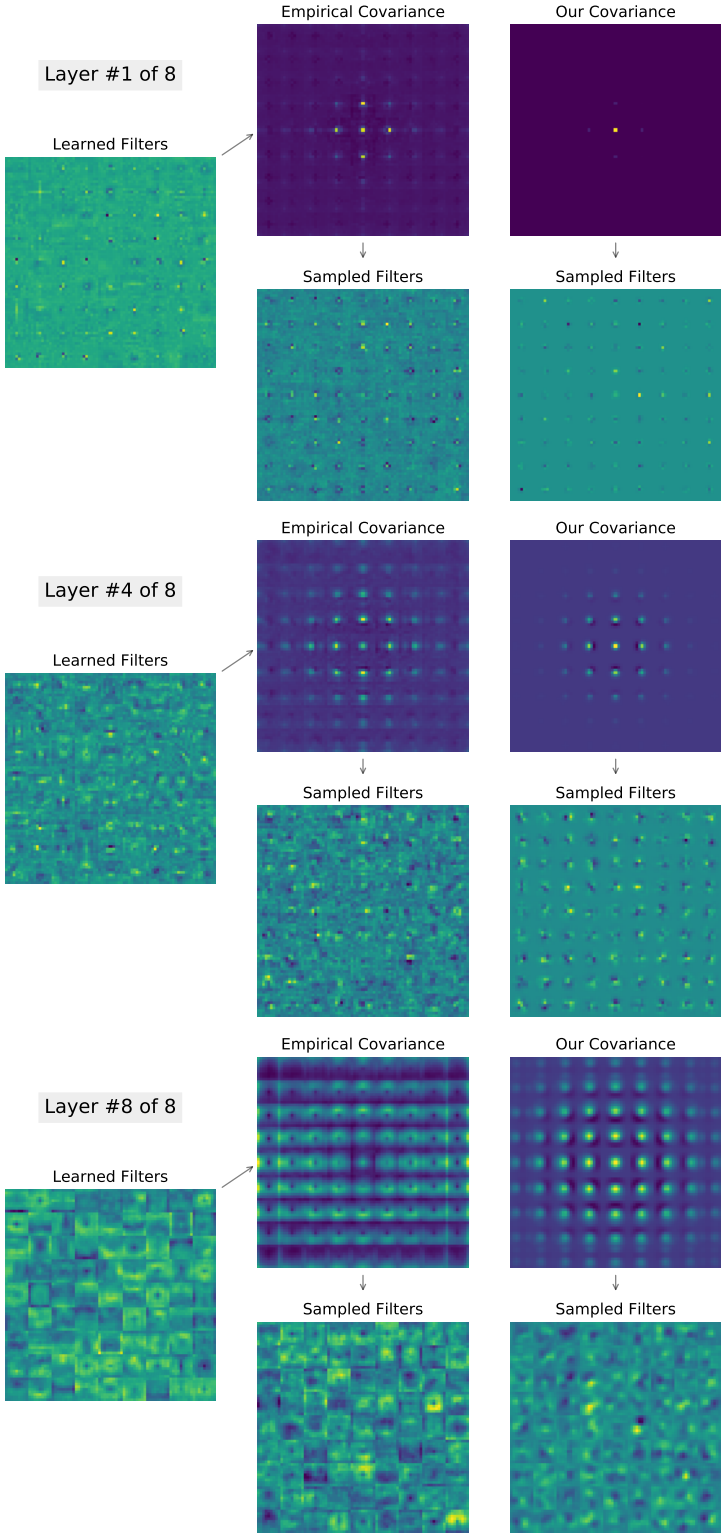


Figure 3.8: Filters learned or generated for ConvMixer-256/8 with 2×2 patches and 9×9 filters trained on CIFAR-10: learned filters (*left*), filters sampled from the Gaussian defined by the empirical covariance matrix of learned filters (*center*), and filters from our initialization technique (*right*).

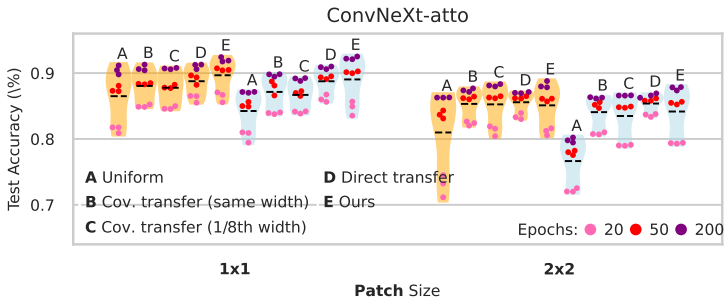


Figure 3.9: Our init also improves ConvNeXt's accuracy on CIFAR-10 (group **E** *vs.* **A**).

CIFAR-10 Results

Thawed filters. In Fig. 3.3, we show that large-kernel models using our initialization (group **E**) outperform those using uniform initialization (group **A**), covariance transfer (groups **B, C**), and even those directly initializing via learned filters (group **D**). For 2×2-patch models (200 epochs), relative to uniform, our initialization causes up to a 1.1% increase in accuracy for ConvMixer-256/8, and up to 1.6% for ConvMixer-256/24. The effect size increases with the the filter size, and is often more prominent for shorter training times. Results are similar for 1×1-patch models, but with a smaller increase for 7×7 filters (0.15% vs. 0.5%). Our initialization has the same effects for ConvNeXt (Fig. 3.9). However, our method works poorly for 3×3 filters, which we believe have fundamentally different structure than larger filters; this setting is better-served by our original covariance transfer techniques.

In addition to improving the final accuracy, our initialization also drastically speeds up convergence of models with thawed filters (see Fig. 3.10), particularly for deeper models. A ConvMixer-256/16 with 2×2 patches using our initialization reaches 90% accuracy in approximately 50% fewer epochs than uniform initialization, and around 25% fewer than direct learned filter transfer. The same occurs, albeit to a lesser extent, for 1×1 patches—but note that for this experiment we used the same initialization parameters for both patch sizes to demonstrate robustness to parameter choices.

Frozen filters. Our initialization leads to even more surprising effects in models with frozen filters. In Fig. 3.3, we see that frozen-filter 2×2-patch models using our initialization often *exceed the performance of their uniform, thawed-filter counterparts* by a significant margin of 0.4% – 2.0% for 200 epochs, and an even larger margin of 0.6% – 5.0% for 20 epochs (for large filters). That is, group **E** *(frozen)* consistently outperforms groups **A-D** *(thawed)*, and in some cases even group **E** *(thawed)*, especially for the deeper 24-layer ConvMixer. While this effect breaks down for 1×1 patch models, such frozen-filter models still see accuracy increases of 0.6%–3.5%. However, the effect can still be seen for 1×1-patch ConvNeXts (Fig. 3.9). Also note that frozen-filter models can be up to 40% faster to train (see Fig. 3.2), and may be more robust (Cazenavette et al.).

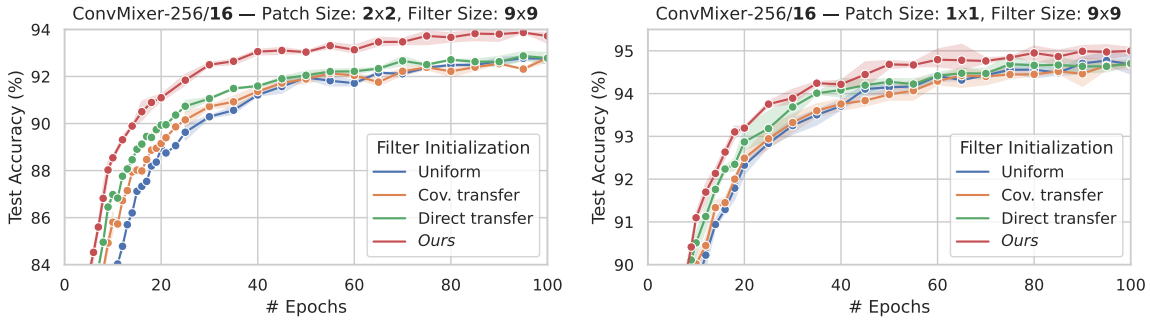


Figure 3.10: Convergence plots: each data point runs through a full cycle of the LR schedule, and all points are averaged over three trials with shaded standard deviation.

Table 3.1: ImageNet-1k accuracy from various architectures and initializations. "Ours" denotes our proposed initialization. **Bold** indicates best within architecture and category (frozen or thawed).

Model				THAWED			FROZEN		
Architecture	Filter Size	Patch Size	# Epochs	Uniform	Ours .15 .5 .25	Ours .15 .25 1.0	Uniform	Ours .15 .5 .25	Ours .15 .25 1.0
ConvMixer-512/12	9	14	50	67.03	**67.41**	67.34	60.47	**64.43**	64.12
ConvMixer-512/24	9	14	50	67.76	**69.60**	69.52	62.50	**66.57**	66.38
ConvMixer-512/32	9	14	50	65.00	68.78	**68.84**	55.79	**66.59**	66.32
ConvMixer-1024/12	9	14	50	73.55	73.62	**73.75**	68.96	**71.48**	71.30
ConvMixer-1024/24	9	14	50	74.19	75.33	**75.50**	69.65	**73.42**	74.31
ConvMixer-1024/32	9	14	50	72.18	**74.98**	74.95	64.94	73.00	**73.12**
ConvMixer-512/12	9	7	50	72.05	71.92	**72.32**	67.25	68.91	**68.92**
ConvNeXt-Atto	7	4	50	**69.96**	67.84	68.06	51.43	**64.52**	64.43
ConvNeXt-Tiny	7	4	50	75.99	76.08	**77.11**	64.17	74.62	**75.21**
ConvMixer-1536/24	9	14	150	80.11		**80.28**			
ConvNeXt-Tiny	7	4	150	79.74		**79.81**			

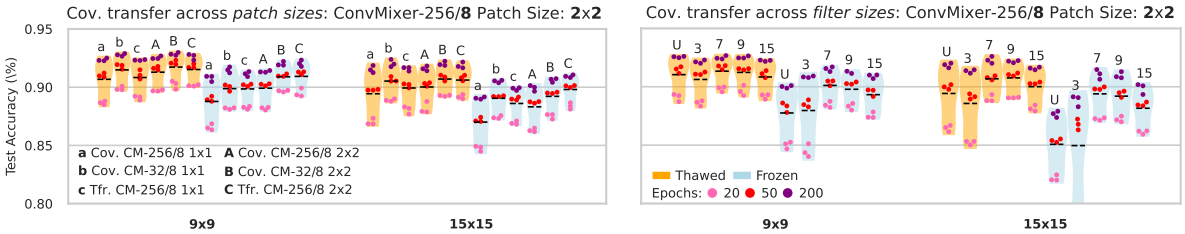


Figure 3.11: Initializing via covariances from models with different patch (*left*) and filter sizes (*right*). *Left:* Lowercase denotes initializing from patch size 1×1, and uppercase 2×2. *Right:* Annotations denote the reference filter size, **U** is uniform.

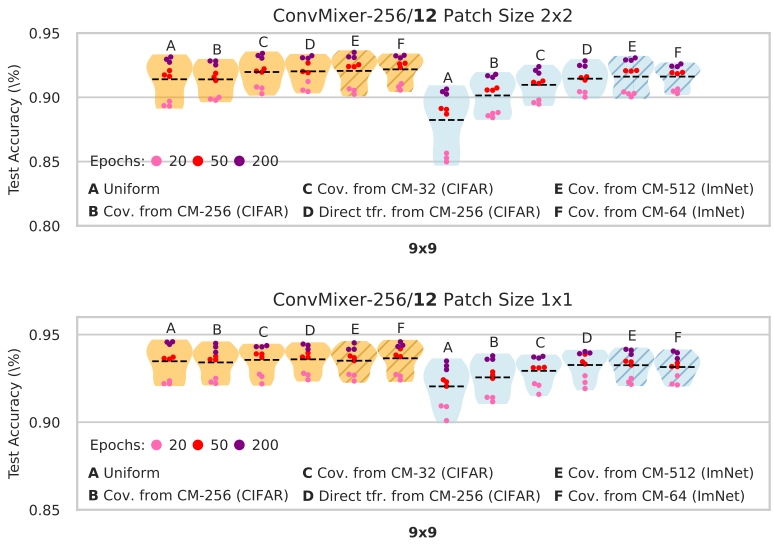


Figure 3.12: Using filter distributions from pre-trained ImageNet models to initialize models trained on CIFAR-10 is also effective (represented by groups **E** and **F**, with hatch marks).

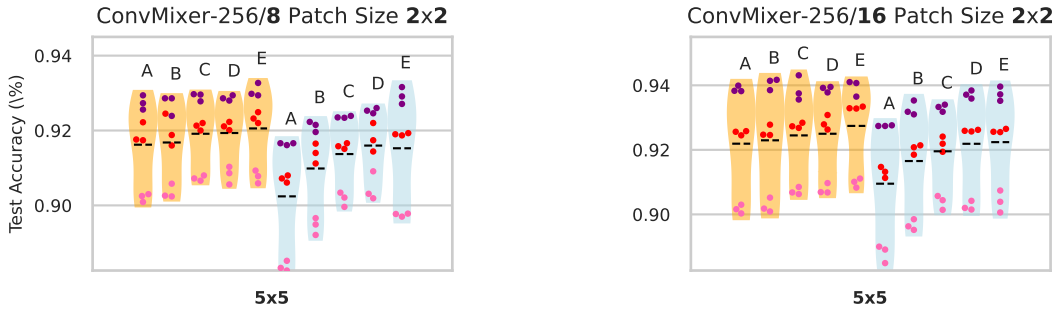


Figure 3.13: Our initialization is also effective for 5×5 filters. (The same legends in Fig. 3.3 apply.)

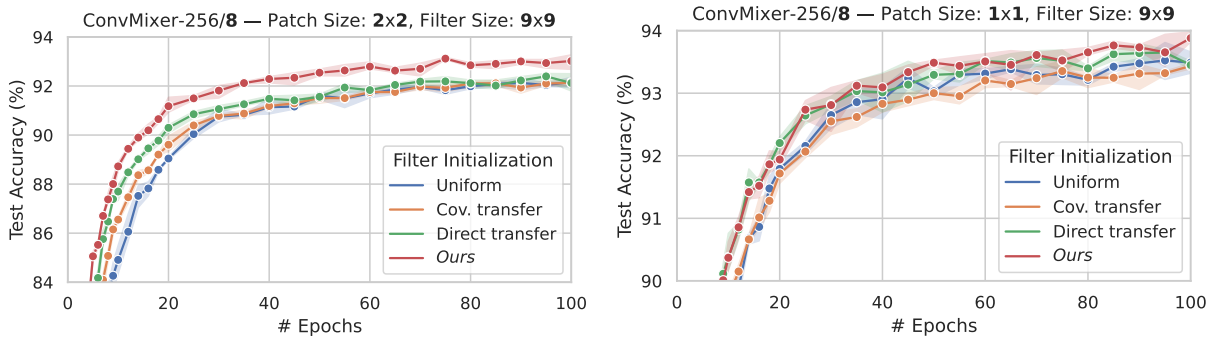


Figure 3.14: Convergence plots: each data point runs through a full cycle of the LR schedule, and all points are averaged over three trials with shaded standard deviation.

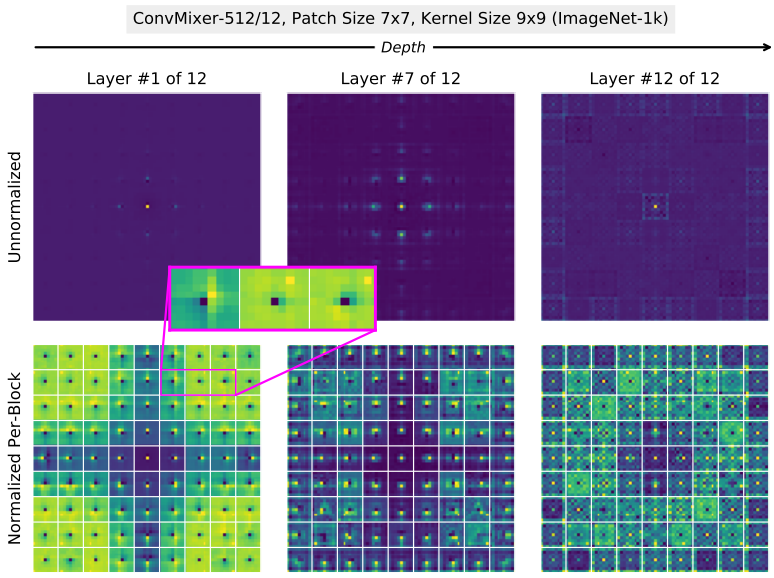


Figure 3.15: Covariance matrices from a ConvMixer trained on ImageNet exhibit similar structure to those of ConvMixers trained on CIFAR-10; however, later layers tend to have more structure, including a "checkerboard" pattern in each sub-block.

ImageNet Experiments

Our initialization performs extremely well on CIFAR-10 for large-kernel models, almost always helping and rarely hurting. Here, we explore if the performance gains transfer to larger-scale ImageNet models. We observe in Fig. 3.15, Appendix B.3 that filter covariances for such models have finer-grained structure than models trained on CIFAR-10, perhaps due to using larger patches. Nonetheless, our initialization leads to quite encouraging improvements in this setting.

Experiment design. We used the "A1" training recipe from Wightman et al. (2021), with cross-entropy loss, fewer epochs, and a triangular LR schedule as in Trockman and Kolter (2022). We primarily demonstrate our initialization for 50-epoch training, as the difference between initializations is most pronounced for lower training times. We also present two full, practical-scale 150-epoch experiments on large models. We also included covariance transfer experiments in Appendix B.3.

Thawed filters. On models trained for 50 epochs with thawed filters, our initialization improves the final accuracy by $0.4\% - 3.8\%$ (see Table 3.1). For the relatively-shallow ConvMixer-512/12 on which we tuned the initialization parameters, we see a gain of just 0.4%; however, when increasing the depth to 24 or 32, we see larger gains of 1.8% and 3.8%, respectively, and a similar trend among the wider ConvMixer-1024 models. Our initialization also boosts the accuracy of the 18-layer ConvNeXt-Tiny from 76.0% to 77.1%; however, it decreased the accuracy of the smaller, 12-layer ConvNeXt-Atto. This is perhaps unsurprising, seeing as our initialization seems to be more helpful for deep models, and we used hyperparameters optimized for a model with a substantially different patch and filter size.

Our initialization is also beneficial for more-practical 150-epoch training, boosting accuracy by around 0.1% on both ConvMixer-1536/24 and ConvNeXt-Tiny (see Table 3.1, bottom rows). While the effect is small, this demonstrates that our initialization is still helpful even for longer

training times and very wide models. We expect that within deeper models and with slightly more parameter tuning, our initialization could lead to still larger gains in full-scale ImageNet training.

Frozen filters. Our initialization is extremely helpful for models with frozen filters. Using our initialization, the difference between thawed and frozen-filter models decreases with increasing depth, *i.e.,* it leads to $2 - 11\%$ improvements over models with frozen, uniformly-initialized filters. For ConvMixer-1024/32, the accuracy improves from 64.9% to 73.1%, which is over 1% *better than the corresponding thawed, uniformly-initialized model*, and only 2% from the best result using our initialization. This mirrors the effects we saw for deeper models on our earlier CIFAR-10 experiments. We see a similar effect for ConvNeXt-Tiny, with the frozen version using our initialization achieving 75.2% accuracy *vs.* the thawed 76.0%. In other words, our initialization so effectively captures the structure of convolutional filters that it is hardly necessary to train them after initialization; one benefit of this is that it substantially speeds up training for large-filter convolutions.

3.1.4 An efficient convolutional filter dilation schedule

An important detail of the previously-presented initialization is that the variance of the filters increase with depth within the network; see Figure 3.7 and Appendix B.1.1. We investigate whether this observation could have broader consequences for efficient neural architecture design. In particular, increasing the variance of the filters corresponds to increasing their "receptive field", so one could use smaller and more efficient 3×3 filters early in the network and larger, *e.g.*, 9×9 filters in final layers. Alternatively, the *dilation* of the filters could be increased in deeper layers, allowing for a larger receptive field without significantly changing memory- or compute-overhead (Yu and Koltun, 2015). A convolutional layer with dilation factor d inserts $d - 1$ zero or padding pixels between the filter pixels, resulting in a layer with larger receptive field. That is, a convolutional layer with kernel size $k \times k$ and dilation factor d will have a new kernel size $K \times K$, where $K = k - (k - 1)(d - 1)$.

In Figure 3.16, we show the effect of using various custom depth-dependent convolution dilation schedules on training ConvMixers on CIFAR-10; the dilation schedules are listed in Table 3.2. Schedules **A**-**C** correspond to constant dilation factors 1-3, i.e., schedule **A** is the baseline. By increasing dilation in later layers (schedules **D**-**H**), we can increase test accuracy, especially for short-duration training (pink), and in some cases even improve accuracy for long-duration training (purple; upper left). For example, schedule **E** uses $d = 1$ (no dilation) in the first 50% of the layers, then uses $d = 2$ for the next 25% and $d = 3$ for the final 25%; this mimics the increasing filter sizes we saw in the earlier initialization, as well as the tendency for unconstrained networks like ResMLP (Touvron et al., 2021a) to learn convolution-like layers with increasing receptive fields. This schedule (**E**) results in a nearly 1% improvement in accuracy for all training durations for ConvMixer-256/8 with 1×1 patches, though the effect is less obvious for larger patches and deeper networks.

This shows evidence that the observations behind mimetic initialization may be useful beyond initialization itself, *e.g.*, for efficient architecture design, by anticipating the weight structure that

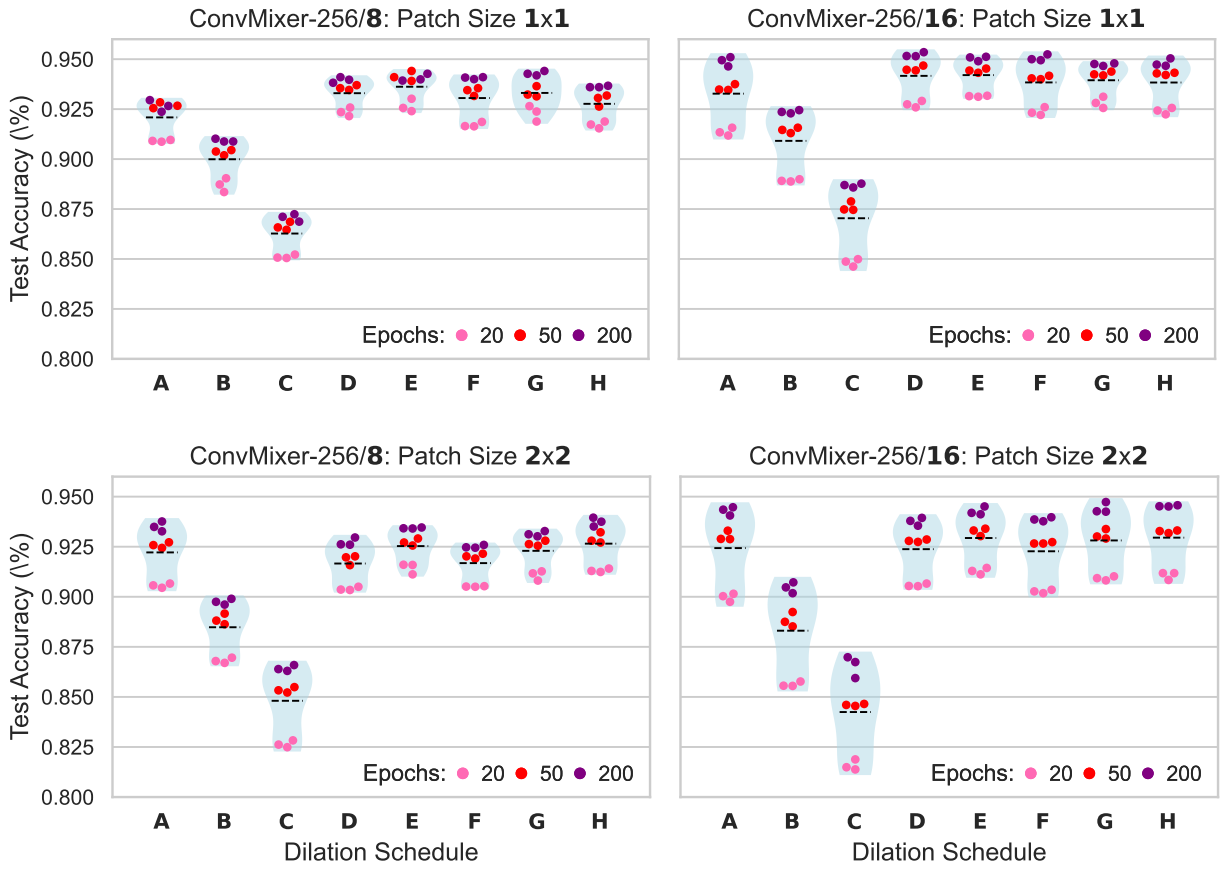


Figure 3.16: Changing the dilation schedule of 3×3 convolutions in line with our initialization observations can—to a lesser extent—improve performance, especially or short-duration training times on CIFAR-10. The effect is most obvious for ConvMixer-256/8 with patch size 1×1, in the upper left. The custom schedules (C-H) tend to either be better than or comparable to the default schedule A. Simply using larger dilations throughout (B, C) actually harms performance. See Table 3.2 for a description of the dilation schedules A-H.

final trained models "want" to have. There is similar evidence from He and Hofmann (2023), jointly with Trockman and Kolter (2023), which suggests to remove the projection weight (and a residual connection) from transformers based on the observation that the product of the value and projection weights in self-attention layers often has negative-identity-like structure; i.e., this negative identity may cancel out a residual connection in the self-attention block. Studying the structure of the weights of trained models may therefore be a powerful way to design more efficient architectures.

3.1.5 Summary of contribution

In this paper, we proposed a simple, closed-form, and learning-free initialization scheme for large depthwise convolutional filters. Models using our initialization typically reach higher accuracies more quickly than uniformly-initialized models. We also demonstrated that our random initial-

Table 3.2: Dilation schedules for an 8-layer ConvMixer — we split the network into quarters, so this may be extrapolated easily to deeper ConvMixers as well. The dilation schedules mimic the increasing filter size used in our initialization and observed in pretrained ConvMixers.

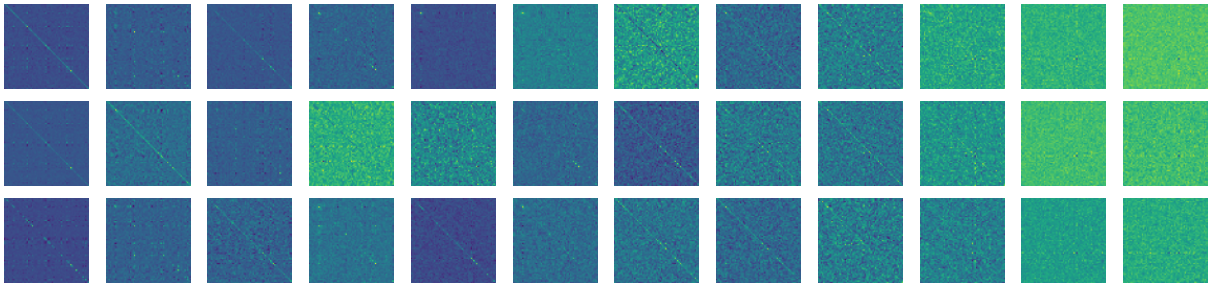
Schedule	Dilation Pattern
A	1 1 1 1 1 1 1 1
B	2 2 2 2 2 2 2 2
C	3 3 3 3 3 3 3 3
D	1 1 2 2 3 3 3 3
E	1 1 1 1 2 2 3 3
F	1 1 2 2 2 2 2 2
G	1 1 1 1 2 2 2 2
H	1 1 1 1 1 1 2 2

ization of convolutional filters is so effective, that in many cases, networks perform nearly as well (or even better) if the resulting filters do not receive gradient updates during training. Moreover, like the standard uniform initializations generally used in neural networks, our technique merely samples from a particular statistical distribution, and it is thus almost completely computationally free. ***In summary,*** *our initialization technique for the increasingly-popular large-kernel depthwise convolution operation almost always helps, rarely hurts, and is also free.*

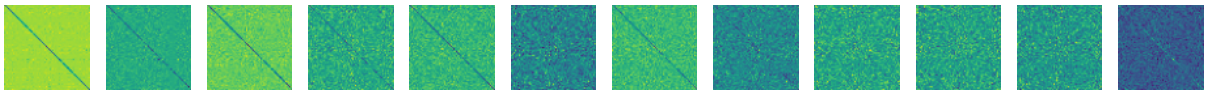
3.2 Mimetic Initialization of Self Attention Layers

Despite their excellent performance in the regime of large-scale pretraining, Transformers are notoriously hard to train on small-scale datasets Dosovitskiy et al. (2020). In this setting, convolutional networks such as the ResNet tend to massively outperform Vision Transformers, with the gap only being closed by the addition of techniques such as self-supervised pretraining, auxiliary losses, convolution-inspired tokenizers, or the addition of other architectural components that promote convolution-like inductive biases. Similar effects are seen in language modeling, where classic models such as LSTMs outperform vanilla Transformers without extreme regularization and long-duration training.

In this work, we take a step towards bridging this gap via a novel initialization technique for Transformers. We focus primarily on Vision Transformers (ViTs), though we also investigate our technique in the context of language modeling. We note that in pretrained ViTs, the weights of self-attention layers are often quite correlated, in that $W_Q W_K^T \propto I + \epsilon$ and $W_V W_{proj} \propto \epsilon - I$. Our proposal is merely to initialize the self-attention weights to mimick this observation, with the added caveat of requiring standard sinusoidal position embeddings. While we propose only one technique here, we believe that this concept is worthy of future research, as it may enhance



(a) $W_Q W_K^T$ often has a noticeable positive diagonal. $\rightarrow$ Layers 1-12



(b) $W_V W_{proj}$ often has a prominent negative diagonal. Here, we sum over heads.

Figure 3.17: Self-attention weights of an ImageNet-pretrained ViT-Tiny. Pictured are 3 heads for each of the 12 layers. Clipped to 64x64.

the understanding of the inner-workings of deep models and lead to cheaper training and better optima. We propose to call this type of technique *mimetic initialization*, as we initialize by *mimicking* the structures and patterns observed in the weights of *pretrained* models. Importantly, the sort of *mimetic initialization* we propose seeks to mimic *solely* through hand-crafted, interpretable formulas: it involves absolutely no pretraining and is practically compute-free; *i.e.,* there is no learning procedure involved.

Fundamentally, we seek to investigate the question proposed by Zhang et al. (2022): might some of the benefits of pretraining actually just be a result of it serving as a good initialization? Our approach is to attempt to find good initializations that do not involve pretraining to begin to explore this question.

Our initialization shows strong advantages for ViTs, allowing gains of up to 5% when training on small datasets like CIFAR-10, and up to 4% for larger datasets, *i.e.,* ImageNet-1k within a standard ResNet-style training pipeline. We also see smaller performance gains on language modeling tasks such as WikiText-103.

TOOD

3.2.1 The difficulty of training Vision Transformers

It is conventional wisdom that CNNs have a stronger inductive bias than ViTs. In practice, this means that CNNs perform particularly well on small datasets, while ViTs only surpass their performance when pretrained on very large (*e.g.,* ImageNet-21k- or JFT-300B-scale) datasets. To remedy this situation, numerous works have proposed to integrate convolutions explicitly into ViTs: Dai et al. (2021) introduces CoAtNet, which directly integrates depthwise convolution and self-attention. Wu et al. (2021) introduces CvT, a Transformer modification involving convolutional tokenization and projections. Yuan et al. (2021a) proposes the Convolution-enhanced Image Transformer (CeiT), which makes various modifications to bring about CNN-like induc-

tive bias. These techniques are uniformly effective: ViT/CNN hybrids tend to achieve higher accuracies with less data than their vanilla ViT counterparts. In contrast to these works, we seek to make ViTs more trainable *without* the use of convolutions, guided by the observation that pretrained ViTs eventually become effective without them given sufficient training time.

There are relatively few works on initializing Transformers; these works tend to be theoretical, focusing on eliminating normalization or skip connections. Huang et al. (2020) investigates training Transformers without learning rate warmup and normalization, and proposed a rescaling of weights that allows these to be removed. He et al. (2023) extends work on Deep Kernel Shaping to train Transformers without normalization and skip connections. Rather than initializing W_Q, W_K in a particularly structured or principled way, they ensure the product is zero and instead add a controllable bias inside the softmax of the self-attention layers. Similarly, Zhao et al. (2021) proposes to set the query and key weights to zero and the identity, respectively; however, the product of these weights remains zero.

In contrast, we attempt to better-initialize standard *vanilla* Transformers, which use skip connections and normalization. Moreover, we do so by controlling the behavior of the query and key weights themselves, aiming to replicate the behavior of pretrained models without any training.

Touvron et al. (2021c) proposes LayerScale, which multiplies the skip connections by a learnable diagonal matrix; though this is an actual architectural change and not an initialization, we will discuss the potential (albeit weak) connection to our initialization in Sec. 3.2.5. Cordonnier et al. (2019) and d'Ascoli et al. (2021) propose a scheme to initialize self-attention to implement convolution; however, this requires the use of relative positional embeddings and the (gated) self-attention layers proposed must have a particular number of heads to match the kernel size. In contrast, our scheme makes no architectural changes to the Transformer and still achieves comparable performance. Importantly, we do not seek to make self-attention emulate convolution explicitly, but rather emulate the behavior of self-attention *itself* after large-scale pretraining.

An inspiration for our work, Zhang et al. (2022) proposed a so-called "mimicking initialization" as an alternative to large-scale pretraining for language models. However, this technique actually *trains* self-attention layers to mimick the behavior of a handcrafted, convolution-like target similar to attention maps seen in trained models; in contrast, we attempt to bring about desirable behavior of self-attention entirely by hand, without any form of training. In that sense, our method is vaguely similar in spirit to Trockman et al. (2022), who propose a learning-free, structured multivariate initialization for convolutional filters.

Many works have modified Vision Transformers to more effectively train on small-scale datasets. Gani et al. (2022) proposes to learn the weight initialization in a self-supervised fashion, noting that ViTs are highly sensitive to initialization. This achieves good results on CIFAR-10 and other small-scale datasets. Cao et al. (2022) proposes another self-supervised technique for from-scratch training. Hassani et al. (2021) proposes a Compact Convolutional Transformer that can perform well on small datasets, which involves the use of a convolutional tokenizer. Lee et al. (2021) improves performance on small-scale datasets by introducing Shifted Patch Tokenization and Locality Self-Attention. Liu et al. (2021b) proposes a "dense relative localization" auxiliary task which improves the performance of transformers on small-scale datasets. In contrast to these works, which introduce auxiliary tasks or novel components, we use standard ResNet-style training and still achieve good results on small datasets with *completely vanilla Transformers*.

3.2.2 Query and key weights are correlated in pretrained models

Preliminaries We denote the query and key weight matrices for a single head of self-attention by $W_Q, W_K \in \mathbb{R}^{d\times k}$, where d is the dimension (or width) of the Transformer and $k = d/\#\text{heads}$ is the head dimension. We consider the value and projection matrices to be full-rank: $W_V, W_{proj} \in \mathbb{R}^{d\times d}$. For inputs $X \in \mathbb{R}^{n\times d}$ with additive positional embeddings $P \in \mathbb{R}^{n\times d}$, we denote the "attention map" as follows:

$$\mathsf{Softmax}\left(\tfrac{1}{\sqrt{k}} X W_Q W_K^T X^T\right).$$

Our initialization is based on *mimicking* the patterns we observed in pre-trained vision transformers. In Fig. 3.17, we visualize said patterns for a ViT-Tiny, pretrained on ImageNet. The diagonal of the product of W_Q and W_K^T is noticeably positive in many cases. Similarly, and somewhat surprisingly, the product of W_V and W_{proj} tends to have a noticeably negative diagonal. This similarly holds for ViTs of different sizes. This suggests that, in rough approximation, W_Q and W_K may be the "same" low-rank random normal matrix, as such matrices are approximately semi-orthogonal. This is based on the fact that an appropriately-scaled random normal matrix is approximately orthogonal. That is, if $Z \in \mathbb{R}^{d\times k}$ and $Z \sim \mathcal{N}(0, I/k)$, then $ZZ^T \approx I$. On language models (see Fig. 3.23), we see a similar, albeit not quite so clear pattern. In contrast, the products W_Q and W_K^T are often negative instead of positive, and vice versa for W_V and W_{proj}.

In Figure 3.18, we show the attention maps in a ViT-Tiny for a variety of training settings, averaged over the three heads and over a batch of CIFAR-10 inputs. Note the difference between the untrained model (a) and the untrained one using our initialization (d). Further, there is some degree of similarity between the ImageNet-pretrained model (c) and our untrained one (d). After training our initialized ViT on CIFAR-10, the early layers are similar to those of the ImageNet-pretrained ViT while the later layers are more like those of the only-CIFAR-trained ViT (b). The last layers of the ImageNet-pretrained ViT implement a kind of broadcasting operation which we do not attempt to mimick.

3.2.3 Mimetic init for self-attention layers

We note that there are two relatively simple choices for modeling $W_Q W_K^T$ and $W_V W_{proj}$. The simplest technique is to merely set the two matrices in the product to the same random normal matrix, *i.e.*, $W_Q = W_K = N(0, I/k)$, which is scaled by the Transformer head dimension k so that the average magnitude of the diagonal is ≈ 1. In the case of the value/projection matrices, whose diagonal we want to be negative, this would be

$$Z \coloneqq N(0, I/d), W_V = Z, W_{proj} = -Z.$$

However, no matter how we scale the random normal matrix, the ratio between the magnitude of the on-diagonal and the off-diagonal noise remains the same.

To gain more flexibility in the prominence of the diagonal, we instead propose to use a slightly more involved technique. Here, we explicitly model the products as follows:

$$W_Q W_K^T \approx \alpha_1 Z_1 + \beta_1 I \tag{3.12}$$

$$W_V W_{proj}^T \approx \alpha_2 Z_2 - \beta_2 I \tag{3.13}$$

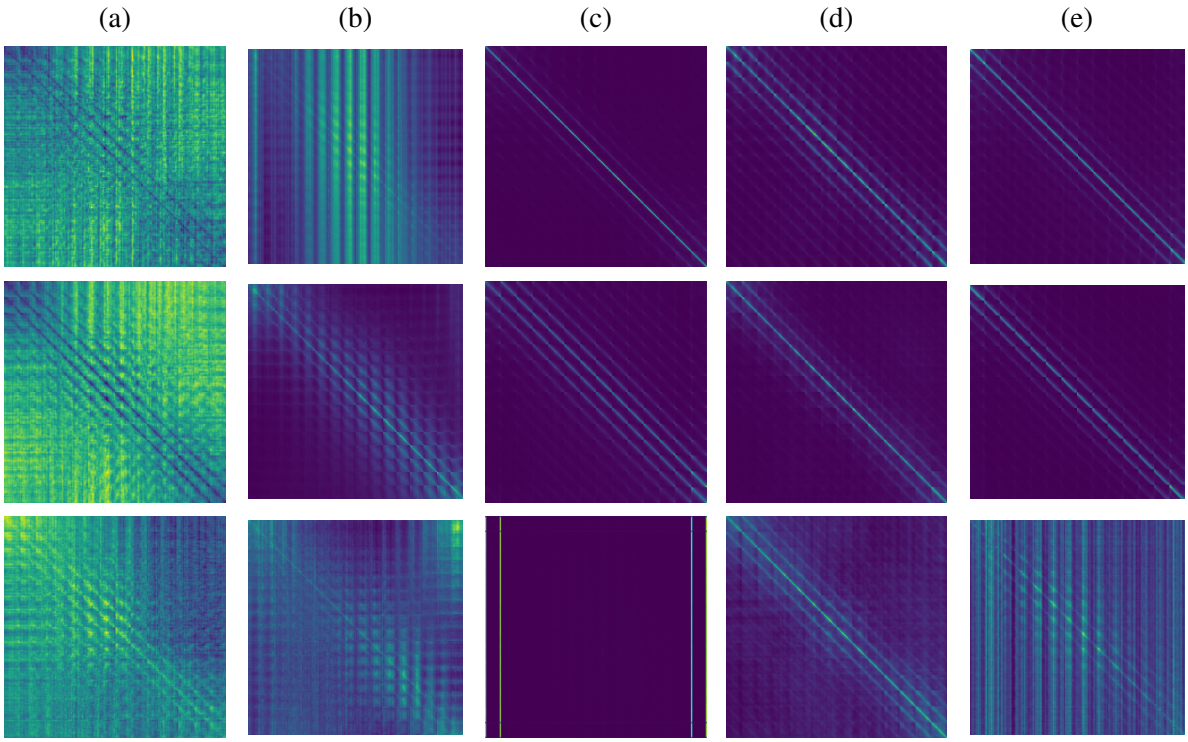


Figure 3.18: Attention maps computed from one CIFAR-10 batch for ViT-Tiny (a) untrained (b) CIFAR-10 trained (c) ImageNet pretrained (d) using our init (e) our init and then CIFAR-10 trained.
Rows: ↓ Layers #1, 4, 11

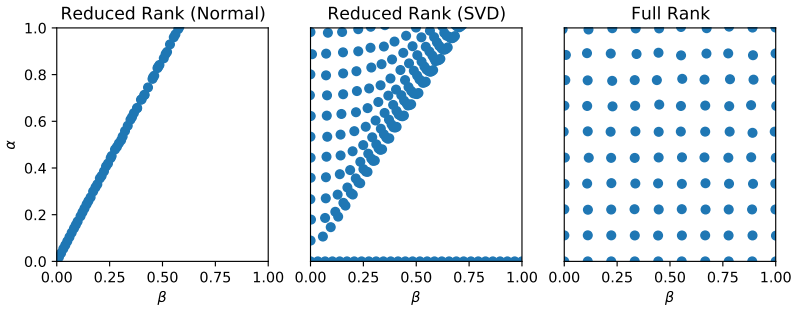


Figure 3.19: Possible α, β for different weight constructions.

where $Z_i \sim \mathcal{N}(0, \frac{1}{d}I)$ and $\alpha_i, \beta_i \in [0, 1]$. That is, we explicitly control the tradeoff between the noise Z_i and the diagonal I by choosing the parameters α_i, β_i. In order to recover the factors W_V, W_{proj}, we use the singular value decomposition:

$$\alpha_1 Z_1 + \beta_1 I = U_1 \Sigma_1 V_1^T \tag{3.14}$$

$$W_V \coloneqq U_1 \Sigma_1, W_{proj} \coloneqq V_1 \Sigma_1^{1/2}, \tag{3.15}$$

and for the low-rank factors W_Q, W_K^T, the reduced SVD:

$$\alpha_2 Z_2 + \beta_2 I = U_2 \Sigma_2 V_2^T \tag{3.16}$$

$$W_Q \coloneqq U_2[:, : k] \Sigma_2[: k, : k]^{1/2} \tag{3.17}$$

$$W_K \coloneqq V_2[:, : k] \Sigma_2[: k, : k]^{1/2}. \tag{3.18}$$

Note that we resample Z_2 for each head.

In Fig. 3.19, we show the different α, β that can be achieved through the two methods proposed above. Using equal random normal matrices, there is a linear relationship between α and β, for both low-rank and full-rank matrices. Using the SVD technique, we achieve a wider variety of selections even in the low-rank case. Consequently, we use this in all experiments.

Attention map structure In practice, our initialization results in attention maps that with a strong diagonal component which reflect the structure of the position embeddings, which we denote by $P \in \mathbb{R}^{n \times d}$. We show this visually in Fig. 3.18, though it is also possible to (roughly) compute their expected value.

Assuming that $X \in \mathbb{R}^{n \times d}$ and $X \sim \mathcal{N}(0, I)$ (which is a reasonable assumption due to the use of LayerNorm), and assuming W_Q, W_K are full-rank and $W_Q W_K^T = \alpha Z + \beta I$ due to our initilization, we can show $\mathbb{E}[(X+P)(\alpha Z + \beta I)(X+P)^T] = \beta d I + \beta P P^T$, as the only products with non-zero mean are $X X^T \approx I$ (on the diagonal) and $P P^T$. Thus, roughly speaking, our initialization results in expected attention maps of the form

$$\mathsf{Softmax}\left(\tfrac{1}{\sqrt{k}}(\beta_1 d I + \beta_1 P P^T)\right). \tag{3.19}$$

That is, our initialization may bias attention maps towards mixing nearby tokens according to the structure of $P P^T$, which can be seen in Fig. 3.18.

Table 3.3: 100 epoch CIFAR-10 classification (ViT-Tiny).

Width	Depth	Heads	Acc. (Base)	Acc. (Init)	Δ Acc.
96	6	3	84.75	87.90	3.15
96	12	3	84.75	88.84	4.09
192	6	3	85.85	89.68	4.63
192	12	1	85.25	89.88	4.63
192	12	3	86.07	90.78	4.71
192	12	6	86.74	91.38	4.64
192	24	3	86.36	91.85	5.49
384	12	3	86.26	91.56	5.30
384	12	6	84.40	92.17	7.77
384	12	12	86.39	92.30	5.91

3.2.4 Accelerating ViT training with mimetic init

CIFAR-10

Training vanilla ViTs from scratch on CIFAR-10 is notoriously difficult, requiring semi-supervised pretraining techniques, additional inductive bias, or heavy data augmentation with long training times Liu et al. (2021b); Lee et al. (2021); Gani et al. (2022); Hassani et al. (2021). In this section, we demonstrate the substantial benefits of using our initialization for vanilla ViTs on from-scratch CIFAR-10 training.

Setup We train all ViTs using a simple pipeline: we use RandAugment and Cutout for augmentation, a batch size of 512, AdamW with 3×10^{-3} learning rate, 0.01 weight decay, and 100 epochs. We use a vanilla ViT with embedding dimension 192, depth 12, patch size 2, and input size 32 unless otherwise noted (ViT-Tiny). We use a class token and sinusoidal position embeddings. We use $\alpha_1 = \beta_1 = 0.7$ and $\alpha_2 = \beta_2 = 0.4$ for all experiments.

Basic results In Table 3.6, we show our main results for CIFAR-10. Across a variety of ViT design parameters, our initialization results in substantial accuracy gains between 2.5-6%. While the benefit of our initialization is quite significant in all cases, we note that it seems to have the most benefit for larger models. For example, we see an improvement of over 6% for a ViT with dimension (width) 384, depth 12, and 6 heads (a ViT-Small), while we see a smaller 4.8% gain for a model with dimension 192 and 3 heads, and a 4.1% gain for dimension 96.

Ablations In Table 3.4, we show some ablations of our initialization technique. If we use the default normal initialization for W_Q, W_K, we see a substantial loss of accuracy of nearly 2%; similarly, if we use default initialization for W_V, W_{proj}, we see an even greater hit to accuracy of around 3.5%. Using neither (just sinusoidal position embeddings), we lose almost 4% accuracy.

Table 3.4: Ablations on CIFAR-10, ViT-Ti

Ablation	Acc.
Our initialization	91.38
Random pos. embeddings	88.70
No init (only sinusoidal pos. embeddings)	87.39
Init only W_Q, W_K	89.17
Init only W_V, W_{proj}	87.23
$W_V W_{proj} \propto -cI \implies W_V W_{proj} \propto +cI$	89.65
GPSA (8 heads)	90.03
GPSA (4 heads)	90.83
+ $W_V W_{proj} \propto -cI$	91.21
Pretrained W_K, W_Q, W_V, W_{proj} & pos. embed	91.15

Further, setting the diagonal of W_V, W_{proj} to be negative rather than positive is in fact quite important, accounting for around 1.5% accuracy. These results suggest that all of the components of our initialization work together, and all are very important. We note that in Fig. 3.17 the prominence of the diagonal tends to fade with depth; we saw no improvement from mimicking this.

GPSA comparison GPSA (Gated Positional Self-Attention) was proposed for use in the ConViT model by Cordonnier et al. (2019). This self-attention variation has two attention maps, one of which is initialized with "soft" convolutional inductive biases to emulate convolution. The effect of each attention map is determined by a learnable gating parameter.

While our goal was to improve Transformers without architectural modifications, this technique is the most similar to our own. (Though it requires, *e.g.,* a particular number of heads and a new, custom layer.) We replaced all self-attention layers with GPSA layers. With 4 heads (approximately 2x2 convolution), accuracy comes fairly close to our own by around 0.6%. Interestingly, adding our $W_V W_{proj}$ initialization to GPSA further narrows the gap by around 0.4%. This shows that our technique may even be useful for self-attention variants. More importantly, it shows that our technique is competitive even with those requiring more extensive architectural changes or explicitly-constructed convolutional biases.

Pretrained weights Our initialization technique only considers position embeddings and the query, key, value, and projection weights. Consequently, we consider transfering just these weights from an ImageNet-pretrained ViT as a baseline initialization technique. This achieves 91.15% accuracy, which is marginally lower than our own initialization. While this does not say anything about the initialization of the patch embedding and MLP layers, this may provide some evidence that our self-attention initialization is close to optimal.

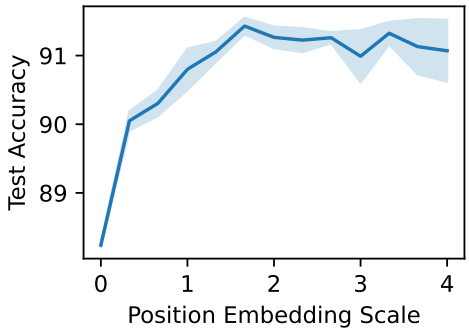


Figure 3.20: Increasing the scale of the position embeddings improves CIFAR-10 performance (ViT-Tiny).

Table 3.5: ImageNet Results

Arch.	Patch Size	Batch Size	Input Size	Acc. (Base)	Acc. (Init)	Δ Acc.
↓ ResNet-style Training Pipeline (150 epochs) ↓						
Vit/Ti	16	640	224	70.28	**73.08**	2.8
Vit/Ti	16	1024	224	67.80	**71.92**	4.1
↓ DeiT-style Training Pipeline (300 epochs) ↓						
Vit/Ti	16	1024	224	72.08	**72.65**	0.57
Vit/S	16	1024	224	79.83	**80.36**	0.53

Position embeddings According to Table 3.4, the use of sinusoidal position embeddings instead of randomly-initialized ones is crucial for our initialization. Using random rather than sinusoidal position embeddings with our initialization is disastrous, resulting in a decrease of 3% in accuracy. However, *only* initializing the position embeddings is not helpful either; ablating the rest of the init gives a similar performance decrease. In other words, it is the *interaction* of our initialization with the position embeddings which is useful. Consequently, with Eq. 3.19 in mind, we investigated the scale of the position embeddings, which changes their importance relative to the inputs themselves.

Position embedding scale Adding a new hyperparameter, we multiplied the embeddings by a factor γ, and tried several choices as shown in Fig. 3.20. Increasing the scale from 1 to ≈ 2 substantially improves performance, by around 0.5%.

Internal resolution ViTs are typically trained using high-resolution inputs and large patch sizes. In contrast, we trained most of our models on CIFAR-10 using small 32×32 inputs and 2×2 patches. Consequently, we investigate how the choice of patch and input size affects

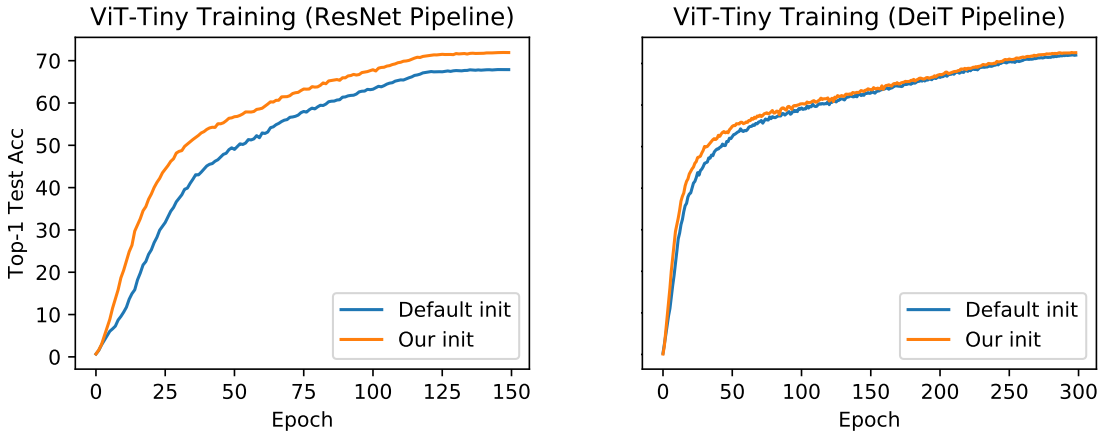


Figure 3.21: Training curves for DeiT-Tiny in a (a) ResNet-style training pipeline and a (b) DeiT-style pipeline. In the ResNet pipeline, we see a 4.1% improvement, compared to a 0.5% improvement in the DeiT pipeline.

performance. In Appendix **??**, Table **??**, we can see that our initialization is quite beneficial for many such combinations.

Data efficiency We hypothesize that our initialization leads ViTs to have an inductive bias more suitable for images, and thus would expect the initialiation to be associated with especially high performance gains on small datasets. Consequently, we trained on a variety of subsets of CIFAR-10 (see Fig. 3.22). Surprisingly, we did not see performance gains inversely proportional to the size of the dataset. More research, *e.g.,* on larger datasets, would be necessary to understand how our initialization changes the data requirements of ViTs.

Other Transformer initializations While the motivation of our initialization is substantially different from that of other Transformer initialization techniques, we provide some comparisons in Table 3.7. T-Fixup (Huang et al., 2020) and ZerO (Zhao et al., 2021) focus on initializing the whole network rather than just the self-attention layers. For ZerO initialization, we only apply the initialization to self-attention layers. For T-Fixup, we apply the initialization to both self-attention and MLPs. Nonetheless, T-Fixup harms performance relative to the baseline, and ZerO offers only a small improvement.

Tuning hyperparameters It is infeasible for us to search over all combinations of α_i and β_i, so we first fixed α_1 and β_1 according to a guess of (0.6, 0.3), and then tuned α_2 and β_2. From this, we chose α_2, β_2. Then, holding this fixed, we tuned α_1, β_1. Our grid search was performed for 100-epoch CIFAR-10 training on a ViT-Tiny.

ImageNet

Here, we show that our initialization benefits training ViTs from scratch on another relatively "small" dataset (for Transformers): ImageNet-1k. We test two settings: a ResNet-style Wight-

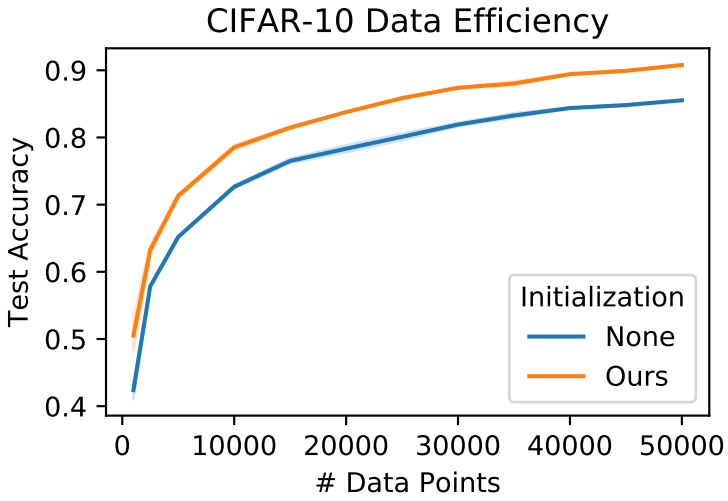


Figure 3.22: Adjusting the number of training points on CIFAR-10.

man et al. (2021) training pipeline with 150 epochs and standard cross-entropy loss (*i.e.,* the technique of Trockman and Kolter (2022)), and the 300-epoch DeiT training pipeline from Touvron et al. (2021b). In both cases, we see significant improvements for using our initialization, with gains between 2.8-4.1% for a ViT-Tiny in the ResNet-style pipeline and around 0.5% in the DeiT pipeline. We find it surprising that we see relatively high gains even for very-long training times. Notably, we used the same hyperparameters as found for the CIFAR-10 experiments, though with a position embedding scale of 1.

The large performance in the ResNet-style training pipeline is particularly notable. One of the main contributions of Touvron et al. (2021b) was to propose a particular training pipeline which was effective for training ViTs on ImageNet-scale datasets, as ViTs did not work well in ResNet-style training pipelines. However, our initialization provides a major boost in accuracy for ViT-Tiny in this setting, suggesting that it begins to bridge the gap between ViT and ResNet training.

In Fig. 3.21, we show training progress for both ViT training pipelines; the difference is smaller for the DeiT pipeline, which has a larger batch size and more epochs.

Other Datasets

To further show that our initialization is not overfit to CIFAR-10 or ImageNet in particular, we present results for CIFAR-100, SVHN, and Tiny ImageNet using our initialization. We use the same settings as before with a ViT-Tiny, though with 4×4 patches for TinyImageNet. In Table 3.6, we see that our initialization leads to improvements in test accuracy over 5% for Tiny ImageNet and CIFAR-100, but only 0.39% for the perhaps-easier SVHN dataset.

Table 3.6: Our initialization on other datasets (ViT-Tiny, 100 epochs).

Dataset	Acc. (Base)	Acc. (Init)	Δ Acc.
Tiny ImageNet	45.24	50.87	5.63
CIFAR-100	60.94	67.33	6.39
SVHN	96.40	96.79	0.39

3.2.5 Why does this initialization work?

We have shown that our mimetic initialization is quite effective for enhancing visual recognition on small datasets. Here, we propose some additional explanations for why our method is effective. The first section concerns the query and key weights, while the next two investigate the somewhat-more-mysterious negative diagonal of the value and projection product.

Near-identity attention maps. In Fig. 3.18 and Eq. 3.19, we see that our initialization, much like pretraining, makes the attention maps somewhat similar to identity matrices, particularly in earlier layers. The resemblance of our attention maps using our initialization to those in pretrained models is notable in itself. He et al. (2023) notes that forcing attention maps to be the identity avoids rank collapse, which can otherwise prevent trainability. However, they note that exact-identity attention cannot pass gradients to the query and key parameters, meaning it is not actually a viable initialization technique. We hypothesize that our initialization strikes a balance between untrained attention maps (as in Fig. 3.18a) and identity attention maps.

LayerScale analogy In Touvron et al. (2021c), a simple technique called LayerScale is proposed to train deeper Transformers more effectively, in which the layer at a skip connection is multiplied by a learnable diagonal scaling matrix D:

$$X_l' = X_l + D \cdot \mathsf{SelfAttn}(\eta(X_l)) \tag{3.20}$$

where η denotes LayerNorm. Here, we show that the way we initialize W_V, W_{proj} has a relatively weak resemblance to this technique. Considering Eq. 3.19, we approximate the attention maps after our initialization as being close to the identity, and assume that $\eta(X_l) \approx X_l$:

$$\begin{aligned} X_l' &= X_l + \mathsf{SelfAttn}(\eta(X_l)) && (3.21)\\ &\approx X_l + I\eta(X_l)W_V W_{proj} && (3.22)\\ &\approx X_l + I\eta(X_l)(\alpha Z - \beta I) \quad \text{(due to our init)} && (3.23)\\ &\approx (I-\beta)X_l + \alpha\eta(X_l)Z && (3.24) \end{aligned}$$

Scaling X_l by $(I-\beta)$ is similar in spirit to LayerScale, except in our case we are multiplying the left-hand instead of the right-hand term in the skip connection. This motivates us to compare our technique for setting to $W_V W_{proj}$ to using LayerScale, or our variant of LayerScale above.

Table 3.7: Other initializations

T-Fixup	ZerO	LayerScale Original	LayerScale Our Version	Our Initialization
85.38	87.41	89.90	88.68	91.38

We searched ten choices of initialization for the diagonal elements in $[0, 1]$ for both LayerScale techniques, replacing our $W_V W_{proj}$ initialization, and report the best results in Table 3.7. Note we leave our $W_Q W_K^T$ initialization unchanged. Neither method achieves the performance of ours (with a difference of about 1.5%) though LayerScale comes closest. We conclude that the benefits of our initialization extend beyond its possible similarity to LayerScale.

Convolution analogy. Many works which successfuly train ViTs on small datasets do so by adding aspects of convolution, whether implicitly or explicitly. Here, we explore adding locality to self-attention through convolutional biases:

$$\mathsf{Softmax}\left(XW_Q W_K^T X^T + \gamma C\right), \tag{3.25}$$

where C is a doubly-block circulant convolution matrix and γ is a learnable scalar. Here, C is reminiscent of the PP^T term in Eq. 3.19. This achieves 87.5% accuracy on CIFAR-10 within our usual training pipeline (without our init). For comparison, plain self-attention with no special initialization achieves 88.1% accuracy. Next, we move the convolution outside the softmax:

$$\mathsf{Softmax}\left(XW_Q W_K^T X^T\right) + \gamma C, \tag{3.26}$$

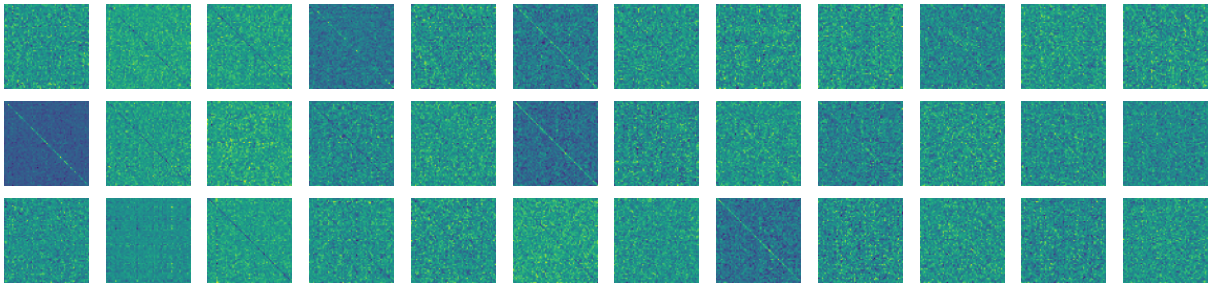
This has a more considerable advantange, resulting in 89.9% accuracy. Then, if we instead use $C' = \mathsf{Softmax}(\gamma C)$ to restrict C' to be all-positive, we achieve 75% accuracy. That is, it appears that the negative component of the convolution matrix is necessary.

Thus, we hypothesize that initializing $W_V W_{proj}$ to have a negative diagonal is perhaps beneficial for the same reason: this allows for some degree of "negative" or edge-detector-like spatial mixing to occur, a potentially useful starting point for the purpose of visual recognition.

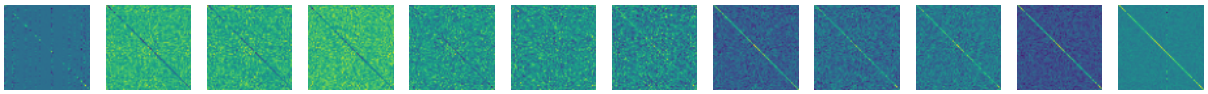
3.2.6 Language modeling explorations

While our method was primarily inspired by pretrained Vision Transformers, in this section we investigate its potential for use in language models. As noted in Sec. 3.2.2 and seen in Fig. 3.23, we do not see precisely the same pattern in a pre-trained GPT-2 model as we do in a ViT. Nonetheless, we use the same technique here without modification; we saw no improvement from, *e.g.,* attempting to model the positive diagonals of $W_V W_{proj}$.

Small-scale Generally, it is hard to train Transformers from scratch on small language tasks Dai et al. (2019); it requires substantial regularization, *e.g.,* in the form of dropout. For word-level



(a) $W_Q W_K^T$ has a wider array of diagonal magnitudes (first 3 heads shown). $\rightarrow$ Layers 1-12, $\downarrow$ Attention Heads 1-3 of 12



(b) $W_V W_{proj}$ becomes positive in deeper layers.

Figure 3.23: A pretrained GPT-2 shows considerably different patterns in the products of $W_Q W_K^T$ and $W_V W_{proj}$, compared to ViTs.

modeling on Penn TreeBank (PTB), we thus add one regularization tweak: word-level embedding dropout (*i.e.,* dropout of entire embedding vectors). This allows us to achieve sub-100 perplexity.

We use a training setup identical to that of Bai et al. (2018), training for 100 epochs and reducing the learning rate when it plateaus. We used a vanilla Transformer with sinusoidal position embeddings, with embedding dimension 384, 12 layers, 8 attention heads, and weight-tied embeddings.

First, on char-level PTB we did a small-scale hyperparameter search for those α_i, β_i yielding the best validation BPC. We chose $\alpha_1 = 0$, $\beta_1 = 0.5$, and $\alpha_2 = \beta_2 = 0.2$. We used these parameters on subsequent word-level modeling tasks. On char-level PTB, we see a small but significant reduction in BPC from 1.233 to 1.210 through using our initialization. Similarly, we see a small reduction in perplexity on word-level PTB, from 84.84 to 82.34. (For both tasks, smaller is better.)

While our initialization does not make a large amount of difference for these small-scale language tasks as it does for vision tasks, it does show a small amount of improvement. We suspect that it may be the case that a mimetic initialization scheme more finely-tuned to the language setting may show still better performance.

Medium-scale Next, we tried our initialization on a larger-scale task, WikiText-103. Here, we used an embedding dimension of 410 with 16 layers, 10 heads, and sinusoidal embeddings, with the same hyperparameters as for the previous task. As this dataset is around 110 times larger than PTB, we trained for only 50 epochs. Here, we see a more significant performance gain from using our initialization, reducing the test perplexity from 28.87 to 28.21 (see Table 3.8). While this is not a massive improvement, this is consistent with our observation on vision tasks that the improvement from our technique may be more significant for larger models. Further, we also

Table 3.8: Language results

Task	Metric	Base	Init
Char-level PTB	bpc	1.233	**1.210**
Word-level PTB	ppl	84.84	**82.34**
WikiText-103	ppl	28.87	**28.21**

note that in this case the number of parameters being initialized is quite small relative to the total number of parameters of the language model due to the word embedding weights, something which does not occur with vision models.

3.2.7 Summary of contributions

Our proposed initialization technique for Transformers is particularly effective at improving performance on small-scale image recognition tasks, leading to an increase of over 5% accuracy in some cases. In other words, we address the problem that Vision Transformers are hard to train in ResNet-style pipelines solely through a structured initialization of the weights, without need for any kind of pretraining or architectural modifications. To a lesser extent, we demonstrated that our initialization leads to non-trivial gains on WikiText-103, showing that it also has potential to similarly improve language modeling on relatively small datasets. More broadly, we proposed a class of techniques we call *mimetic initialization*, in which we attempt to gain some benefits of pretraining by mimicking the surface-level qualities of pretrained models. We speculate that it may be possible to use domain knowledge to "program" models before training in order to reach more desirable optima that may have been out of reach with a completely random initialization. With better structured initialization techniques like our own, perhaps Transformers really *are* the universal architecture.

3.3 Mimetic Initialization for State Space Models

State Space Models (SSMs) show promise as a potential replacement for Transformers (Vaswani, 2017) with substantially lower inference costs (Gu and Dao, 2023; Dao and Gu, 2024). While Transformer memory grows linearly with the input sequence length, SSMs use only a constant amount, compressing all the context into a fixed-size state. SSMs perform comparably to Transformers on a variety of common benchmarks. However, recent research has highlighted a set of tasks on which SSMs perform substantially worse than Transformers (Waleffe et al., 2024), particularly those involving copying or recall (Jelassi et al., 2024; Arora et al., 2024). This is perhaps unsurprising, as it is harder to recall from a compressed, fixed-size representation, particularly as its length grows.

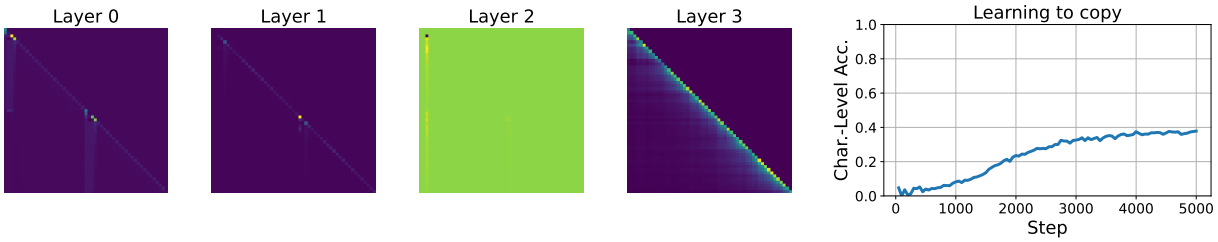
Nevertheless, SSMs use relatively large state sizes in practice, and we wonder if their poor performance on tasks such as copying could be due to training difficulties rather than fundamental capacity constraints. We present a qualitative study of the failure modes of SSMs on

the copying task. In particular, we inspect the time-dependent linear transformation matrix of Mamba layers, which is analogous to the attention map of self-attention layers. We compare these layers to their counterparts in self-attention/Mamba hybrid architectures that successfully learn to copy, and based on these comparisons, we propose a structured initialization technique that allows Mamba layers to more readily mimic self-attention. Our technique makes use of the fact that state space layers can be seen as a form of linear attention with a learnable, structured causal mask. We find evidence that such linear-attention-like Mamba layers arise naturally after large-scale pretraining, suggesting that this pattern may be fundamental to the recall abilities of SSMs.

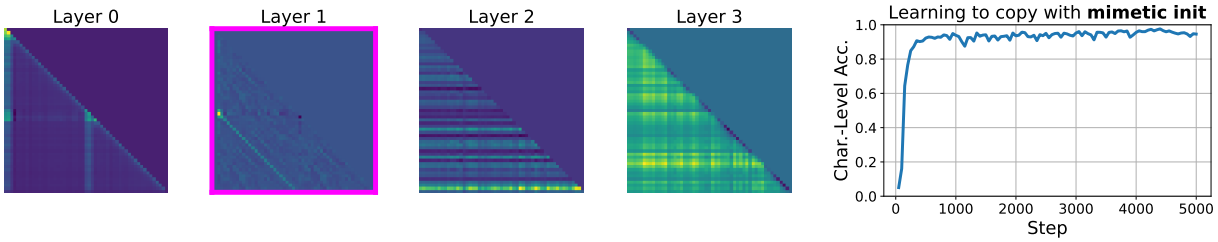
The proposed mimetic initialization allows Mamba to quickly learn to copy and do associative recall on up to $4\times$ longer strings, and we show for the first time that **SSMs can achieve $2\times$ length generalization** or more. Mimetic initialization is **essentially compute-free**, but we show **it is comparable to pretraining** in allowing Mamba to learn to copy and recall. Our work helps to better understand the capacity of SSMs relative to Transformers in practice and can assist in further studies of their capabilities, which may have been underestimated by previous research.

Related work Recently, Jelassi et al. (2024) did a thorough investigation of the ability of state space models (in particular Mamba 1) to copy in comparison to Transformers. Their theoretical results demonstrate that SSMs with a fixed state size have fundamentally limited copying capacity, unlike Transformers which can strongly generalize. Empirically, they find that Transformers (especially with their proposed custom position embeddings) vastly outperform SSMs on copying, both in terms of learning and length generalization. They note that in practice, SSMs may be better at copying than expected due to their relatively large state sizes, but do not observe very good copying performance in their experiments. Similarly, Arora et al. (2024) note that SSMs struggle on recall tasks due to their limited state size. They propose an effective intervention in the form of interleaved kernelized linear attention layers that boost recall performance. The second, improved version of the Mamba architecture improves upon associative recall ability, although the authors note that this task remains difficult for SSMs (Dao and Gu, 2024).

Initialization has been important for SSMs since their introduction to deep sequence modeling by Gu et al. (2021); a structured initialization of the state matrix was crucial to the performance of these earlier time-invariant SSMs (Gu et al., 2020; Gupta et al., 2022; Gu et al., 2022; Smith et al., 2023). Our work further demonstrates the importance of initialization for SSMs, taking inspiration from *mimetic initialization* (Trockman and Kolter, 2023; Trockman et al., 2022), which uses pretrained models as case studies of good initialization. For example, previous work noted that self-attention layers in pretrained Vision Transformers may try to imitate the local mixing ability of convolutions, which is reflected in the correlations between query/key and value/projection weights; initializing weights with statistical structure that mimics this pattern greatly improved trainability. We follow a similar methodology to propose a novel mimetic initialization technique for state space layers based on our observations that (1) these layers can represent linear attention, which can improve recall and (2) they sometimes approximate linear attention in pretrained models.



(a) Training a Mamba with default initialization to copy.



(b) Mamba with mimetic initialization learns to use its attention-like abilities.

Figure 3.24: Mambas initialized with our technique learn to copy more effectively than those with default initialization. We see evidence of copying ability in the Mamba attention maps; see Layer 1.

3.3.1 State space model background

Recently, state space models have become popular as a choice of token mixing layer, i.e., as a replacement for self-attention. We refer to layers that use state space models for this purpose as *state space layers*. As it is common in the literature, with a slight abuse of definitions, we refer to architectures like Mamba 1 & 2 that use state space layers only for sequence mixing as *state space models*.

State space models For a scalar sequence $x \in \mathbb{R}^{\mathtt{T}}$, SSMs are linear recurrences of the form

$$h_{t+1} = \bar{A}h_t + \bar{B}x_t, \quad y_t = Ch_t, \tag{3.27}$$

where $h_t \in \mathbb{R}^{\mathtt{N}}$ is a hidden state, and $\bar{A} \in \mathbb{R}^{\mathtt{N}\times\mathtt{N}}$, $\bar{B} \in \mathbb{R}^{\mathtt{N}\times 1}$, $C \in \mathbb{R}^{1\times\mathtt{N}}$ are the state space model parameters. Traditionally, SSMs are continuous systems, and the bar notation refers to the *discretized* form of parameters A and B, which depend on the step size Δ that is used to sample an implicit underlying continuous signal $x_t = x(\Delta t)$. Typically, some structure is imposed on $A \in \mathbb{R}^{\mathtt{N}\times\mathtt{N}}$, such as diagonal-plus-low-rank (S4), diagonal (Mamba), or scalar-times-identity (Mamba 2).

In contrast, *selective* SSMs such as the S6 layer in Mamba allow the parameters $\bar{A}_t, \bar{B}_t, C_t$ to vary with time, i.e., depend on x_t. The particular state space layer in Mamba operates on sequences of $\mathtt{D}$-dimensional tokens $X \in \mathbb{R}^{\mathtt{D}\times\mathtt{T}}$. Indexing tokens with t and *channels* with d, it computes

$$h_{(t+1),d} = \bar{A}_{td}h_{td} + \bar{B}_{td}X_{td}, \quad y_{td} = C_t h_{td}, \tag{3.28}$$

where $\bar{A}_{td}, \bar{B}_{td}, C_t$ depend on *all* channels of input x_t, but with different discretization parameters Δ_{td}, hence the dependence of $\bar{A}_{td}$ and $\bar{B}_{td}$ on d. Define the underlying parameters

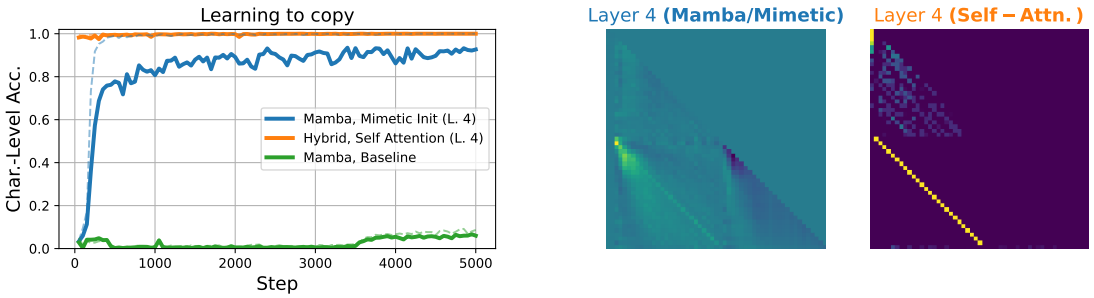


Figure 3.25: A hybrid Mamba architecture with one Self-Attention layer easily learns to copy. Dotted lines: performance on training length (50), solid: $2\times$ length generalization (100).

$W_B, W_C \in \mathbb{R}^{\mathtt{N}\times\mathtt{D}}$, and $A \in \mathbb{R}^{\mathtt{D}\times\mathtt{N}}$. Let $W_\Delta \in \mathbb{R}^{\mathtt{D}\times\mathtt{D}}$ be a rank-r matrix, and bias $b_\Delta \in \mathbb{R}^{\mathtt{D}}$. Then the continuous state space model parameters are computed as $B_t = W_B^T X_{:,t}$ and $C_t = W_C^T X_{:,t}$. The parameters of the discretized state space models are then computed as follows:

$$\Delta_{t,d} = \mathsf{softplus}(W_{\Delta d}^T X_{:,t} + b_{\Delta,d}), \quad \bar{A}_{td} = \exp(A_d \Delta_{t,d}), \quad \bar{B}_{td} = B_t \Delta_{t,d}. \tag{3.29}$$

Please refer to Dao and Gu (2024) for a more detailed discussion on selective SSMs.

Matrix form of SSMs The operations of Eq. 3.29 can be written concisely in matrix form:

$$\Delta \coloneqq \mathsf{softplus}\left(W_\Delta X + b_\Delta\right) \qquad \in \mathbb{R}^{\mathtt{D}\times\mathtt{T}} \tag{3.30}$$

$$\bar{B}_d \coloneqq W_B X \odot \mathbf{1}_n \Delta_d \qquad \in \mathbb{R}^{\mathtt{N}\times\mathtt{T}} \tag{3.31}$$

$$C \coloneqq W_C X \qquad \in \mathbb{R}^{\mathtt{N}\times\mathtt{T}} \tag{3.32}$$

$$\bar{A}_d \coloneqq \exp\left(A_d^T \Delta_d\right) \qquad \in \mathbb{R}^{\mathtt{N}\times\mathtt{T}} \tag{3.33}$$

As noted first by Ali et al. (2024), the time-varying discrete recurrence $h_{t+1} = \bar{A}_t h_t + \bar{B}_t x_t,\ y_t = Ch_t$ can be unrolled and viewed as a matrix operation. Namely, channel d of the output of an SSM layer, denoted with $Y_d \in \mathbb{R}^{\mathtt{T}}$, can be written as $Y_d \coloneqq M_d X$, where $M_d \in \mathbb{R}^{\mathtt{T}\times\mathtt{T}}$ is a matrix transformation dependent on d. Each matrix M_d represents a time-dependent linear transformation, much like attention maps in self-attention. For $i, j \in [\mathtt{T}]$, the M_d matrix of the Mamba state space layer for channel d can be expressed as follows, where $\mathbf{1}\{i \le j\}$ does causal masking:

$$M_{d,i,j} = C_{:,i}^T \left(\Pi_{k=j+1}^{i} \mathsf{diag}(\bar{A}_{d,:,k})\right) \bar{B}_{d,:,j} \times \mathbf{1}\{i \le j\}. \tag{3.34}$$

Eq. 3.34 can be viewed as a linear attention matrix computed from $\bar{B}$ and C with a learnable causal mask parameterized by $\bar{A}$ (Dao and Gu, 2024). As it will be useful later, we note that in practice, A is parameterized as $A \coloneqq -\exp(A_{\mathsf{log}})$ with $A_{\mathsf{log}} \in \mathbb{R}^{\mathtt{D}\times\mathtt{N}}$. The selective state space layer of Mamba 2 is broadly similar to that of Mamba 1; it follows equations 3.30–3.33, but instead of having $\mathtt{D}$ different A_d and Δ_d, it has $\mathtt{H}$ independent A and Δ, each of which are repeated $\mathtt{D}/\mathtt{H}$ times to construct A_d and Δ_d. Each of these $\mathtt{H}$ independent A are parameterized as scalar-times-identity matrices, resulting in just H parameters. These $\mathtt{H}$ components correspond to "heads", leading to only $\mathtt{H}$ unique $\bar{A}_d$ and $\bar{B}_d$ parameters, and only $\mathtt{H}$ "attention matrices" M_d (*c.f.* Eq. 3.34), as in multi-head attention.

Mamba architecture Mamba 1 and 2 are prominent sequence modeling architectures that combine selective state space layers (as the sequence mixer) with more standard layers. We describe below the Mamba 1 block, and refer the reader to (Dao and Gu, 2024) for details on Mamba 2, which are not essential to our work. Omitting the final LayerNorm, the Mamba block is a composition of two sequence mixer layers (1D convolution and a selective SSM layer) a gated linear block:

$$W_3\{\texttt{SSM}\left[\sigma(\mathsf{DepthwiseConv1d}(W_1X))\right] \odot \sigma(W_2X)\} + X, \tag{3.35}$$

where σ is SiLU (Elfwing et al., 2018). Mamba 2 simplifies this block, merging all projections into W_1. For both, the convolution layer before the SSM will be considered in our initialization.

Mamba attention maps Throughout this work, we visually inspect M_d to better understand the operation implemented by Mamba layers. However, it is infeasible to look at all $\texttt{D}$ maps, and we instead visualize and report the average over channels $\frac{1}{\texttt{D}}\sum_{d=1}^{\texttt{D}} M_d$, which we hereafter refer to as the *attention map* of a Mamba layer. In practice, the inter-channel variation in maps is relatively small, as the behavior of M_d is dominated by $\bar{B}_d$ and C. We also sometimes find it useful to inspect the *average attention mask* $\frac{1}{\texttt{DN}}\sum_{d=1}^{\texttt{D}}\sum_{n=1}^{\texttt{N}}(\Pi_{k=j+1}^{i}\mathsf{diag}(\bar{A}_{d,:,k}))_{n,n}$ to approximately determine the effective receptive field of the Mamba layer (i.e., how far into the past it can look).

Copying task Most of our experiments focus on copying, a simple task where SSMs are known to fall far behind Transformers. We train the model to predict the paste string given the copy string, emitting a stop token at completion.

$$\underbrace{\texttt{abcdefghijk}}_{\text{copy string}} \quad | \quad \underbrace{\texttt{abcde}\,\underline{?}}_{\text{paste string}} \cdots \square \tag{3.36}$$

Since Transformers cache the whole sequence, it is easy for them to learn the task and to generalize far beyond the training length. However, since SSMs compress tokens into a fixed-size state, it is hard for them to store and decode back long sequences. We consider copying sequences of varying length and of different vocabulary size, drawing tokens uniformly at random. We also investigate *stack-order* copying, where the paste string needs to be generated in the reverse order.

Multi-query associative recall Another synthetic task that has been shown to be an important discriminator between Transformer and SSM abilities is multi-query associative recall, which tests models' ability to store and recall many key-value pairs. Transformers are well-suited for this task, as they can implement induction heads easily (Olsson et al., 2022).

$$\underbrace{\texttt{a1 b2 c3 d4}}_{\text{key}-\text{value pairs}} \quad | \quad \underbrace{\texttt{c3 b}\,\underline{?}}_{\text{queries}} \cdots \square \tag{3.37}$$

Similarly to copying, we investigate length generalization on multi-query associative recall. In our implementation, each key may occur only once, i.e., it cannot be overwritten by later key/value pairs.

3.3.2 Initializing state space layers to be more like attention

To better understand why Mamba often fails to learn to copy, we start by examining a small model trained to copy 50-character strings. In Figure 3.24a, we can see that Mamba plateaus. Visual inspection of its attention maps reveals that it has probably failed to learn an interpretable copying operation.

Attention enables copying To explore what Mamba might be missing to allow it to copy, we trained a hybrid eight-layer Mamba whose fourth layer is single-head self-attention. As shown in Fig. 3.25, this one layer enables perfect copying performance, both on in-distribution length-50 strings (dotted lines) and generalizing to length-100 strings (solid lines). The softmax attention head learns a sharp "look-behind" operation, constructing the paste string by directly attending to the copy string, likely exploiting an implicit position embedding learned by the preceding Mamba layers. We propose two initialization changes that allow state space layers to better use their state capacity.

1. State space layers can be linear attention While there is likely more than one way to learn to copy, we suspected that Mamba's copying ability is tied to its ability to represent a similar operation to the one in this self-attention layer. Notably, in Figure 3.24a, the Mamba layers tend to look only into the recent past, while the self-attention layer in Figure 3.25 can attend all the way to the beginning of the string. While SSMs cannot look arbitrarily far into the past because of their fixed state size, even in the simplest time-invariant SSMs, the amount of history stored in the state is controlled by the parameter A, whose initialization was crucial to the initial success of these models (Gu et al., 2021).

Consequently, we focus on the state matrix A, which controls the "receptive field" of the state space layer. Note in Eq. 3.34 that if $\bar{A}_d \approx 1$, then $M_{d,i,j} \approx C_{:,i}^T \bar{B}_{d,:,j}$. That is, the state space layer's attention map resembles a product of *queries* and *keys*. The only inter-channel variation in this equation is from Δ_d in Eq. 3.31, so that if $\Delta_d \approx 1$ then $\bar{B}_d \approx W_B X$, which results in $M = X^T W_C^T W_B X$, which is simple linear attention before applying the causal mask. Thus, if we set parameters so that $\Delta_d, \bar{A}_d = 1$, the state space transformation is the same for every channel, and it is simple (non-kernelized) linear attention with head dimension $\mathtt{N}$ and no value/projection matrices:

$$\Delta_d, \bar{A}_d \approx 1 \quad \Longrightarrow \quad Y \approx X \cdot \mathsf{CausalMask}\left(X^T W_C^T W_B X\right) \in \mathbb{R}^{\mathtt{D} \times \mathtt{T}}. \tag{3.38}$$

However, both $\bar{A}_d$ and Δ_d are parameterized and input-dependent, so we cannot directly set them to one. We use details of the Mamba implementation: To make $\bar{A}_d = \exp(A_d^T \Delta_d) \approx 1$, we parameterize $A = -\exp(-cA_{\mathsf{log}})$, which is nearly 0 for large c, making $A_d^T \Delta_d \approx 0$ in Eq. 3.33. We choose c from $\{2, 4, 8\}$. We then set $W_\Delta \approx 0$ and $b_\Delta = \mathsf{softplus}^{-1}(1) \approx 0.54$ in Eq. 3.30 so $\Delta_d \approx 1$. **This makes the state space layer close to its linear attention counterpart at initialization.**

2. Correlated tokens should attend to each other Having shown that state space layers can mimic linear attention, we now try to make them mimic attention layers that can copy, such as the

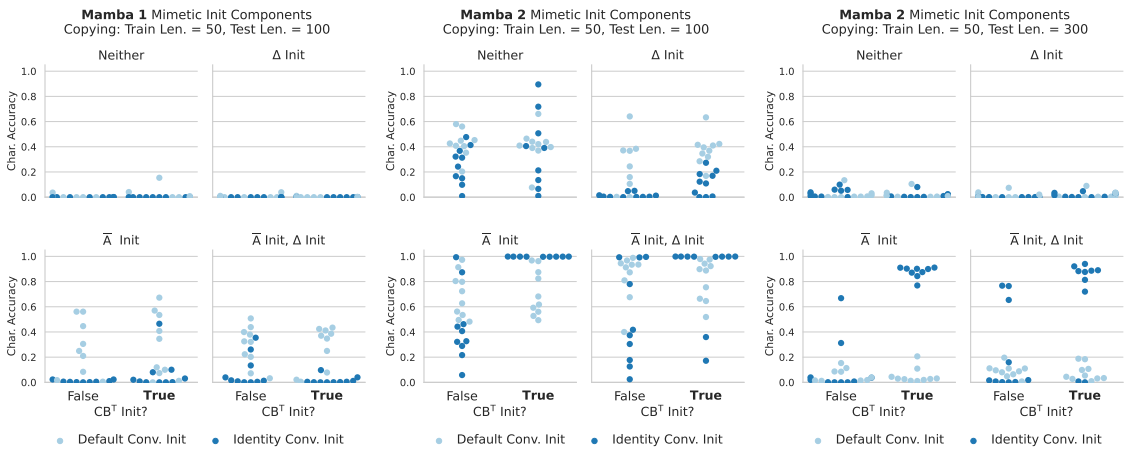


Figure 3.26: Testing the four components of our initialization on Mamba 1 & 2 for 10 seeds.

one in Fig. 3.25, which implements a look-behind operation. We focus on a single linear attention/state space layer, *assuming* the layers before it learned a representation amenable to copying. Consider a copying example of length n, where we have already copied $k < n$ of the $\mathtt{D}$-dim. tokens past the delimiter $x_\parallel$ and want to copy the $(k{+}1)^{\mathsf{st}}$ one: $X = (x_1, \cdots, x_n, x_\parallel, x_1, \cdots, x_k) \in \mathbb{R}^{(n+k+1)\times \mathtt{D}}$. We assume that preceding layers f have learned to superimpose a position embedding as follows:

$$f(X) = (x_1 + p_1, \cdots, x_n + p_n, x_\parallel + p_1, x_1 + p_2, \cdots, x_k + p_{k+1}) = X + P \in \mathbb{R}^{(n+k+1)\times \mathtt{D}},$$

so that token with index k in the paste string will attend to token $k+1$ in the copy string because $(x_{i+1} + p_{i+1})^T (x_i + p_{i+1}) > 0$, assuming $x_{i+1}^T x_i, x_j^T p_j \approx 0$ (uncorrelated) and $p_j^T p_j = 1$ (correlated). That is, $f(X)^T f(X) \approx P^T P$ will have similar structure to that in Fig. 3.25. In this case, copying behavior will arise in our state space/linear attention layer if $P^T W_C^T W_B P \approx P^T P$, i.e., when $W_C^T W_B \approx I$. Since W_C, W_B are low rank ($\mathtt{N} < \mathtt{D}$), their product cannot be exactly the identity; using the fact that random Gaussian matrices are semi-orthogonal, we could set $W_C := W_B$ to get $W_C^T W_B \approx I$. Initializing the queries and keys to be correlated was also noted by Trockman and Kolter (2023), who suggest these weights should not be strictly equal, so we instead set $W_C := \frac{1}{2}(W_C' + W_B)$. In summary, assuming the model has learned a useful correlation structure between tokens, setting $W_C^T W_B \approx I$ ensures this structure can be leveraged by attention. For similar reasons, we experiment with initializing the convolution in Mamba layers to the identity.

Which of these components matter? In Fig. 3.26, we investigate the interaction of these four possible mimetic initialization components, displaying all sixteen possible off/on combinations. We investigate copying on 50-long strings and generalizing to 100- and 300-long strings for a 24-layer Mamba with hidden size 1024 as in Jelassi et al. (2024). For the A and Δ initializations, we fix $c = 8$ and $b_\Delta = 0.54$. For Mamba 1, we see that there is only a significant

Initialization	Purpose
$A \approx 1$ $\Delta \approx 1$	Approximate linear attn
$W_C^T W_B \approx I$ Conv1d $\approx I$	Encourage recall

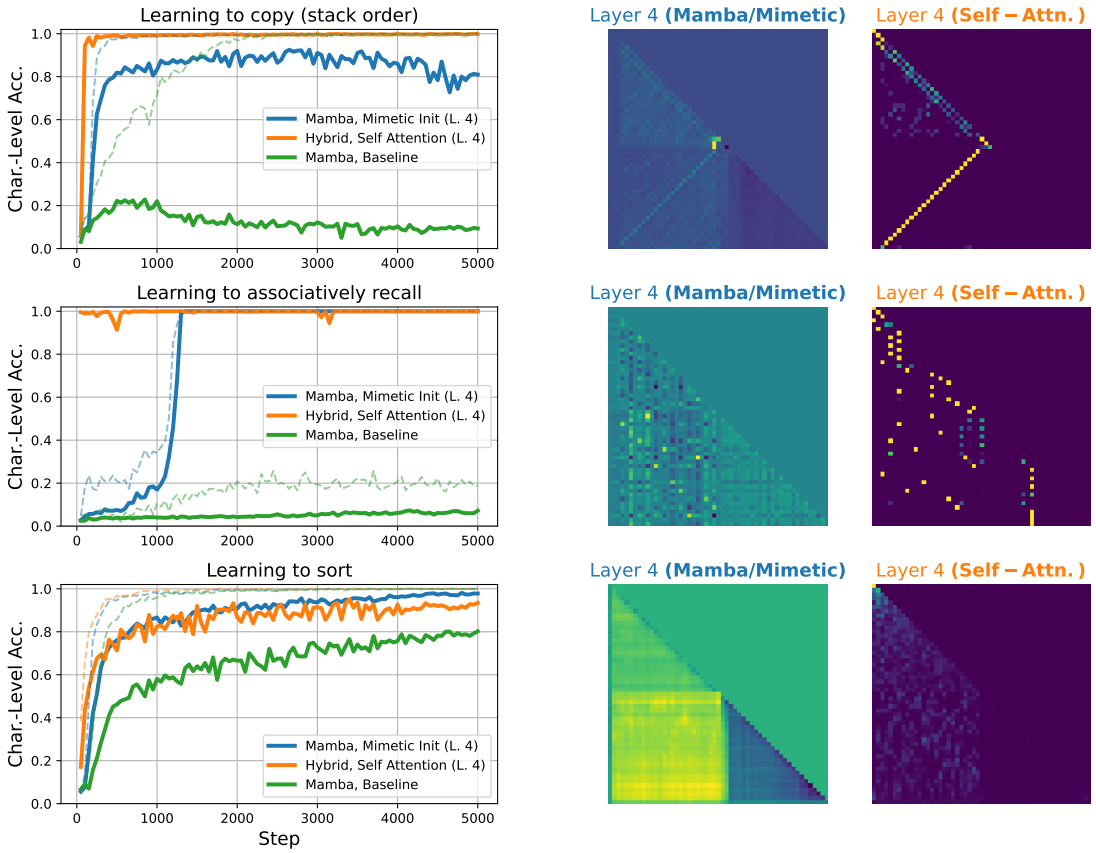


Figure 3.27: Mimetically initialized Mamba layers learn similar operations to Self-Attention layers in the same location *naturally* with no additional supervision on several tasks. Dotted lines: accuracy at training length (50), solid lines: generalizing to length 100.

effect when setting $A \approx 1$, with no apparent benefit to setting $\Delta \approx 1$; while setting $W_C^T W_B \approx 1$ has only a tiny effect, using identity convolution initialization seems somewhat harmful.

For Mamba 2, we see a similar advantage to using $A \approx 1$ initialization, and a advantage to $W_C^T W_B \approx 1$ even without $A \approx 1$, and the two interact to create even better models. Adding identity convolution initialization leads to much better performance still, reaching 100% accuracy in many cases. The positive interaction between $A \approx 1$ and $W_C^T W_B \approx 1$ and identity convolution is especially apparent for 300-long strings.

The difference in the best initialization strategy for the two architectures is likely explained by the removal of linear blocks after the convolutional layer in Mamba 2, as well as the addition of multiple state space heads. Unless otherwise noted, we use the observations above to determine our initialization strategy depending on the Mamba version: For Mamba 1, we use $A, \Delta \approx 1, W_C^T W_B \approx I$, and for Mamba 2 we add identity convolution initialization.

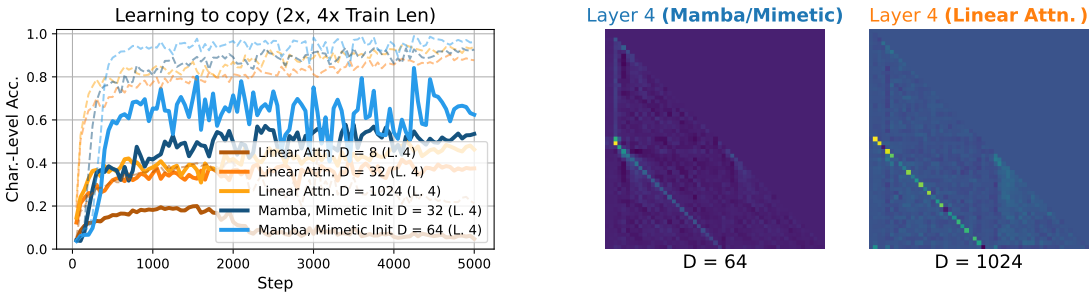


Figure 3.28: Simple linear attention underperforms Mamba even for very high head dimension, especially at generalization. Dotted lines: accuracy at length 100, solid: at length 200; train length: 50.

3.3.3 State Space Models want to be Transformers: Mimetic Initialization lets them get closer

Mimetic initialization leads to immediate and significant improvements in copying ability. In Fig. 3.24b, we can see that mimetic initialization allows a small 4-layer Mamba to learn to copy strings with twice the training length with reasonable accuracy in just a few hundred steps, which is far better than the tens of thousands of steps reported in previous work (Jelassi et al., 2024). Note that mimetic initialization leads to Mamba learning a state space layer whose attention map replicates the structure of that of self-attention in Fig 3.25; i.e., this layer has learned to (continue to) implement linear attention. Mimetic initialization allows Mamba to quickly learn to copy from scratch.

One mimetic init is all you need? We continue our investigation of using mimetic initialization to help Mamba learn recall tasks: Given our observations that a single self-attention layer is sufficient to learn these tasks to high fidelity, and that a single Mamba layer can roughly approximate this attention, we use mimetic init for *just one layer* in the same position (Layer 4) of an 8-layer Mamba.

In addition to copying (Fig. 3.25), we present results for additional three synthetic tasks in Fig. 3.27. First, we investigate copying in stack order, as unpacking the compressed string in most-recently-added order is potentially easier for SSMs. Unlike normal copying, baseline Mamba is able to fit to the training length, but it fails to generalize. Mamba with mimetic init fits the training length much faster and generalizes better, while the self-attention hybrid generalizes nearly immediately. The story is similar for multi-query associative recall – mimetic initialization leads to rapid learning and generalization to twice the length. We also consider the sorting task, where tokens are sampled without replacement from a vocab of size 512. Surprisingly, Mamba with mimetic init does even better than self-attention. Mimetic initialization results in large improvements for all synthetic tasks considered.

Is Mamba with mimetic init *just* linear attention? In Figures 3.25 & 3.27, notice that the mimetic initialized Mamba layer tends to mimic the corresponding self-attention layer in the

hybrid model; the resemblance is clear for copying in normal and stack order. For associative recall, it is less clear, but the Mamba layer looks significantly more like it could implement a induction-head-like function than typical Mamba layers. Similarly, the interpretation is unclear for sorting, but the overall structure matches. At a high level, it seems like Mamba attempts to learn an approximation to self-attention, but has much less capacity and sharpness. Consequently, we ask if our initialization merely turns state space layers into single-head linear attention layers.

In Figure 3.28, we present an ablation study where we replace the target Mamba layer in our copying experiment with simple causal linear attention with various head dimensions. According to Eq. 3.38, we may expect mimetic init to make Mamba layers equivalent to unkernelized linear attention layers with head dimension equal to the state dimension. Consequently, we compare Mamba with state size 32 to linear attention with head dimension 32, which comes relatively close. We plot generalization to $2\times$ and $4\times$-length in Fig. 3.28, as the difference for fitting to the training length is small. Nonetheless, Mamba still performs somewhat better than linear attention. Linear attention performance depends on the head dimension, with dimension 8 severely underperforming Mamba and dimension 1024 barely exceeding the performance of 32. In contrast, doubling the state dimension of Mamba to 64 substantially improves generalization performance. We visualize the difference in attention maps for the two operations; we can see that Mamba's is perhaps sharper/more consistent like that of self-attention. Combined with better performance on copying, we conclude that mimetic init Mamba layers are not *just* linear attention, but rather a related and superior (for this task) non-linear operation. The correlation between this "sharpness" and linear attention performance has been exploited by recent work (Zhang et al.).

3.3.4 Mimetic init experiments across architecture settings

Mimetic initialization improves the recall abilities of Mamba 1 and 2 over a variety of architecture settings and sequence lengths. For all Mamba 1 experiments, we use state size 32, though we explore different state sizes for Mamba 2, which has state size 128 unless otherwise noted. For Mamba 2, we use head dimension 64 for all experiments. All trials are for 5000 steps unless otherwise noted, and we swept over a small set of learning rates; our training pipeline is taken from Jelassi et al. (2024). *Note:* While mimetic initialization has a strong effect size for Mamba 1, the architecture generally struggles to copy for larger vocab sizes in the training lengths studied, so we present Mamba 2 results for most larger-scale experiments in the paper. Error bars are computed over five seeds.

Vocabulary sizes The larger the vocabulary, the more bits it should take to encode content of a token to enable copying, and the harder it may be to memorize and copy the sequence. While the previous work on copying focused on small vocabularies, we showcase the ability of mimetic init to improve copying even for large vocabularies in Fig. 3.29. For Mamba 1, mimetic init allows decent copying performance up until a point, and then degrades. In contrast, baseline never learns to generalize. For Mamba 2, mimetic init enables consistent $2\times$ length generalization across sequence lengths, preventing the degradation with vocab size demonstrated by the baseline.

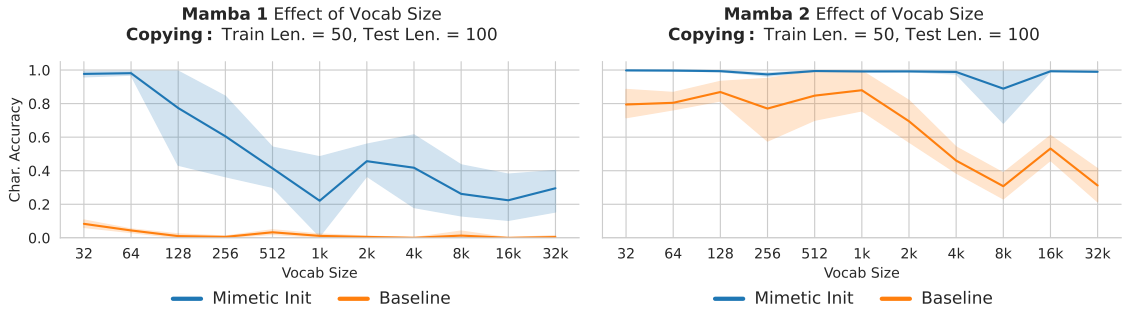
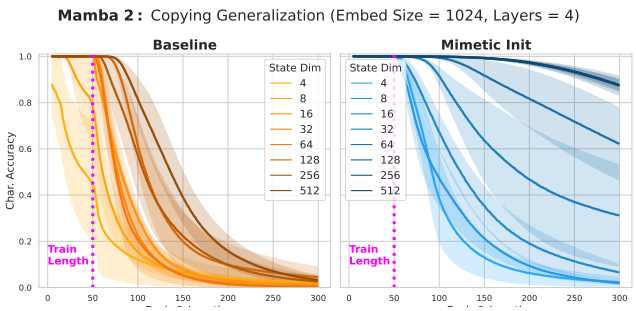
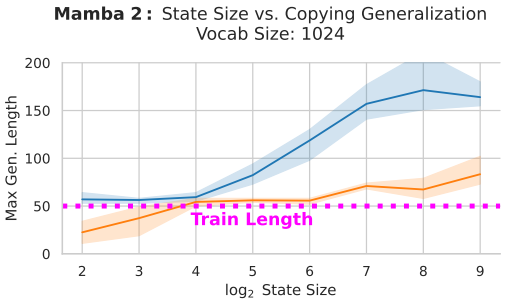


Figure 3.29: Mamba 2 with mimetic init can learn to copy even for large vocabulary sizes.



(a) State size vs. evaluation length



(b) State size vs. max $> 99\%$ gen. length

Figure 3.30: Mimetic initialization allows for better use of the state size for copying; capacity grows roughly linearly with state size, compared to almost not at all with default init.

State dimension The copying ability of Mamba should be directly related to its state size, according to Jelassi et al. (2024). This allows Mamba to more easily approximate self-attention-like maps, as we saw earlier. We show this is indeed the case in Fig. 3.30a. Indeed, for baseline Mamba 2, perfect copying at training length 50 is only possible for sufficiently large state size. However, if we use mimetic initialization, the additional capacity from the state size is much more efficiently used, and generalization (measured with the area under the curve) is far stronger – $\texttt{N} = 32$ with mimetic init achieves performance comparable to $\texttt{N} = 512$ with baseline init, a $16\times$ improvement in the use of capacity. We show another view on this data in Fig. 3.30b; generalization length hardly grows with the log of the state size using baseline initialization, while it grows linearly only after using mimetic initialization. Mimetic init allows Mamba 2 to get closer to its true compression/copying capacity.

Architecture size In Figure 3.31, we investigate mimetic init over different Mamba sizes (dimension, layers). Surprisingly, a mere two layers seems to be sufficient, with deeper networks improving generalization beyond $2\times$ length. With embedding size 1024, Mamba 2 can copy very well for a variety of depths; for multi-query associative recall, slightly deeper networks seem preferable. In almost all cases, mimetic initialization leads to superior generalization performance.

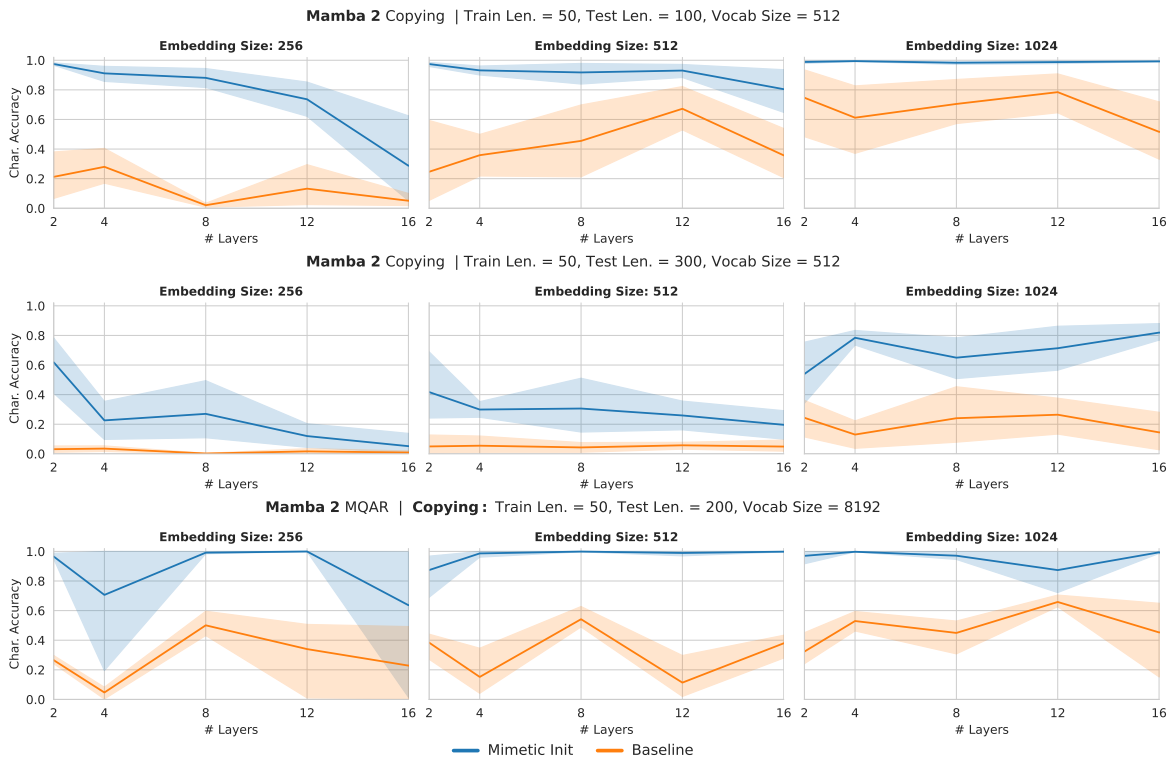


Figure 3.31: Mimetic initialization vs. Mamba 1/2 architecture sizes.

Sequence length Mimetic initialization lets us nearly perfectly fit to the training length even for longer strings for both copying and multiquery associative recall (Fig. 3.32). While baseline tends to struggle to learn to copy even 1000-long strings, mimetic initialization allows fitting to around 4000-long strings. For MQAR, baseline breaks down around 900-long strings, while mimetic initialization allows fitting to 1800-long or more. The benefits apply for better generalization as well, though Mamba still cannot strongly generalize to much longer strings than trained on.

3.3.5 Investigation of pretrained SSMs

Mimetic init mimics benefits of pretraining We hypothesized that Mamba's difficulty in copying may be an optimization issue rather than fundamental capacity limitations. That is, a Mamba that was first pretrained on a general text corpus may be a better representation of true copying abilities; i.e., one should never train from scratch (Amos et al., 2023). In Fig. 3.33, we see that finetuning a pretrained 130M Mamba to copy or do associative recall on 50-character strings results in good generalization, but training from scratch with mimetic init achieves similar results. Note that the pretrained Mamba had a much longer (> 1k) training length than our from-scratch trials. Considering this, our mimetic init results get impressively close (esp. for shorter strings; dotted lines).

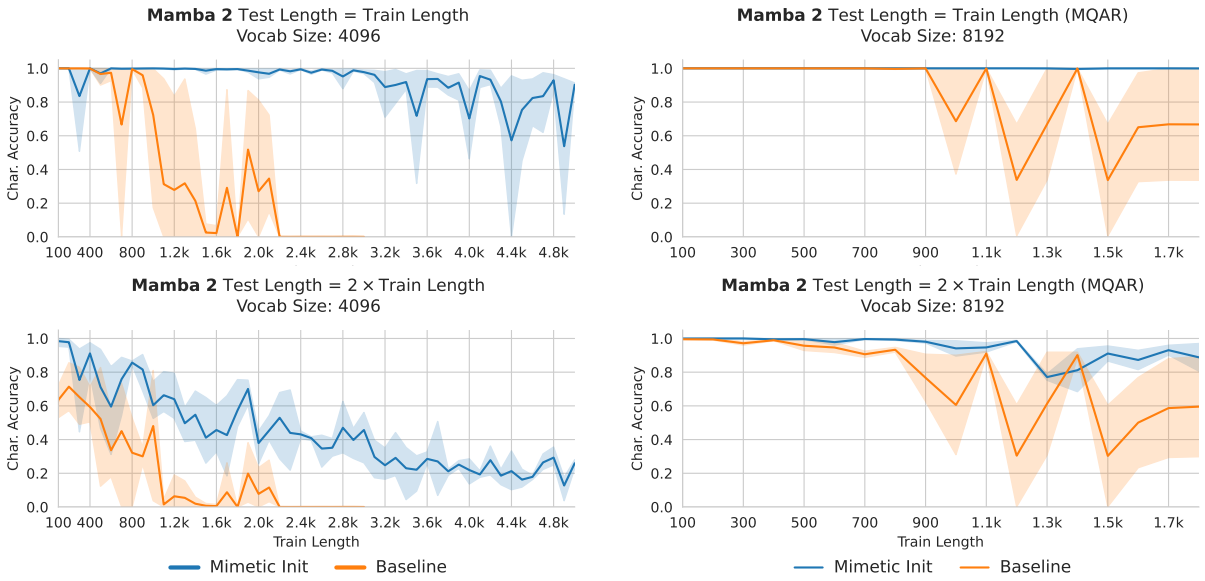


Figure 3.32: Mimetic init lets us nearly perfectly fit in-distribution even for long sequences on copying (left) and MQAR (right), and also boosts generalization performance (1024-dim 2-layer Mamba 2).

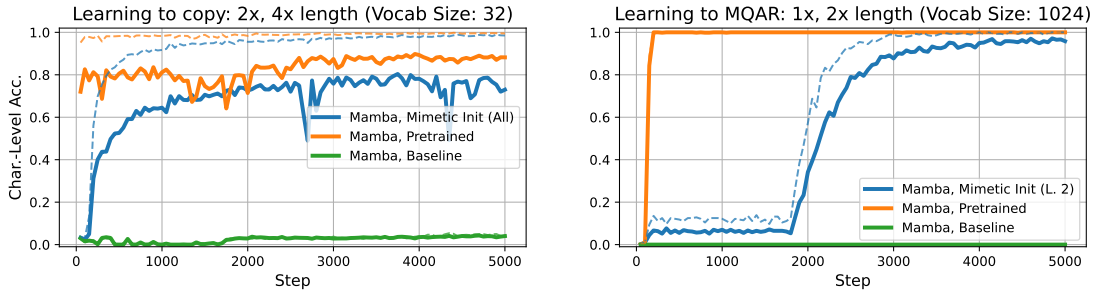
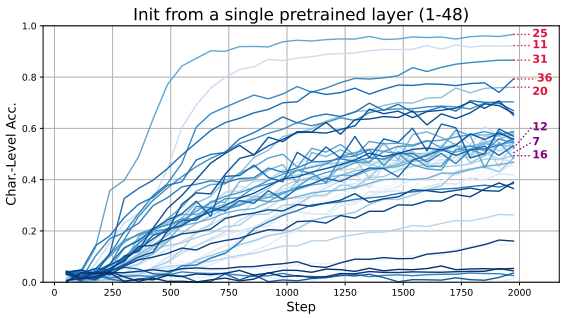


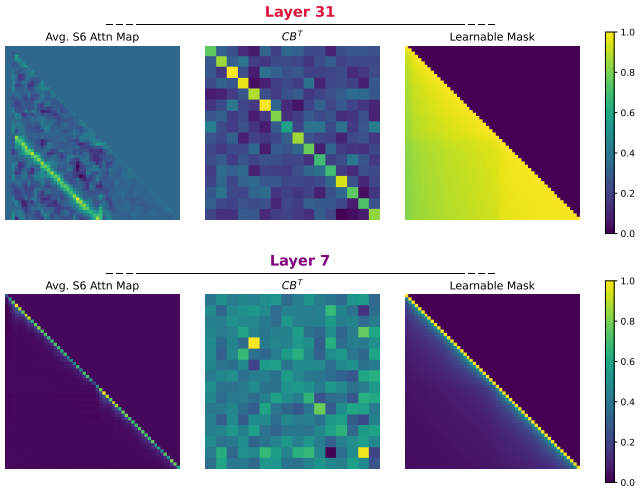
Figure 3.33: Pretrained 768-dim. 24-layer Mamba 1 vs. from-scratch training (w/ mimetic init).

Localizing the benefit of pretrained weights Based on our linear attention observations, the copying abilities of a pretrained Mamba may be localized to a few layers, so we explore the capabilities of individual layers: We use a pretrained teacher Mamba with layers $T_i : i \in [L]$, and then train L student Mambas where each of the $S_j : j \in [M]$ layers is initialized with $S_j \coloneqq T_i$ for $i \in [L]$. In this case, $L = 48$ and $M = 12$. Using these pretrained weights can make it much easier to learn to copy (Fig. 3.34a), but the effect size stands out for some particular layers, such as T_{31}.

We inspected the weights and attention maps of these layers to see what might be behind the improved performance; see some examples in Fig. 3.34b. Some layers such as T_{31} look like our mimetic initialized layers, with nearly all-ones average attention masks, correlated W_C, W_B weights, and lower diagonal structure, similarly to self-attention layers in hybrid Mambas earlier. That is, the structure our initialization provides seems to arise *naturally* in Mambas trained on sufficiently large and varied corpora, and may be fundamental to Mamba's copying and recall



(a) The weights of some pretrained Mamba layers serve as a good initialization for copying, even when those weights are repeated uniformly for all layers in the "student" model. The layers that work well as an initialization for copying tend to have correlated C, B weights and nearly-all-ones masks, such as Layer 31.



(b) Some pretrained Mamba layers have structure conducive to copying (Layer 31); others merely mix tokens with their nearby neighbors; from a 26-char test string.

Figure 3.34: The copying ability of a pretrained Mamba may be attributable to a fraction of its layers.

abilities.

3.3.6 Summary of contributions

We presented mimetic initialization for state space layers, a simple and closed-form technique to greatly improve the copying and recall abilities of state space models. Mimetic initialization makes state space layers mimic linear attention at initialization time, and also mimics the structure of state space layers that contribute to copying and recall abilities in pretrained models. Our technique allows to estimate capabilities of SSMs more accurately, which have been alternatively over- and under-estimated in the literature (Jelassi et al., 2024; Waleffe et al., 2024). Using a better initialization such as ours may assist in developing new architectures starting from a smaller scale, allowing for better predictions of their full-scale performance, as is often done in practice in testbeds (Poli et al., 2024). From a theoretical perspective, our particular construction may provide insights into the tradeoffs between state space layers and attention, and may help to study the recall vs. non-recall capabilities of state space layers. Improving the ability of state space layers to approximate attention has already been noted in followup work to the original Mamba architecture (Dao and Gu, 2024), and our initialization supports this concept. More broadly and together with previous work on mimetic initialization, our work helps to better understand pretraining, to some extent disentangling its dual purposes of storing knowledge and serving as a good initialization.

3.4 Mimetic initialization for MLPs

Mimetic initialization uses pretrained models as case studies of good initialization, using observations of structures in trained weights to inspire new, simple initialization techniques. So far, it has been applied only to spatial mixing layers, such convolutional, self-attention, and state space layers. In this work, we present the first attempt to apply the method to channel mixing layers, namely multilayer perceptrons (MLPs). Our extremely simple technique for MLPs—to give the first layer a nonzero mean—speeds up training on small-scale vision tasks like CIFAR-10 and ImageNet-1k. Though its effect is much smaller than spatial mixing initializations, it can be used in conjunction with them for an additional positive effect.

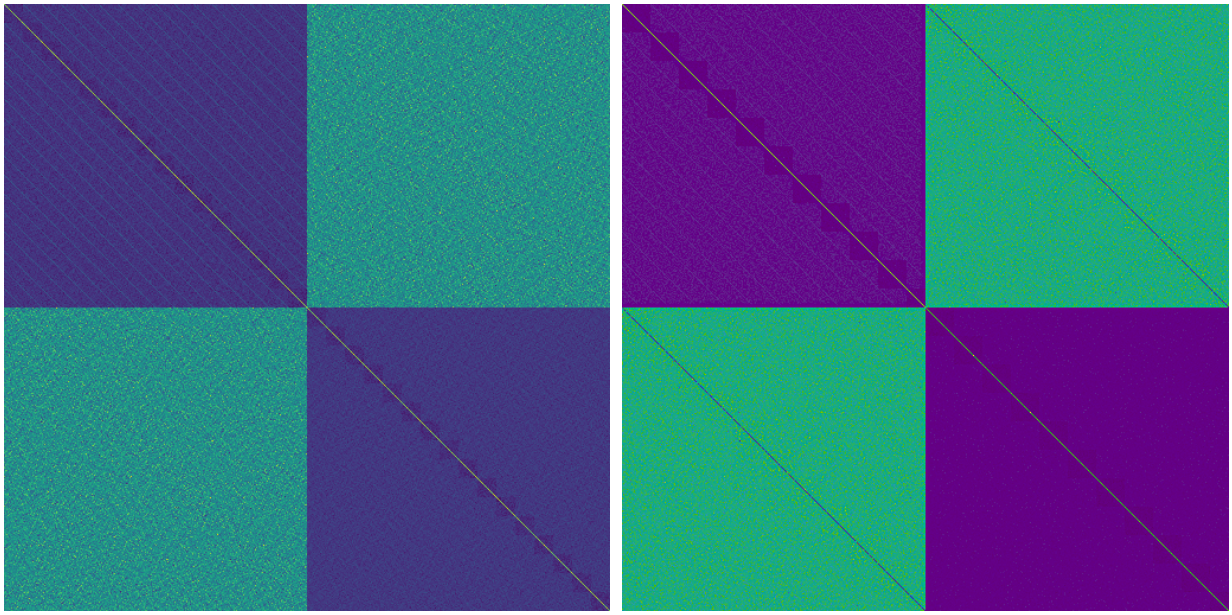
Background Neural network weights are typically initialized at random from univariate distributions, as in Xavier Glorot and Bengio (2010) and Kaiming He et al. (2015) initializations. These techniques focus on the *signal propagation* perspective, where the variance of the univariate random weights is scaled according to the dimensions of the layer weights with the goal of preventing vanishing and exploding gradients.

However, modern deep networks use normalization layers such as BatchNorm (Ioffe and Szegedy, 2015) and LayerNorm (Lei Ba et al., 2016) in addition to adaptive optimizers like Adam (Kingma and Ba, 2017); taken together, these methods make precise control of scale at initialization somewhat less critical. Of course, scale still plays an important role (Kunin et al., 2024).

The most effective initialization is *pretraining*, or transfer learning Kolesnikov et al. (2020); pretrained networks have stored transferrable knowledge that can be leveraged to quickly adapt to downstream tasks. Consequently, pretrained networks are much easier to train. But could it be that this transferrable knowledge is only part of the story— that the geometry of the weight space of pretrained networks simply makes them easier to optimize, in a way we could capture or study without pre-training?

Related work Recently, several works on *mimetic initialization* have proposed that the effect of pretraining can be decomposed into two components: (1) *storing knowledge* and (2) *serving as a good initialization* (Zhang et al., 2022; Trockman et al., 2022; Trockman and Kolter, 2023; Trockman et al., 2024; Krizhevsky et al., 2017). Moreover, it seems that to some extent, the hypothesized *good initialization* component can be disentangled from the other by "hand", in closed form. Mimetic initialization uses pretrained models as case studies of good initialization, and summarizes the weight space structures observed in simple but *multivariate* initialization schemes that specify the high-dimensional statistical dependence between weights.

A mimetic initialization for convolutions Trockman et al. (2022) specified the structure of weights for individual filters depending on the depth of the layer. Mimetic initialization for self-attention layers Trockman and Kolter (2023) attempted to make self-attention more "convolutional" (localized receptive field) at initialization, among other observations: the query and key weights should be correlated, value and projection weights anti-correlated, and sinusoidal position embeddings should be used at initialization; see also Zheng et al. (2024). Most recently, mimetic initialization for state space layers Trockman et al. (2024) used the correspondence



(a) ConvNeXt (b) ViT (enhanced; may need to zoom for structure)

Figure 3.35: Full empirical covariance matrices for unrolled MLP weights W_1, W_2 (jointly).

between state space layers and linear self-attention layers to leverage the previous technique. These methods allow training networks in fewer steps and to higher final accuracies, especially in small-scale and data-limited settings.

This work All previous work on mimetic initialization has focused on *spatial mixing* layers, i.e., large-filter convolutions, self-attention layers, and state space layers. In this work, we investigate whether the same technique may apply to some *channel mixing* layers, in particular MLPs, one of the most important primitives in deep learning, i.e., y = $W_2\sigma(W_1x)$ for nonlinear activation function σ and projections $W_1 \in \mathbb{R}^{p\times n}$, $W_2 \in \mathbb{R}^{n\times p}$ and $x \in \mathbb{R}^n$. The importance of the initialization of the value and projection weights in Trockman and Kolter (2023) could be viewed as an application of mimetic init to channel mixing layers; however, this finding only held in conjunction with the attention map (query/key) init itself. In contrast, we focus on MLPs in isolation. While we observe interesting statistical structure in trained MLPs, our only empirically-backed finding so far suggests a modification to the mean of W_1.

3.4.1 Understanding the covariance structure of trained MLPs

Previous work on mimetic initialization has focused on the covariance structure of pretrained weights. Trockman et al. (2022) studied the covariance structure of convolutional filters *within single networks*, i.e., a network with embedding dimension n has n filter samples per l layers, from which one can calculate l empirical covariance matrices. Trockman and Kolter (2023) study the cross-covariance of query/key and value/projection weights, again focusing on samples of rows or columns within individual networks.

In this work, we adopt a somewhat more powerful technique; instead of studying empirical covariances calculated *within single networks* over slices of weights (rows, columns, filters, etc.), we study empirical covariances calculated over *populations of networks*. That is, we study the distribution of $[\vec{(}W_1)\ \vec{(}W_2)] \in \mathbb{R}^{2np}$ rather than, say, $[W_1[i]\ W_2[:, i]] \in \mathbb{R}^{2n}$.

We start our investigation using by training tiny ConvNeXt (Liu et al., 2022b) models, a simple convolutional architecture similar to ConvMixer (Trockman and Kolter, 2022) that uses isolated MLP layers (alternately with depthwise convolution) instead of linear layers. We train $\approx 10^4$ such models on CIFAR-10 classification to estimate weight space covariances; as this is computationally expensive, we use models with only a few thousand or up to $\approx 10^4$ parameters. For example, we use just 3–4 layers and an internal dimension of around 16–32 with MLP expansion factor 2. We train over thousands of different seeds and unroll the MLP weights W_1, W_2 to compute the full empirical covariances of each weight $\mathsf{Cov}(\vec{W})$ and cross covariance $\mathsf{Cov}(\vec{W}_1, \vec{W}_2)$.

In Fig. 3.35a, we show the full empirical covariance matrix of $[\vec{(}W_1)\ \vec{(}W_2)]$. Notably, the covariances corresponding to each weight matrix are striped. This suggests that the rows or columns (depending on row/column major order for unrolling) are correlated in trained MLPs. For example, each row or column could have a nonzero "group mean" while the overall weight matrix still has a global/"grand" mean of zero. This group mean could itself be randomly distributed with mean zero. We also note that the stripes tend to be more pronounced in $\mathsf{Cov}(W_1)$ than in $\mathsf{Cov}(W_2)$.

We attempt to mimic this very simple structure at initialization time. We add a small amount of random noise to $b_n \sim \mathcal{N}(0, \sigma I_n)$ to W_1, broadcasting over rows:

$$W_1' = W_1 + \mathbf{1}_p b_n^T. \tag{3.39}$$

This mimics the structure seen in pretrained weights, and we find that it also speeds up training considerably. However, we note that an even simpler solution exists to mimic the structure in Fig. 3.35a: simply add a single, constant small bias term $b \in \mathbb{R}$ to W_1:

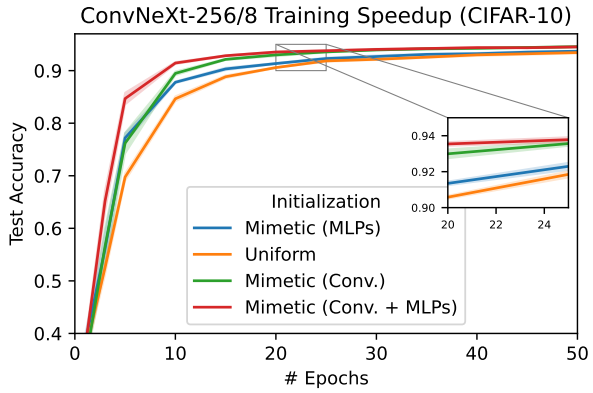
$$W_1' = W_1 + b\mathbf{1}_p\mathbf{1}_n^T. \tag{3.40}$$

Both strategies result in randomly-sampled matrices with significant stripes as in Fig. 3.35a. In our experiments, this extremely simple tweak is surprisingly effective at increasing the trainability of small ConvNeXts.

In Fig. 3.35b, we do the same analysis for a small Vision Transformer (ViT) trained in the same setup. We see the same patterns as in Fig. 3.35a; moreover, we see that W_1 and W_2 weights are apparently anti-correlated as noted for the value and projection weights in previous work (Trockman and Kolter, 2023); that is, a mimicking initialization may be: $W_1' = \frac{1}{2}(W_1 - W_2)$. We do not see this pattern within ConvNeXt. Further, we saw no benefit to initializing MLP weights to be anticorrelated, unlike our W_1-mean initialization above.

3.4.2 Experiments on channel plus spatial mixing mimetic init

We test our technique on small ConvNeXt and ViT models trained on CIFAR-10 with RandAugment, using the same pipeline as in (Trockman et al., 2022). We made some noteworthy changes



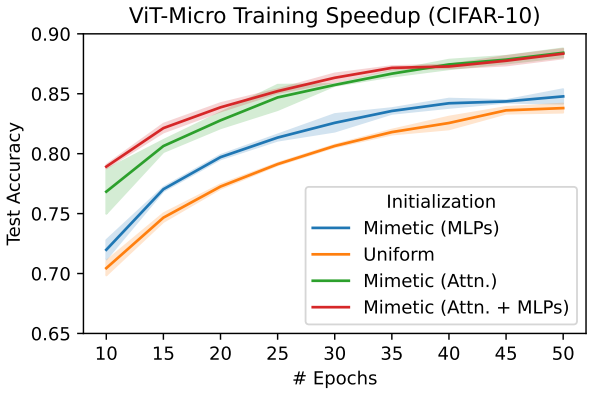


Figure 3.36: CIFAR-10 experiments for ConvNeXt and ViT in conjunction with previous mimetic inits for convolutional and self-attention layers. The effect of the MLP init is significant in early training, but eventually evens out. The effect of conv. and attn. mimetic init, however, remains. *Note:* each point in the graph represents the accuracy of a completed training run of x epochs, mean/std. reported over 5 seeds.

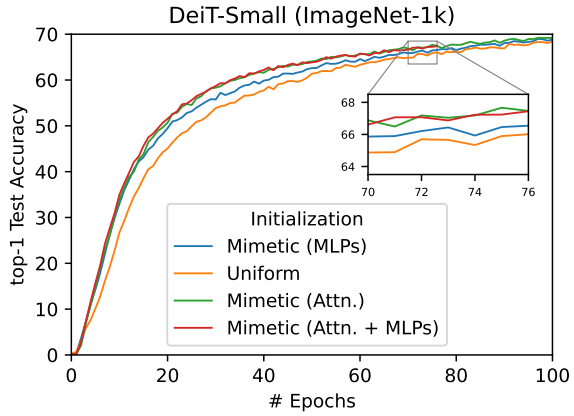


Figure 3.37: Our MLP init improves early ImageNet-1k training in a data-efficient ViT, and maintains a 0.5% advantage over baseline for the full training time.

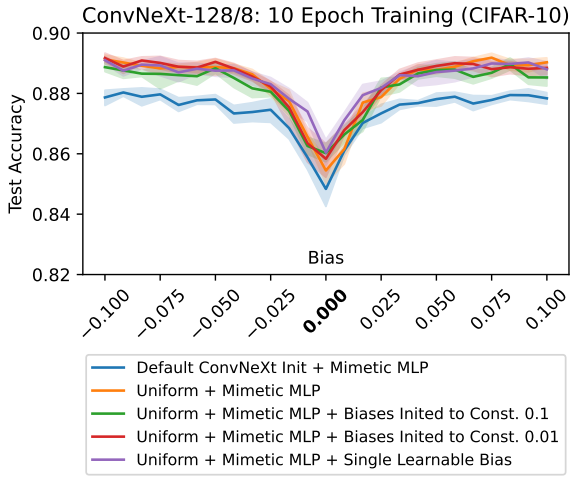


Figure 3.38: Sweep of the bias parameter b of our init; across several baselines, $b = 0$ is noticeably suboptimal. Mean/std. over 5 seeds per b.

to the default initialization scheme of ConvNeXt, beyond our proposed technique: we replaced the truncated normal initialization with fixed standard deviation $\sigma = 0.02$ and zero bias initialization with the standard Kaiming uniform initialization in PyTorch; this lead to broadly improved accuracies for our training setup. The CIFAR-10 models presented use 2×2 patches for 32×32 inputs, and the ImageNet models 16×16 for 224×224 inputs.

We find that the simple initialization tweak in Eq. 3.40 improves ConvNeXt and ViT results for small scale and short-duration training. In Fig. 3.38, we show accuracy for a variety of settings of b; a wide variety of settings improve performance, with a notable dip at $b = 0$. However, unlike previous work on mimetic initialization, it seems that advantage of using our proposed MLP init decreases with increasing training time (see Fig. 3.36).

ConvNeXt results In Fig. 3.36, we show the effect of training with various mimetic init techniques compared to uniform (default) initialization, on an isotropic ConvNeXt-256/8 with 7×7 filters. Our MLP init significantly increases accuracy for short-duration training (such as 10-20 epochs), though the effect tapers off for longer training times (such as 50). The MLP init also works with the convolution init from Trockman et al. (2022), together outperforming either mimetic init alone; but in contrast, the convolution init maintains an advantage over uniform even after many epochs.

ViT results We also trained tiny ViTs with mimetic initialization, which have dimension 192, depth 8, and 3 heads; see Fig. 3.36. The benefit of our MLP init persists for longer training times compared to ConvNeXt, but the additive effect with self-attention mimetic init may be smaller. In Fig. 3.37, we trained a DeiT-Small (22M params) on ImageNet-1k using the training pipeline from Touvron et al. (2021b). Our init results in considerable initial gains in accuracy, though the difference narrows significantly with training time; using our MLP init *with* self-attention mimetic init results in rather small gains. The difference may be larger in a simpler ResNet-style training pipeline, as in Trockman and Kolter (2023). Nonetheless, our method results in a consistent advantage of 0.5% over 100 epochs, achieving 68.8% over the course of 100 epochs, compared to 68.4% for the baseline.

Is it really the init? We thought our method may just be compensating for the default linear layer bias of zero in the ConvNeXt implementation, but this was not the case— see Fig. 3.38. One baseline is to simply add a learnable scalar bias per W_1, as $b\mathbf{1}_p\mathbf{1}_n^T x = b \cdot \mathsf{sum}(x) \cdot \mathbf{1}_p$. This does not absorb the effect of our init (see Fig. 3.38). We also attempted to change the init of the bias of the linear layer itself to a small constant. However, no baselines explain the effect of our MLP init.

3.4.3 Further structures in the weight space covariance

While we computed the empircal covariance matrices of *entire* networks, we have only so far used the blocks of the covariance matrix corresponding to individual MLP layers: the covariance and (within-layer) cross covariance of their weights W_1 and W_2. In this section, we point

out some notable structures from our investigate that have not (yet) yielded useful initialization schemes.

In Figure 3.39, we show the entire covariance matrix for the full weight vector of a tiny, four-layer ConvNeXt trained on CIFAR-10. One minor but notable observation is that biases tend to be correlated with adjacent weights. But more significantly, we note that weights are actually correlated across layers: the second-layer MLP weights W_2 are sometimes slightly correlated across distant layers, as highlighted in orange in Figure 3.39. For example, the last block's MLP W_2 is anticorrelated with the first block's MLP W_2. Downstream weights are also often correlated with the weights in the patch embedding layer (first row of Figure 3.39). We did not attempt to use any of these observations to create an initialization technique, but they may provide insights into neural network training dynamics.

We also investigated empirical covariance matrices including some computed properties of the weights, for example, the product of adjacent weights. In Figure 3.40, we show the correlation between convolutional filter weights and the *product* of adjacent downstream MLP weights. We note that (central) filter pixels are anticorrelated with the magnitude of the corresponding diagonal of the product W_1W_2. For example, if the filter in channel c tends to increase the magnitude of its input, the net effect of applying the MLP is to compensate by decreasing magnitude (roughly speaking) in channel c. We attempted to use this observation in an initialization where filter weights are correlated with MLP weights, but found no significant results. Nonetheless, we think this is an interesting observation about the structure of the weight space in some convolutional networks.

3.4.4 Summary of contributions

We presented a simple concept for a mimetic initialization for MLPs; our method is to simply initialize the first layer of MLPs with a small nonzero mean. We have a small amount of preliminary evidence to suggest that using Eq. 3.39 instead of Eq. 3.40 for larger-scale training may be better; Eq. 3.39 maintains the global mean of zero and may encode a weaker inductive bias than Eq. 3.40. However, more research at larger scale is necessary to draw conclusions. While our results are not as robust as those in the mimetic initialization works for spatial mixing layers, we think our finding is nonetheless interesting in terms of understanding the "good initialization" component of pretraining. We also suspect that, like previous work, our technique would be most important in data-limited settings.

We tried to exploit one of the simplest structures we have seen in the weight space, but many others exist, such as the anticorrelation of weights in ViT MLPs. More generally, we think that our method of studying populations of pretrained network weights, as opposed to samples of weights within single networks, could lead to further insights in the growing field of weight space learning and analysis.

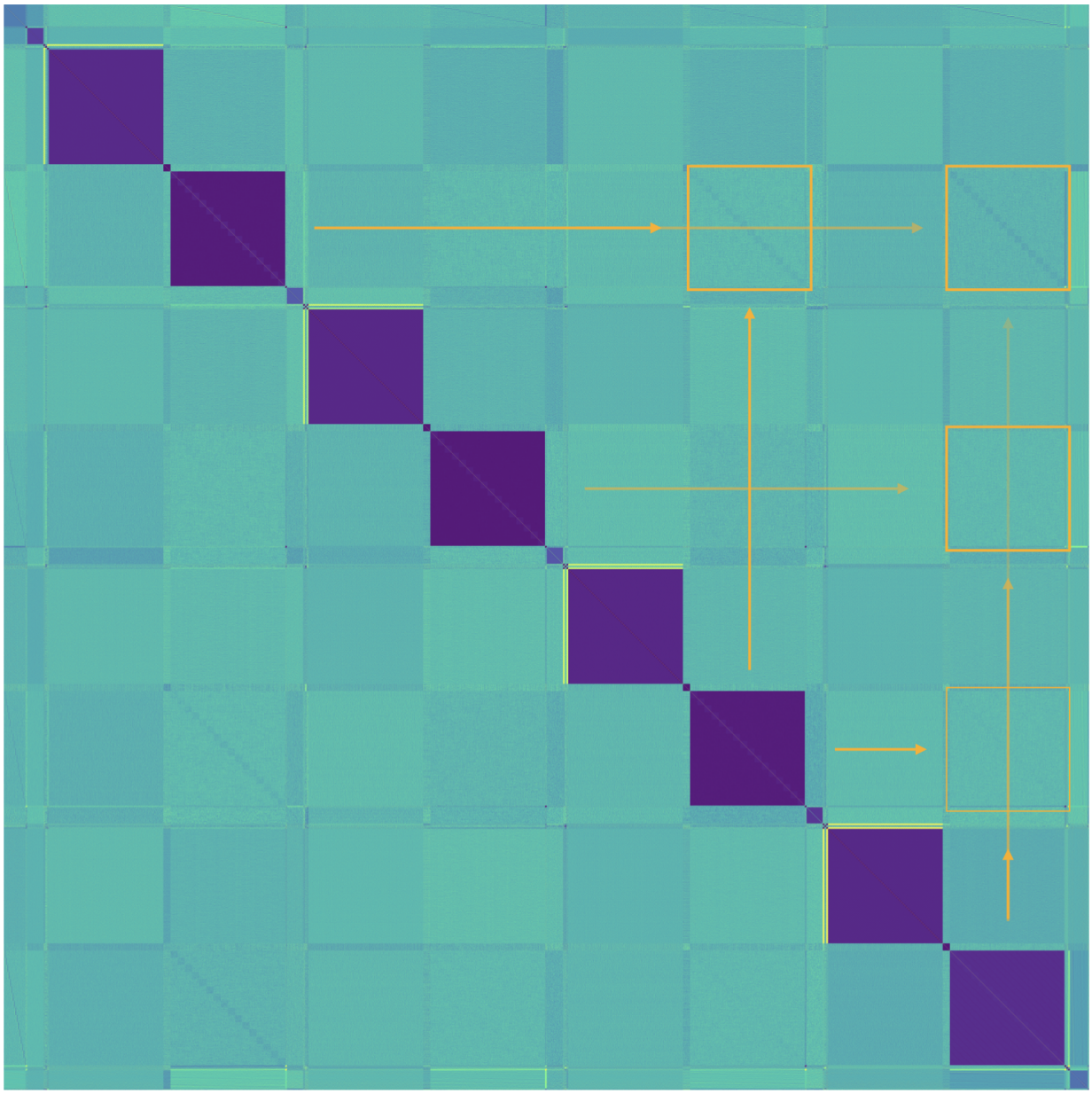


Figure 3.39: The empirical covariance matrix of ≈16,000 tiny 4-layer ConvNeXts trained on CIFAR-10, including convolution weights, MLP weights, biases, and LayerNorm weights. The second weight matrices (the one that "writes" to the residual stream) of the MLP layers are correlated across layers to some extent. (We may recommend folding in the LayerNorm weights.)

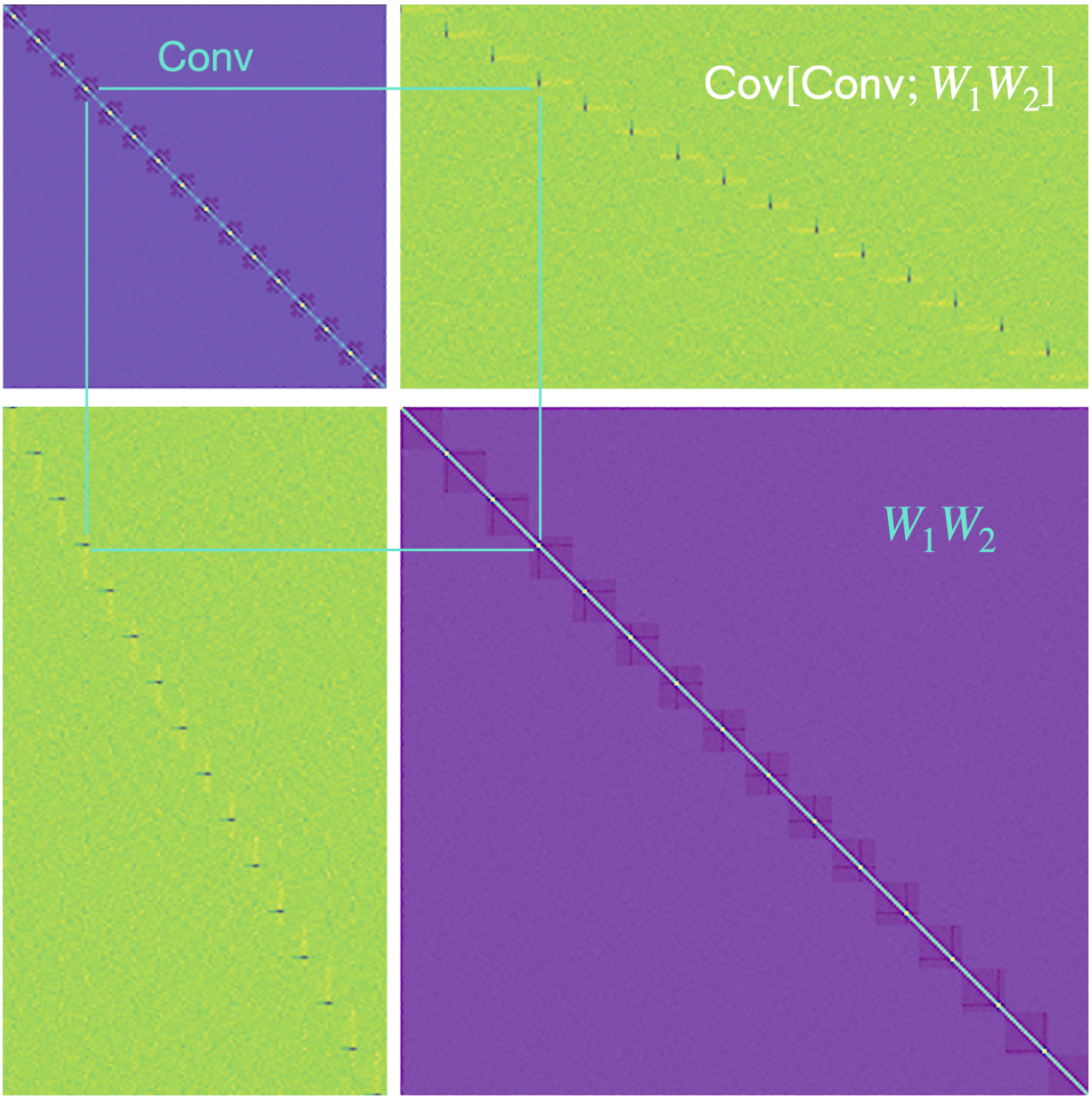


Figure 3.40: The central pixel of the c^{th} convolutional filter is correlated with the c^{th} diagonal of the product of the weight matrices of the downstream MLP layer. Based on the population-level empirical covariance calculated from ≈16,000 ConvNeXt training runs on CIFAR-10. This has not yielded an effective initialization strategy (yet).

Chapter 4

Conclusion

In the preceding three works, we have provided evidence that our hypothesized *good initialization* component of pretraining can be captured in closed form. In particular, we diagnosed the notable difficulty of training Vision Transformers on small-scale datasets through our structured initialization; we demonstrated that there may exist convolutional filters that are better than those traditionally found by gradient descent which are enabled through our initialization; and we showed our first results on mimetic initialization for language modeling by enhancing the recall and copying abilities of State Space Models. We also explored the covariance structure of the *entire weight space* of small neural networks, finding a simple mimetic initialization technique for MLPs in addition to other surprising details about cross-layer dependencies.

While our mimetic initialization techniques work well for vision tasks even at moderately large scale, *e.g.,* for large models on ImageNet-1k, we do not yet have evidence that our initialization is beneficial for very large-scale training, particularly on general language modeling. However, our promising vision results suggest that a good initialization may lead to *fundamentally better* local optima than one could attain with random initialization alone, and we think more research is warranted to better understand mimetic initialization and to attempt to apply it to large-scale language modeling. We propose some future directions below.

4.1 Future work

Understanding the *good initialization* component of pretraining. The previous three works on mimetic initialization have used a constructive approach: inspired by the qualitative structure of the weights in pretrained models, we proposed particular structure for the covariance of the weights, and empirically validated its effectiveness. In contrast, we propose to isolate the initialization component of pretraining in future work; for example, Neyshabur et al. (2020) suggested as future work that one could reuse only the top singular values and directions from a pretrained model for an initialization scheme. One could perform various ablations (*e.g.*, random permutations) to retain only the high-level statistical properties of the weights to attempt to localize the initialization-based components of pretraining.

Mimetic initialization across layers. In our study of the covariance structure of the entire weight space, we have found correlations between weights *across layers*, for example, between adjacent convolutional and MLP layers, or even as distant as between the first and last layer MLPs. It may be possible to use these observations to create a mimetic initialization that takes the structure of the entire network into account.

Mimetic initialization for self-attention in language models. In our first work on mimetic initialization for self-attention layers, we found that the story was more complicated for language models compared to vision models, and we subsequently found only marginal effects from our technique on language tasks. However, our recent work on state space layers uses largely the same techniques, leading to substantial improvements on synthetic language benchmarks. We speculate that a more comprehensive study of mimetic initialization for self-attention layers may reveal significant effects, at least on the level of synthetic benchmarks, which could inform how to modify the initialization for large-scale pre-training. We have limited, preliminary evidence that mimetic initialization can benefit larger-scale language pretraining in two settings: (1) using mimetic initialization for just 1-2 early layers, or even just a few heads, may improve pretraining performance in Mamba architectures (2) mimetic initialization may have nontrivial effect size for language pretraining if the embedding layer is recycled from another model.

Weight recycling and cross-architecture weight transfer. During our research on mimetic initialization, we have found that pretrained weights are simply easier to optimize, even after various permutations and slices. For example, one can train small models to initialize larger ones, or subsample weights from pretrained networks to initialize smaller ones. Convolutional filters may be upsampled, downsampled, or padded and reused in networks with different design choices. Just one pretrained layer radically impacts training, allowing untrained layers to learn faster. Even more surprisingly, we have found that weights can actually be transferred across operations, in particular, between self-attention and state space layers; self-attention query and key weights are an excellent choice of initialization for state space layer weights, and this observation partially led to our investigation of mimetic initialization for state space models. We think this direction is worth future work.

Appendix A

Implementations

A.1 ConvMixer Implementation

```
import torch.nn as nn

class Residual(nn.Module):
    def __init__(self, fn):
        super().__init__()
        self.fn = fn

    def forward(self, x):
        return self.fn(x) + x

def ConvMixer(dim, depth, kernel_size=9, patch_size=7, n_classes=1000):
    return nn.Sequential(
        nn.Conv2d(3, dim, kernel_size=patch_size, stride=patch_size),
        nn.GELU(),
        nn.BatchNorm2d(dim),
        *[nn.Sequential(
                Residual(nn.Sequential(
                    nn.Conv2d(dim, dim, kernel_size, groups=dim, padding="same"),
                    nn.GELU(),
                    nn.BatchNorm2d(dim)
                )),
                nn.Conv2d(dim, dim, kernel_size=1),
                nn.GELU(),
                nn.BatchNorm2d(dim)
        ) for i in range(depth)],
        nn.AdaptiveAvgPool2d((1,1)),
        nn.Flatten(),
        nn.Linear(dim, n_classes)
    )
```

Figure A.1: A more readable PyTorch (Paszke et al., 2019) implementation of ConvMixer, where h = `dim`, d = `depth`, p = `patch_size`, k = `kernel_size`.

```
def ConvMixer(h,d,k,p,n):
 S,C,A=Sequential,Conv2d,lambda x:S(x,GELU(),BatchNorm2d(h))
 R=type('',(S,),{'forward':lambda s,x:s[0](x)+x})
 return
  ↪ S(A(C(3,h,p,p)),*[S(R(A(C(h,h,k,groups=h,padding=k//2))),A(C(h,h,1)))
  ↪ for i in range(d)],AdaptiveAvgPool2d(1),Flatten(),Linear(h,n))
```

Figure A.2: An implementation of our model in less than 280 characters, in case you happen to know of any means of disseminating information that could benefit from such a length.
All you need to do to run this is `from torch.nn import *`.

This section presents an expanded (but still quite compact) version of the terse ConvMixer implementation that we presented in the paper. The code is given in Figure A.1. We also present an *even more terse* implementation in Figure A.2, which to the best of our knowledge is the first model that achieves the elusive dual goals of $80\%+$ ImageNet top-1 accuracy while also fitting into a tweet.

A.2 Implementation: Mimetic initialization for convolutional layers

```
def ConvCov(k, s):
  C = np.zeros((k**2,)*2)
  for i, j in np.ndindex(k,k):
    C[k*i:k*i+k,k*j:k*j+k] = Gauss(k,j,i,s)
  Z,l = Gauss(k,k//2,k//2,s),np.ones((k,k))
  S, M = np.kron(l, Z),np.kron(Z, l)
  return 0.5 * (M * (C - S) + C * S)
```

```
def Gauss(k, mx, my, s):
  res = np.zeros((k, k))
  for i, j in np.ndindex(k,k):
    cx,cy = (j-mx-k//2-1)%k,(i-my-k//2-1)%k
    z = ((cx-k//2)**2+(cy-k//2)**2)/s
    res[i, j] = np.exp(-0.5*z)
  return res.reshape(k, k)
```

Figure A.3: Implementation of our convolution covariance construction in NumPy.

```
def Initialize(wconv, d, s0, sv, sa):
  c, _, ks, _ = wconv.shape
  s = s0 + sv * d + 0.5 * sa * d**2
  cov = ConvCov(ks, s).reshape((ks,)*4).transpose(0,2,1,3).reshape((ks**2,)*2)
  filters = np.random.multivariate_normal(np.zeros(ks**2), cov, size=(c,))
  wconv.data = torch.tensor(filters.reshape(c,1,ks,ks),dtype=wconv.dtype,device=wconv.device)

# Find depthwise convolutional layers
convs = [x for x in model.modules() if isinstance(x, nn.Conv2d) \
  and len(x.weight.shape) == 4 and x.weight.shape[1] == 1]

# Initialize them according to variance schedule
for i, conv in enumerate(convs):
  Initialize(conv.weight, i / (len(convs) - 1), 0.16, 0.32, 2.88)
```

Figure A.4: Code to use our covariance construction and variance schedule to initalize depthwise convolutional layers in PyTorch. `wconv` is the weight of a depthwise convolutional layer (`nn.Conv2d`), and $\texttt{d} \in [0, 1]$ is its depth as a fraction of the total depth.

Appendix B

Additional Experiments

B.1 Covariance Structure: Hyperparameter Grid Searches & Experimental Setup

CIFAR-10 hyperparameter search. We chose an initial setting of our method's three hyperparameters via visual inspection, and then refined them via small-scale grid searches. For CIFAR-10 experiments, we searched over parameters for ConvMixer-256/8 with frozen 9×9 filters trained for 20 epochs, and chose $\sigma_0 = .08, v_\sigma = .37, a_\sigma = 2.9$ for 2×2-patch models, and found the optimal parameters for 1×1-patch models to be approximately doubled. However, note that our initialization is quite robust to different parameter settings, with the difference from our doubling choice being less than 0.1% (see Figure B.1). We used the same parameters across all kernel sizes, as well as for ConvNeXt, a choice which is likely sub-optimal; our search only serves as a rough heuristic.

ImageNet-1k hyperparameter search. We did a small grid search using a ConvMixer-512/12 with 14×14 patches and 9×9 filters trained for 10 epochs on ImageNet-1k (see Appendix B.3), from which we chose two candidate settings: $\sigma_0 = .15, v_\sigma = .5, a_\sigma = .25$ for frozen-filter models and $\sigma_0 = .15, v_\sigma = 0.25, a_\sigma = 1.0$ for thawed models. We use these parameters for all the ImageNet experiments, even for models with different patch and kernel sizes (*e.g.,* ConvNeXt). This demonstrates that *hyperparameter tuning is optional* for our technique; its transferability is not surprising given our results in Sec. 3.1.1.

B.1.1 CIFAR-10 Grid Searches

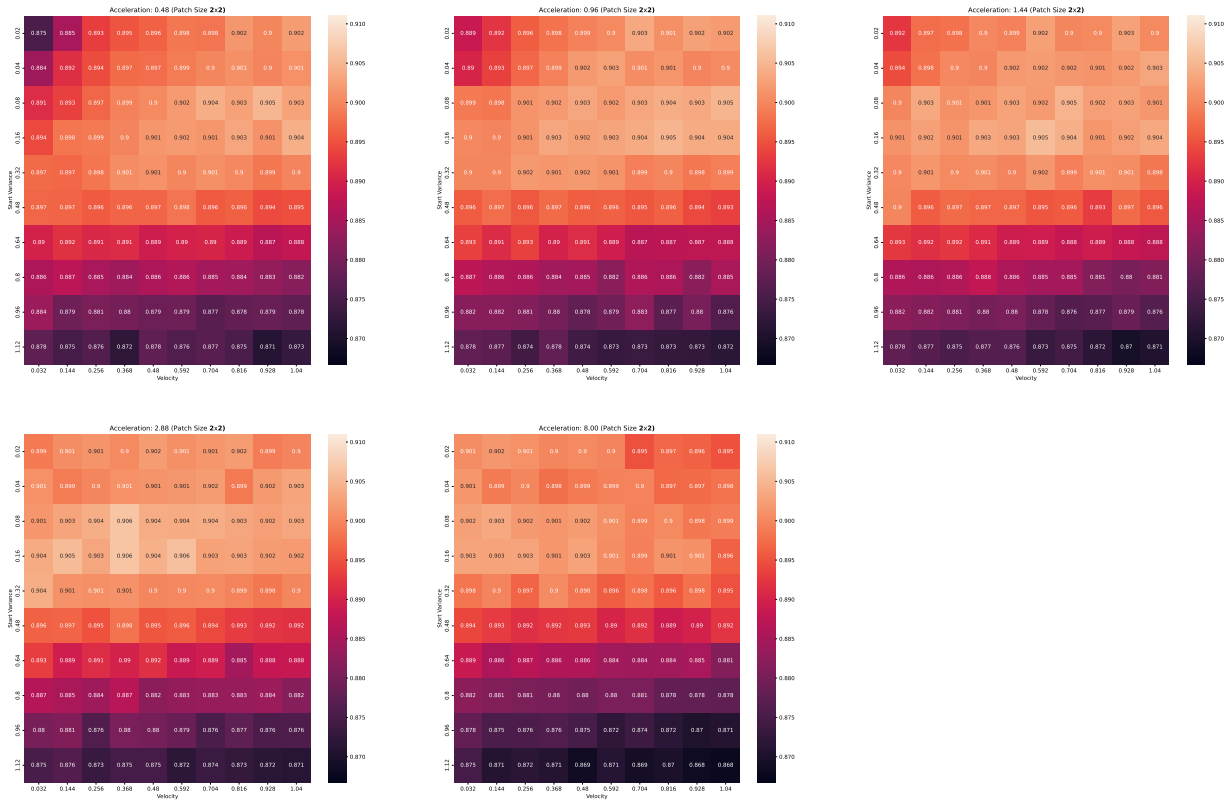


Figure B.1: Grid search over initialization parameters $\sigma_0, v_\sigma, a_\sigma$ for ConvMixer-258/8 with 9×9 *frozen* filters and 2×2 patches trained for 20 epochs on CIFAR-10. Note that the performance of uniform initialization is only ≈85%, i.e., almost all choices result in *some* improvement.

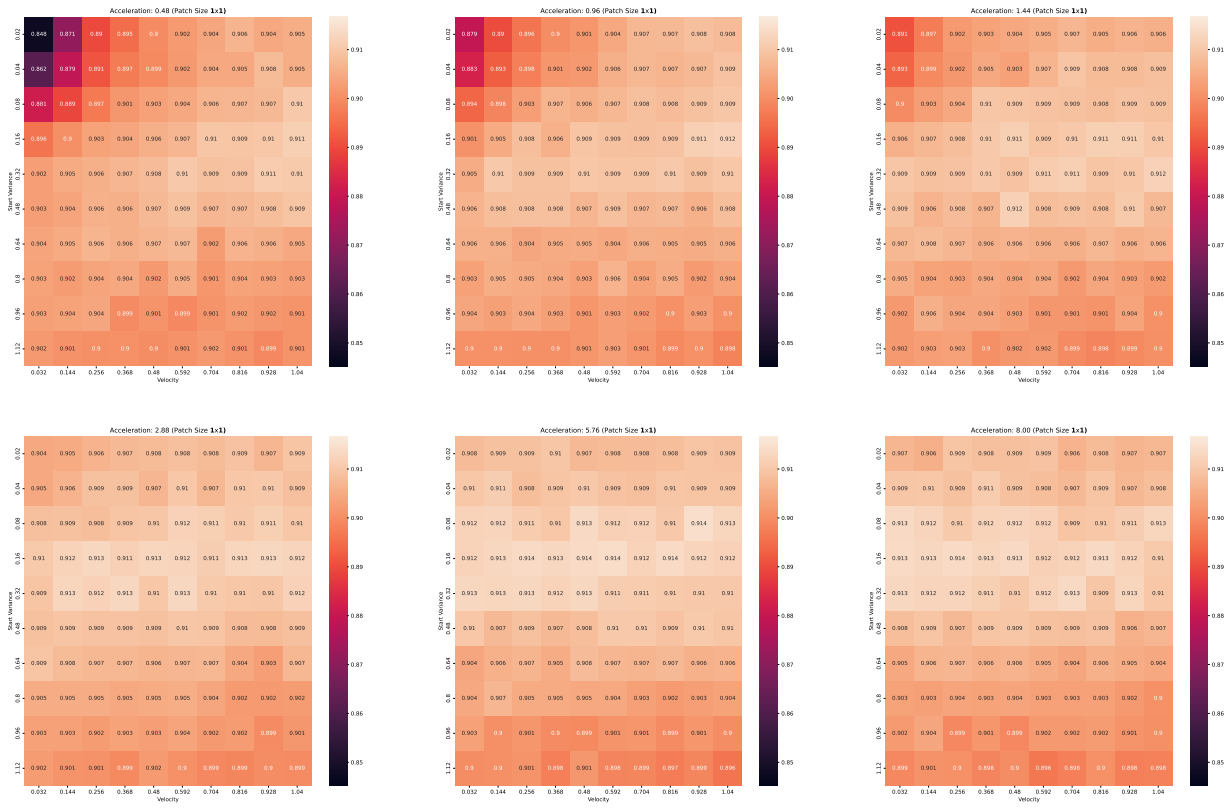


Figure B.2: Grid search over initialization parameters $\sigma_0, v_\sigma, a_\sigma$ for ConvMixer-258/8 with 9×9 *frozen* filters and 1×1 patches trained for 20 epochs on CIFAR-10. Note that the performance of uniform initialization is only ≈88%, i.e., almost all choices result in *some* improvement.

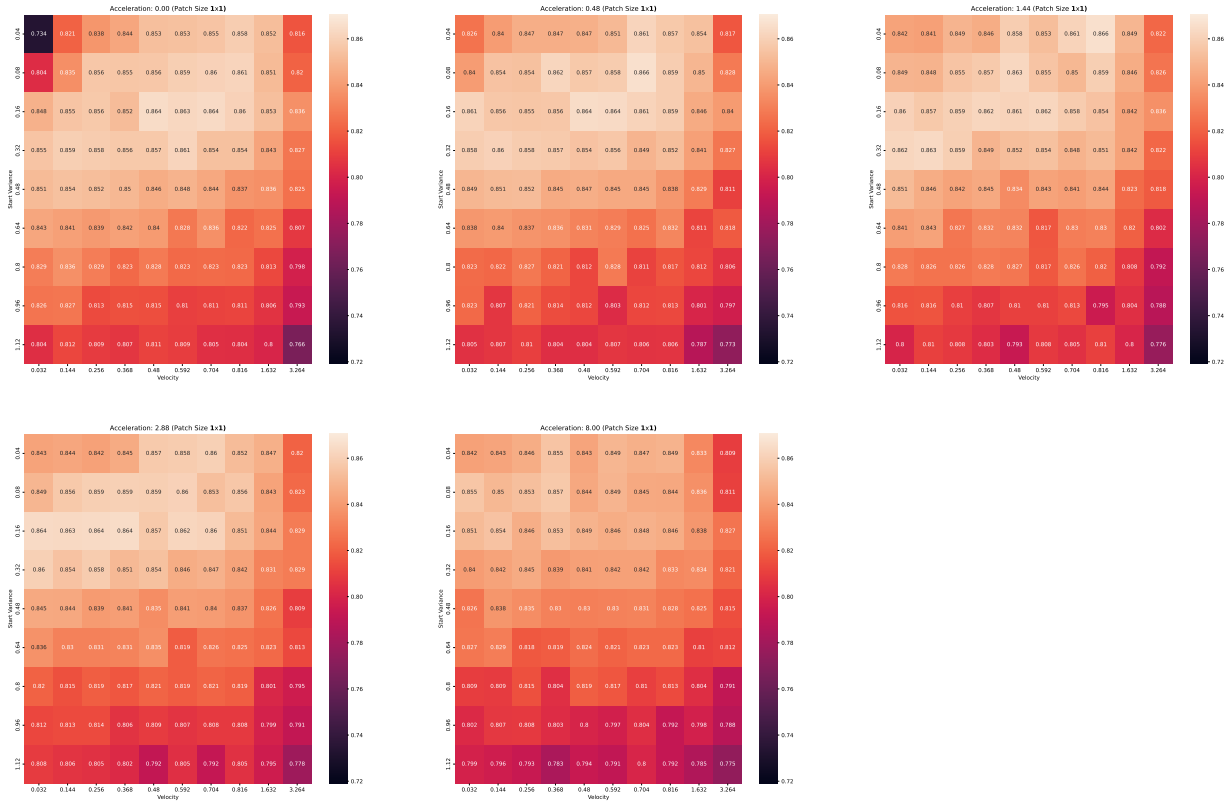


Figure B.3: Grid search over initialiation parameters $\sigma_0, v_\sigma, a_\sigma$ for ConvNeXt-atto on CIFAR-10 with frozen filters and 1×1 patches trained for 20 epochs on CIFAR-10. Note the baseline performance with uniform initialization is around 80%, *i.e.,* compared to ConvMixer there are more potentially disadvantageous parameter combinations.

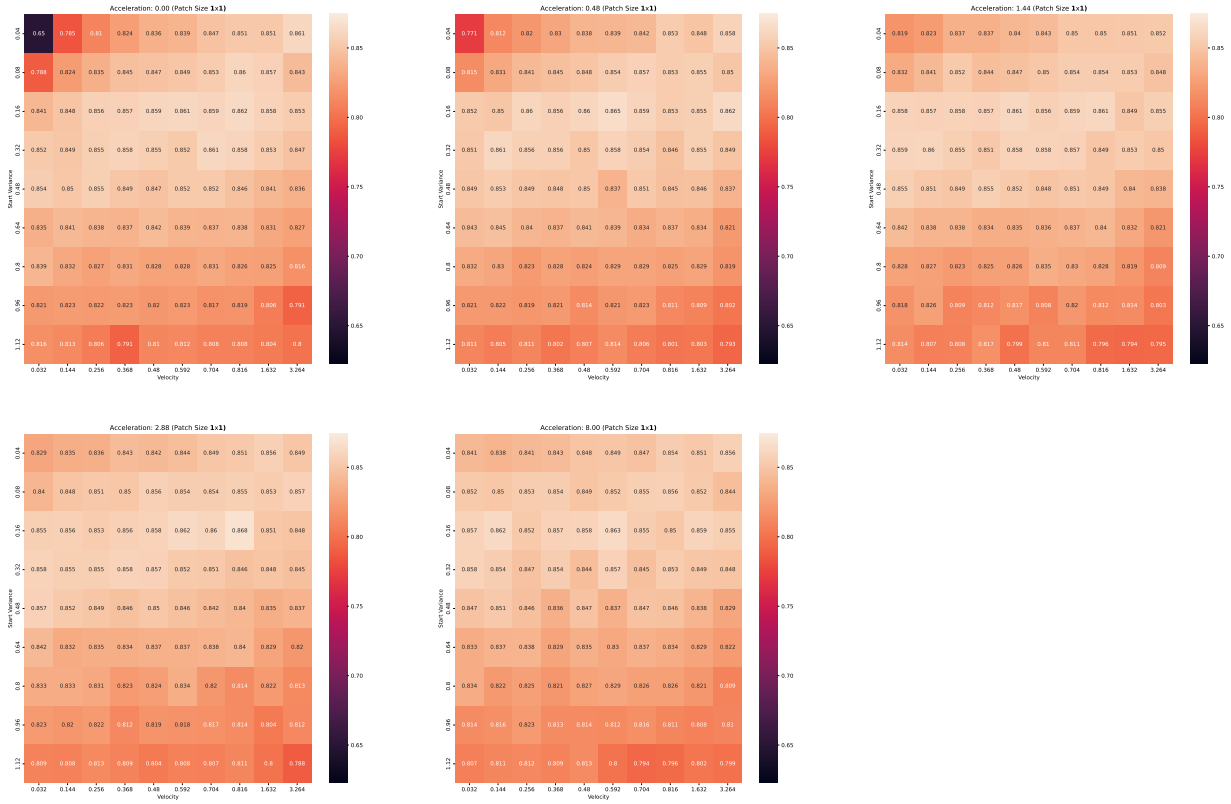


Figure B.4: Grid search over initialiation parameters $\sigma_0, v_\sigma, a_\sigma$ for ConvNeXt-atto on CIFAR-10 with frozen filters and 1×1 patches trained for 20 epochs, using the "sawtooth" variance schedule (see Fig B.5) to account for downsampling layers. While this perhaps shows better robustness to parameter changes than Fig. B.3, the effect could also be due to effectively dividing the parameters by two.

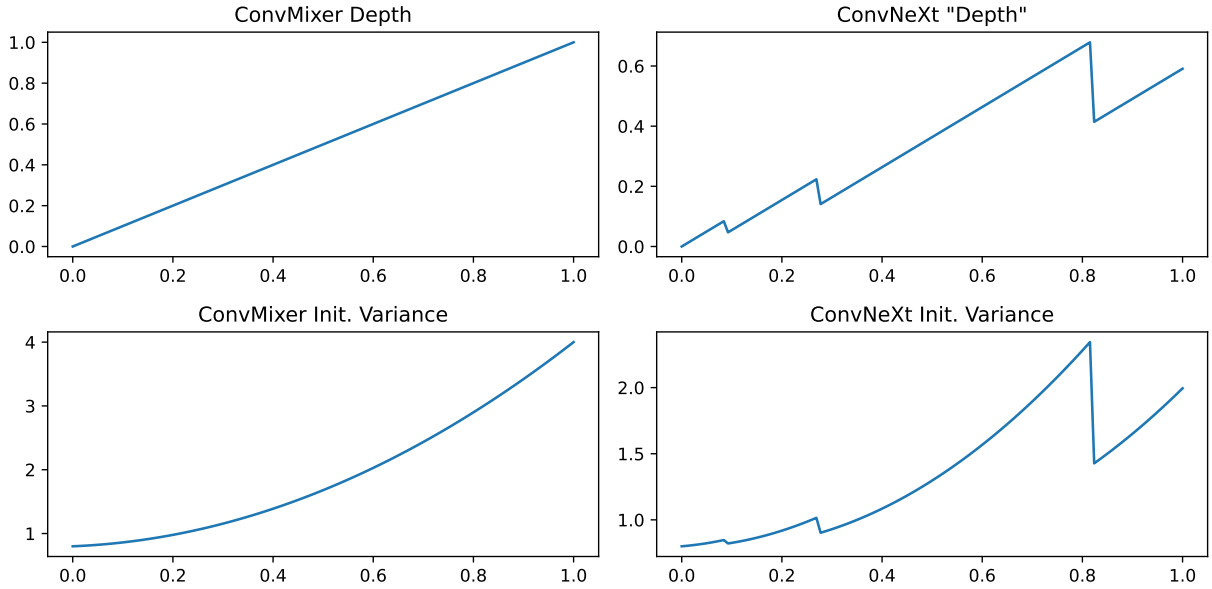


Figure B.5: Proposed stepwise variance schedule for ConvNeXt, *i.e.,* a model including downsampling layers. In our experiments, we saw no advantage to using this scheme.

B.1.2 ImageNet Grid Searches

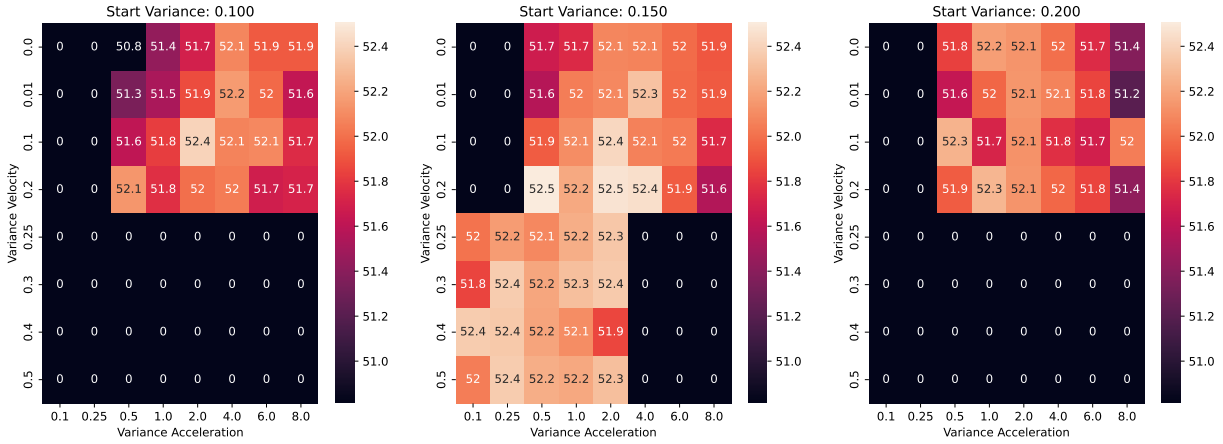


Figure B.6: **Frozen filters:** Grid search over initialization parameters for ConvMixer-512/12 with 14×14 patches and 9×9 filters, 10 epochs. Zeros indicate that the experiment did not run.

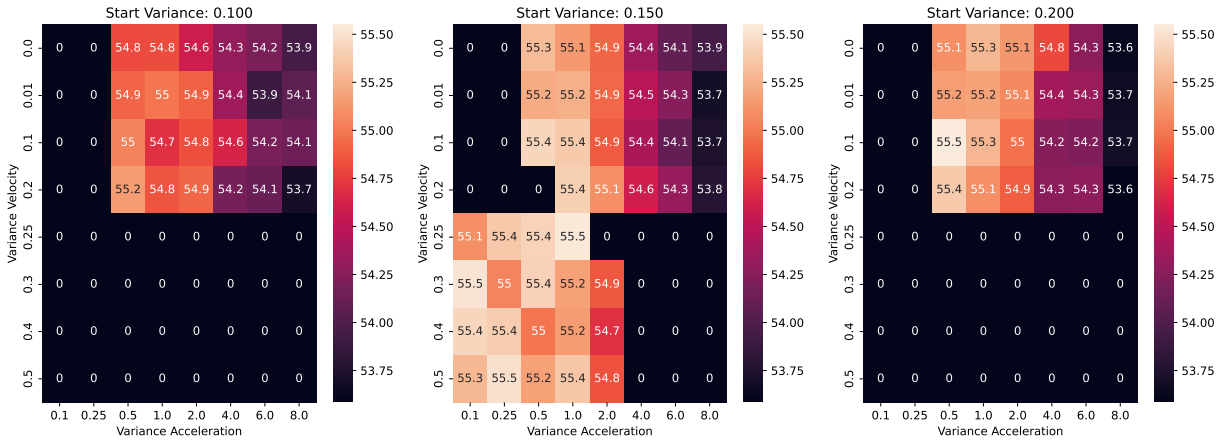


Figure B.7: **Thawed filters:** Grid search over initialization parameters for ConvMixer-512/12 with 14×14 patches and 9×9 filters, 10 epochs.

B.2 Shift Function Definition & Proof

For a given matrix $Z \in \mathbb{R}^{k \times k}$ (*e.g.,* a Gaussian kernel centered at the top left of the filter), we define the Shift operator as follows:

$$\text{Shift}(Z, \Delta x, \Delta y)_{i,j} = Z_{(i+\Delta x) \bmod k, (j+\Delta y) \bmod k}. \tag{B.1}$$

Note that this can be achieved using `np.roll` in NumPy. Then, if

$$[C_{i,j}] = \text{Shift}(Z_\sigma, i, j) \tag{B.2}$$

and the operation $(.)^B$ is defined by

$$\Sigma^B = \Sigma' \iff [\Sigma_{i,j}]_{\ell,m} = \left[\Sigma'_{\ell,m}\right]_{i,j} \text{ for } 1 \leq i, j, \ell, m \leq k, \tag{B.3}$$

then

$$[C_{i,j}]_{\ell,m} = \text{Shift}(Z, i, j)_{\ell,m} = Z_{(i+\ell) \bmod k, (j+m) \bmod k} \tag{B.4}$$

$$= Z_{(\ell+i) \bmod k, (m+j) \bmod k} = \text{Shift}(Z, \ell, m)_{i,j} = [C_{\ell,m}]_{i,j}, \tag{B.5}$$

which shows that $[C_{i,j}]_{\ell,m} = [C_{\ell,m}]_{i,j}$ for all $1 \leq i, j, \ell, m \leq k$, ***i.e.***, C is "block-symmetric", or $C = C^B$. $\square$

B.3 Additional ImageNet Experiments on Mimetic Initialization for Convolutional Layers

ConvMixer-512/12: Patch Size 14, Kernel Size 9	Thawed	Frozen
Uniform init	54.5	47.4
Stats from CM-512/12	55.5	53.4
Stats from CM-64/12	55.2	52.7
Filters transferred from CM-512/12	55.1	54.4
Our init (.15, .3, .5)	55.4	52.2
Our init (.15, .5, .25)	55.5	52.4

Table B.1: ConvMixer performance on ImageNet-1k training with 10 epochs. Our initialization performs comparably to loading covariance matrices from previously-trained models (which were trained for 150 epochs).

ConvMixer-512/12: Patch Size 7, Kernel Size 9	Thawed	Frozen
Uniform init	61.87	56.73
Stats from CM-512/12	62.56	60.79
Stats from CM-64/12	62.72	60.86
Filters transferred from CM-512/12	62.81	61.83
Our init (.15, .3, .5)	62.49	58.94
Our init (.15, .5, .25)	62.59	59.31

Table B.2: ImageNet 10-epoch training

ConvMixer-512/24: Patch Size 14, Kernel Size 9	Thawed	Frozen
Uniform init	50.40	43.00
Stats from CM-512/12	53.03	51.45
Stats from CM-64/12	53.16	51.25
Filters transferred from CM-512/12	52.87	52.12
Our init (.15, .3, .5)	53.80	51.16
Our init (.15, .5, .25)	53.76	50.81

Table B.3: ImageNet 10-epoch training

ConvNeXt-Atto	Thawed	Frozen
Uniform init	31.37	23.63
Stats from the same arch	33.44	40.41
Stats from $1/8^{\text{th}}$-width arch	29.81	31.47
Filters transferred from same arch	31.68	40.48
Our init (.15, .3, .5)	37.64	34.59
Our init (.15, .5, .25)	31.34	34.23
Our init (.15, .25, 1.0)	38.01	33.98

Table B.4: ImageNet 10-epoch training

ConvNeXt-Tiny	Thawed	Frozen
Uniform init	32.51	25.94
Stats from the same arch	42.78	41.54
Stats from $1/8^{\text{th}}$-width arch	44.60	42.86
Filters transferred from same arch	31.01	45.32
Our init (.15, .3, .5)	35.64	35.04
Our init (.15, .5, .25)	40.17	38.91
Our init (.15, .25, 1.0)	40.78	36.62

Table B.5: ImageNet 10-epoch training

ConvNeXt-Atto	Thawed	Frozen
Uniform init	69.96	51.43
Stats from the same arch	68.83	66.71
Stats from $1/8^{\text{th}}$-width arch	68.69	66.31
Filters transferred from same arch	68.01	67.29
Our init (.15, .3, .5)	65.55	63.48
Our init (.15, .5, .25)	67.84	64.52
Our init (.15, .25, 1.0)	68.06	63.43

Table B.6: ImageNet **50-epoch** training

ConvMixer-512/12: Patch Size 14, Kernel Size 9	Thawed	Frozen
Uniform init	67.03	60.47
Stats from the same arch	67.13	65.08
Stats from $1/8^{\text{th}}$-width arch	66.75	64.94
Filters transferred from same arch	67.28	66.11
Our init (.15, .3, .5)	66.12	64.39
Our init (.15, .5, .25)	67.41	64.43
Our init (.15, .25, 1.0)	67.34	64.12

Table B.7: ImageNet **50-epoch** training

ConvMixer-512/24: Patch Size 14, Kernel Size 9	Thawed	Frozen
Uniform init	67.76	62.50
Stats from the same arch	68.92	67.91
Stats from $1/8^{\text{th}}$-width arch	68.78	67.36
Filters transferred from same arch	69.42	68.66
Our init (.15, .3, .5)	69.05	66.20
Our init (.15, .5, .25)	69.60	66.57
Our init (.15, .25, 1.0)	69.52	66.38

Table B.8: ImageNet **50-epoch** training

B.4 Additional CIFAR-10 Tables for Mimetic Initialization for Convolutional Layers

Table B.9: CIFAR-10 results for ConvMixer-256/8 with patch size 2. **Bold** denotes the highest per group, and **blue bold** denotes the second highest.

		Thawed					Frozen				
Filt. Size	# Eps	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init
3	20	$89.69 \pm .20$	$\mathbf{90.08 \pm .39}$	$89.76 \pm .14$	$\mathbf{90.02 \pm .14}$	$89.64 \pm .13$	$88.99 \pm .41$	$\mathbf{89.56 \pm .11}$	$89.54 \pm .40$	$\mathbf{89.94 \pm .08}$	$88.94 \pm .25$
	50	$\mathbf{91.92 \pm .03}$	$\mathbf{92.01 \pm .21}$	$91.83 \pm .02$	$91.69 \pm .18$	$91.63 \pm .07$	$91.12 \pm .22$	$\mathbf{91.42 \pm .15}$	$91.34 \pm .27$	$\mathbf{91.62 \pm .17}$	$91.02 \pm .25$
	200	$\mathbf{93.08 \pm .16}$	$92.94 \pm .26$	$92.92 \pm .19$	$\mathbf{92.97 \pm .18}$	$92.84 \pm .11$	$92.38 \pm .27$	$\mathbf{92.55 \pm .09}$	$92.27 \pm .21$	$\mathbf{92.47 \pm .19}$	$92.09 \pm .13$
7	20	$89.66 \pm .21$	$89.91 \pm .09$	$\mathbf{90.63 \pm .10}$	$90.48 \pm .26$	$\mathbf{90.79 \pm .24}$	$86.73 \pm .27$	$88.86 \pm .36$	$89.36 \pm .09$	$\mathbf{90.00 \pm .27}$	$\mathbf{90.22 \pm .03}$
	50	$91.81 \pm .15$	$91.77 \pm .26$	$91.88 \pm .19$	$\mathbf{91.91 \pm .20}$	$\mathbf{92.48 \pm .08}$	$89.44 \pm .23$	$90.72 \pm .13$	$91.05 \pm .12$	$\mathbf{91.54 \pm .37}$	$\mathbf{92.02 \pm .23}$
	200	$\mathbf{92.86 \pm .15}$	$92.77 \pm .28$	$92.74 \pm .15$	$92.72 \pm .06$	$\mathbf{93.40 \pm .20}$	$90.70 \pm .12$	$91.68 \pm .16$	$91.84 \pm .05$	$\mathbf{92.36 \pm .20}$	$\mathbf{92.83 \pm .24}$
9	20	$89.26 \pm .35$	$89.70 \pm .06$	$\mathbf{90.22 \pm .36}$	$90.14 \pm .06$	$\mathbf{90.85 \pm .14}$	$84.97 \pm .03$	$88.18 \pm .12$	$\mathbf{89.67 \pm .07}$	$89.50 \pm .37$	$\mathbf{90.56 \pm .09}$
	50	$91.74 \pm .12$	$91.63 \pm .15$	$\mathbf{92.11 \pm .18}$	$91.79 \pm .18$	$\mathbf{92.54 \pm .16}$	$88.25 \pm .39$	$90.22 \pm .09$	$91.01 \pm .12$	$\mathbf{91.23 \pm .13}$	$\mathbf{91.96 \pm .12}$
	200	$92.65 \pm .16$	$92.53 \pm .13$	$\mathbf{92.85 \pm .12}$	$92.62 \pm .26$	$\mathbf{93.21 \pm .09}$	$90.04 \pm .33$	$91.34 \pm .08$	$92.00 \pm .18$	$\mathbf{92.05 \pm .28}$	$\mathbf{93.00 \pm .05}$
15	20	$86.64 \pm .30$	$88.19 \pm .51$	$\mathbf{89.27 \pm .05}$	$89.17 \pm .19$	$\mathbf{90.33 \pm .22}$	$81.99 \pm .21$	$86.31 \pm .15$	$87.52 \pm .23$	$\mathbf{88.39 \pm .28}$	$\mathbf{90.38 \pm .06}$
	50	$89.94 \pm .45$	$90.26 \pm .16$	$\mathbf{90.81 \pm .26}$	$90.79 \pm .24$	$\mathbf{92.17 \pm .23}$	$85.05 \pm .33$	$88.71 \pm .08$	$89.51 \pm .06$	$\mathbf{90.08 \pm .09}$	$\mathbf{92.11 \pm .24}$
	200	$91.79 \pm .23$	$91.60 \pm .21$	$\mathbf{92.01 \pm .19}$	$91.87 \pm .26$	$\mathbf{92.94 \pm .11}$	$87.64 \pm .05$	$89.93 \pm .25$	$90.61 \pm .17$	$\mathbf{90.94 \pm .12}$	$\mathbf{93.02 \pm .09}$

Table B.10: CIFAR-10 results for ConvMixer-256/24 with patch size 2. **Bold** denotes the highest per group, and **blue bold** denotes the second highest.

		THAWED					FROZEN				
Filt. Size	# Eps	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init
3	20	$88.37 \pm .11$	$88.79 \pm .07$	**$88.97 \pm .13$**	**$89.22 \pm .26$**	$88.42 \pm .14$	$87.74 \pm .32$	$88.63 \pm .20$	**$88.80 \pm .12$**	**$88.95 \pm .11$**	$88.05 \pm .09$
	50	$92.38 \pm .15$	$92.30 \pm .24$	**$92.46 \pm .07$**	**$92.56 \pm .19$**	$92.19 \pm .24$	$91.86 \pm .18$	$92.02 \pm .32$	**$92.37 \pm .03$**	**$92.31 \pm .14$**	$92.00 \pm .23$
	200	$94.13 \pm .07$	$94.32 \pm .12$	**$94.41 \pm .11$**	**$94.37 \pm .03$**	$94.16 \pm .20$	$93.71 \pm .15$	$93.91 \pm .13$	**$94.28 \pm .20$**	**$93.95 \pm .23$**	$93.84 \pm .10$
7	20	$88.49 \pm .46$	$89.08 \pm .15$	**$89.90 \pm .14$**	$89.81 \pm .16$	**$90.28 \pm .18$**	$85.81 \pm .05$	$87.98 \pm .08$	$89.19 \pm .12$	**$89.34 \pm .40$**	**$90.09 \pm .22$**
	50	$91.90 \pm .17$	$91.73 \pm .26$	**$92.39 \pm .17$**	$92.25 \pm .12$	**$93.15 \pm .08$**	$89.94 \pm .17$	$90.86 \pm .07$	$91.56 \pm .09$	**$91.91 \pm .08$**	**$92.80 \pm .27$**
	200	$93.57 \pm .08$	$93.43 \pm .21$	**$93.71 \pm .20$**	$93.62 \pm .19$	**$94.44 \pm .26$**	$91.78 \pm .22$	$92.78 \pm .16$	**$93.24 \pm .25$**	$93.00 \pm .25$	**$94.03 \pm .23$**
9	20	$87.99 \pm .41$	$88.25 \pm .13$	**$89.44 \pm .03$**	$89.42 \pm .26$	**$90.54 \pm .15$**	$83.42 \pm .33$	$87.38 \pm .18$	$88.46 \pm .45$	**$88.90 \pm .52$**	**$90.31 \pm .12$**
	50	$91.06 \pm .11$	$91.36 \pm .17$	**$91.87 \pm .11$**	$91.69 \pm .06$	**$93.03 \pm .10$**	$88.38 \pm .08$	$90.25 \pm .34$	**$91.12 \pm .16$**	$91.04 \pm .18$	**$92.78 \pm .27$**
	200	$93.12 \pm .37$	$93.08 \pm .17$	**$93.42 \pm .21$**	$93.16 \pm .21$	**$94.12 \pm .18$**	$90.86 \pm .07$	$91.86 \pm .28$	**$92.60 \pm .09$**	$92.53 \pm .15$	**$94.03 \pm .09$**
15	20	$84.95 \pm .50$	$85.80 \pm .48$	$86.67 \pm .29$	**$87.69 \pm .57$**	**$90.08 \pm .23$**	$80.72 \pm .20$	$84.27 \pm .41$	$85.70 \pm .25$	**$86.75 \pm .52$**	**$90.03 \pm .06$**
	50	$89.74 \pm .11$	$89.68 \pm .15$	$90.10 \pm .12$	**$90.22 \pm .27$**	**$92.30 \pm .16$**	$85.10 \pm .27$	$88.05 \pm .11$	$88.78 \pm .33$	**$89.72 \pm .21$**	**$92.93 \pm .24$**
	200	$92.03 \pm .10$	$92.02 \pm .23$	**$92.22 \pm .18$**	$92.20 \pm .03$	**$93.66 \pm .33$**	$88.18 \pm .16$	$90.19 \pm .32$	$90.67 \pm .18$	**$91.19 \pm .20$**	**$94.16 \pm .12$**

Table B.11: CIFAR-10 results for ConvMixer-256/8 with patch size 1. **Bold** denotes the highest per group, and **blue bold** denotes the second highest.

		THAWED					FROZEN				
Filt. Size	# Eps	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init
3	20	$90.41 \pm .11$	$90.60 \pm .31$	$\mathbf{90.78 \pm .22}$	$\mathbf{90.66 \pm .18}$	$89.84 \pm .27$	$89.39 \pm .09$	$\mathbf{90.37 \pm .06}$	$90.14 \pm .03$	$\mathbf{90.64 \pm .17}$	$89.07 \pm .21$
	50	$91.89 \pm .06$	$92.00 \pm .23$	$\mathbf{92.09 \pm .18}$	$\mathbf{92.05 \pm .20}$	$91.49 \pm .10$	$90.97 \pm .19$	$\mathbf{91.87 \pm .09}$	$91.65 \pm .10$	$\mathbf{91.90 \pm .26}$	$90.73 \pm .26$
	200	$92.16 \pm .19$	$92.38 \pm .12$	$\mathbf{92.58 \pm .09}$	$\mathbf{92.54 \pm .18}$	$91.85 \pm .15$	$91.71 \pm .20$	$\mathbf{92.12 \pm .08}$	$\mathbf{92.05 \pm .36}$	$91.98 \pm .16$	$91.47 \pm .14$
7	20	$91.70 \pm .16$	$91.94 \pm .08$	$\mathbf{92.02 \pm .05}$	$\mathbf{92.37 \pm .07}$	$91.80 \pm .23$	$89.84 \pm .04$	$90.78 \pm .27$	$\mathbf{91.30 \pm .11}$	$\mathbf{91.92 \pm .03}$	$91.08 \pm .15$
	50	$93.30 \pm .15$	$93.00 \pm .25$	$93.31 \pm .10$	$\mathbf{93.42 \pm .07}$	$\mathbf{93.33 \pm .29}$	$91.51 \pm .20$	$92.21 \pm .06$	$\mathbf{92.68 \pm .07}$	$\mathbf{93.01 \pm .02}$	$92.31 \pm .32$
	200	$93.67 \pm .13$	$93.59 \pm .08$	$93.68 \pm .12$	$\mathbf{93.81 \pm .23}$	$\mathbf{93.83 \pm .18}$	$92.48 \pm .16$	$92.89 \pm .18$	$92.98 \pm .05$	$\mathbf{93.38 \pm .04}$	$\mathbf{93.16 \pm .20}$
9	20	$91.92 \pm .16$	$91.74 \pm .25$	$92.07 \pm .19$	$\mathbf{92.21 \pm .17}$	$\mathbf{92.37 \pm .19}$	$89.45 \pm .02$	$90.41 \pm .24$	$91.25 \pm .16$	$\mathbf{91.85 \pm .03}$	$\mathbf{91.35 \pm .05}$
	50	$93.25 \pm .23$	$92.92 \pm .05$	$93.22 \pm .16$	$\mathbf{93.37 \pm .01}$	$\mathbf{93.45 \pm .19}$	$91.13 \pm .13$	$91.93 \pm .17$	$\mathbf{92.52 \pm .03}$	$\mathbf{92.85 \pm .16}$	$92.45 \pm .27$
	200	$93.74 \pm .35$	$93.60 \pm .16$	$93.68 \pm .14$	$\mathbf{93.83 \pm .03}$	$\mathbf{94.12 \pm .24}$	$91.96 \pm .07$	$92.67 \pm .04$	$93.08 \pm .17$	$\mathbf{93.38 \pm .01}$	$\mathbf{93.56 \pm .14}$
15	20	$90.65 \pm .23$	$91.00 \pm .05$	$\mathbf{91.91 \pm .17}$	$91.70 \pm .28$	$\mathbf{92.29 \pm .21}$	$86.64 \pm .14$	$89.09 \pm .32$	$90.70 \pm .07$	$\mathbf{91.32 \pm .10}$	$\mathbf{91.53 \pm .12}$
	50	$92.76 \pm .06$	$92.54 \pm .10$	$\mathbf{92.99 \pm .13}$	$92.95 \pm .10$	$\mathbf{93.42 \pm .07}$	$89.20 \pm .17$	$90.65 \pm .15$	$91.72 \pm .23$	$\mathbf{92.35 \pm .13}$	$\mathbf{92.73 \pm .07}$
	200	$93.55 \pm .16$	$93.22 \pm .16$	$93.57 \pm .03$	$\mathbf{93.59 \pm .02}$	$\mathbf{94.13 \pm .13}$	$90.20 \pm .22$	$91.83 \pm .14$	$92.79 \pm .17$	$\mathbf{93.01 \pm .27}$	$\mathbf{93.47 \pm .05}$

Table B.12: CIFAR-10 results for ConvNeXt-atto with patch size 1. **Bold** denotes the highest per group, and **blue bold** denotes the second highest.

		THAWED					FROZEN				
Filt. Size	# Eps	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init
7	20	$81.46 \pm .51$	$85.00 \pm .21$	$84.71 \pm .25$	$\mathbf{86.18 \pm .59}$	$\mathbf{86.44 \pm .78}$	$80.45 \pm .88$	$83.91 \pm .13$	$84.06 \pm .19$	$\mathbf{86.12 \pm .57}$	$\mathbf{84.71 \pm .08}$
	50	$87.55 \pm .46$	$88.38 \pm .10$	$87.92 \pm .27$	$\mathbf{89.07 \pm .75}$	$\mathbf{90.54 \pm .16}$	$85.18 \pm .39$	$87.81 \pm .10$	$86.92 \pm .37$	$\mathbf{89.32 \pm .21}$	$\mathbf{90.13 \pm .14}$
	200	$90.47 \pm .68$	$90.77 \pm .46$	$90.69 \pm .11$	$\mathbf{91.05 \pm .40}$	$\mathbf{92.03 \pm .35}$	$87.10 \pm .08$	$89.70 \pm .21$	$89.08 \pm .24$	$\mathbf{90.83 \pm .21}$	$\mathbf{92.27 \pm .22}$

Table B.13: CIFAR-10 results for ConvNeXt-atto with patch size 2. **Bold** denotes the highest per group, and **blue bold** denotes the second highest.

		THAWED					FROZEN				
Filt. Size	# Eps	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init	Uniform	Cov. transfer	Cov. transfer ($\frac{1}{8}$ width)	Direct transfer	Our init
7	20	$72.96 \pm .70$	$\mathbf{82.36 \pm .32}$	$81.29 \pm .78$	$\mathbf{83.42 \pm .47}$	$81.13 \pm .54$	$72.18 \pm .30$	$\mathbf{80.82 \pm .17}$	$79.03 \pm .07$	$\mathbf{83.61 \pm .23}$	$79.36 \pm .07$
	50	$83.71 \pm .60$	$86.21 \pm .21$	$\mathbf{86.32 \pm .16}$	$\mathbf{86.28 \pm .20}$	$85.97 \pm .24$	$77.92 \pm .34$	$85.16 \pm .45$	$84.83 \pm .09$	$\mathbf{85.93 \pm .20}$	$\mathbf{85.45 \pm .20}$
	200	$86.28 \pm .02$	$87.40 \pm .27$	$\mathbf{88.14 \pm .18}$	$87.02 \pm .10$	$\mathbf{88.20 \pm .52}$	$79.82 \pm .38$	$86.25 \pm .10$	$\mathbf{86.60 \pm .02}$	$86.59 \pm .33$	$\mathbf{87.69 \pm .28}$

Bibliography

A. Ali, I. Zimerman, and L. Wolf. The hidden attention of mamba models. *arXiv preprint arXiv:2403.01590*, 2024. 3.3.1

I. Amos, J. Berant, and A. Gupta. Never train from scratch: Fair comparison of long-sequence models requires data-driven priors. *arXiv preprint arXiv:2310.02980*, 2023. 1.1, 3.3.5

S. Arora, S. Eyuboglu, M. Zhang, A. Timalsina, S. Alberti, D. Zinsley, J. Zou, A. Rudra, and C. Ré. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*, 2024. 3.3, 3.3

S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018. 3.2.6

I. Bello. Lambdanetworks: Modeling long-range interactions without attention. *arXiv preprint arXiv:2102.08602*, 2021. 2.1.3

I. Bello, B. Zoph, A. Vaswani, J. Shlens, and Q. V. Le. Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3286–3295, 2019. 2.1.3

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. 1.1

Y.-H. Cao, H. Yu, and J. Wu. Training vision transformers with only 2040 images. *arXiv preprint arXiv:2201.10728*, 2022. 3.2.1

G. Cazenavette, J. Julin, and S. Lucey. Rethinking the role of spatial mixing. 3.1.1, 3.1.3

S. Chen, E. Xie, C. Ge, D. Liang, and P. Luo. Cyclemlp: A mlp-like architecture for dense prediction. *arXiv preprint arXiv:2107.10224*, 2021. 2.1.3

Y. Chen, J. Liu, X. Qi, X. Zhang, J. Sun, and J. Jia. Scaling up kernels in 3d cnns. *arXiv preprint arXiv:2206.10555*, 2022. 3.1

J.-B. Cordonnier, A. Loukas, and M. Jaggi. On the relationship between self-attention and convolutional layers. *arXiv preprint arXiv:1911.03584*, 2019. 2.1.3, 3.2.1, 3.2.4

E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020. 2.1.2, 3.1.1

Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019. 3.2.6

Z. Dai, H. Liu, Q. V. Le, and M. Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34:3965–3977, 2021. 2.1.3, 3.2.1

T. Dao and A. Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024. 3.3, 3.3, 3.3.1, 3.3.1, 3.3.1, 3.3.6

S. d'Ascoli, H. Touvron, M. Leavitt, A. Morcos, G. Biroli, and L. Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. *arXiv preprint arXiv:2103.10697*, 2021. 2.1.3

X. Ding, X. Zhang, J. Han, and G. Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11963–11975, 2022. 3.1

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. (document), 1.1, 2.1, 2.2, 3.2

S. d'Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *International Conference on Machine Learning*, pages 2286–2296. PMLR, 2021. 3.2.1

S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018. 3.3.1

H. Gani, M. Naseer, and M. Yaqub. How to train vision transformer on small-scale datasets? *arXiv preprint arXiv:2210.07240*, 2022. 3.2.1, 3.2.4

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010. 1.1, 3.1, 3.4

A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. 3.3

A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020. 3.3

A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. 3.3, 3.3.2

A. Gu, K. Goel, A. Gupta, and C. Ré. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022. 3.3

J. Guo, K. Han, H. Wu, C. Xu, Y. Tang, C. Xu, and Y. Wang. Cmt: Convolutional neural networks meet vision transformers. *arXiv preprint arXiv:2107.06263*, 2021. 2.1.3

A. Gupta, A. Gu, and J. Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022. 3.3

Q. Han, Z. Fan, Q. Dai, L. Sun, M.-M. Cheng, J. Liu, and J. Wang. On the connection between local attention and dynamic depth-wise convolution. In *International Conference on Learning Representations*, 2021. 3.1

A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021. 3.2.1, 3.2.4

B. He and T. Hofmann. Simplifying transformer blocks. *arXiv preprint arXiv:2311.01906*, 2023. 3.1.4

B. He, J. Martens, G. Zhang, A. Botev, A. Brock, S. L. Smith, and Y. W. Teh. Deep transformers without shortcuts: Modifying self-attention for faithful signal propagation. *arXiv preprint arXiv:2302.10322*, 2023. 1.1, 3.2.1, 3.2.5

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 1.1, 1.1, 3.1, 3.1.1, 3.4

D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 2.1.2

Q. Hou, Z. Jiang, L. Yuan, M.-M. Cheng, S. Yan, and J. Feng. Vision permutator: A permutable mlp-like architecture for visual recognition, 2021. 2.1.3

X. S. Huang, F. Perez, J. Ba, and M. Volkovs. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pages 4475–4483. PMLR, 2020. 3.2.1, 3.2.4

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL `https://arxiv.org/abs/1502.03167`. 1.1, 3.4

S. Jelassi, D. Brandfonbrener, S. M. Kakade, and E. Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024. 1.1, 3.3, 3.3, 3.3.2, 3.3.3, 3.3.4, 3.3.4, 3.3.6

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017. URL `https://arxiv.org/abs/1412.6980`. 1.1, 3.4

A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby. Big transfer (bit): General visual representation learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 491–507. Springer, 2020. 1.1, 3.4

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 3.1, 3.4

D. Kunin, A. Raventós, C. Dominé, F. Chen, D. Klindt, A. Saxe, and S. Ganguli. Get rich quick: exact solutions reveal how unbalanced initializations promote rapid feature learning. *arXiv preprint arXiv:2406.06158*, 2024. 3.4

G. Leclerc, A. Ilyas, L. Engstrom, S. M. Park, H. Salman, and A. Madry. ffcv. `https://github.com/libffcv/ffcv/`, 2022. commit f253865. 3.1.1

S. H. Lee, S. Lee, and B. C. Song. Vision transformer for small-size datasets. *arXiv preprint*

arXiv:2112.13492, 2021. 3.2.1, 3.2.4

J. Lei Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *ArXiv e-prints*, pages arXiv–1607, 2016. 3.4

H. Liu, Z. Dai, D. R. So, and Q. V. Le. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021a. 2.1.3

S. Liu, T. Chen, X. Chen, X. Chen, Q. Xiao, B. Wu, M. Pechenizkiy, D. Mocanu, and Z. Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022a. 3.1

Y. Liu, E. Sangineto, W. Bi, N. Sebe, B. Lepri, and M. Nadai. Efficient training of visual transformers with small datasets. *Advances in Neural Information Processing Systems*, 34: 23818–23830, 2021b. 3.2.1, 3.2.4

Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021c. (document), 2.2

Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022b. 3.1, 3.1.1, 3.4.1

I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. 2018. 2.1.2

J. Martens, A. Ballard, G. Desjardins, G. Swirszcz, V. Dalibard, J. Sohl-Dickstein, and S. S. Schoenholz. Rapid training of deep neural networks without skip connections or normalization layers using deep kernel shaping. *arXiv preprint arXiv:2110.01765*, 2021. 1.1, 3.1

L. Melas-Kyriazi. Do you even need attention? a stack of feed-forward layers does surprisingly well on imagenet, 2021. 2.1.3

B. Neyshabur, H. Sedghi, and C. Zhang. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523, 2020. 1.1, 4.1

C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, S. Johnston, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html. 3.3.1

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. (document), A.1

M. Poli, A. W. Thomas, E. Nguyen, P. Ponnusamy, B. Deiseroth, K. Kersting, T. Suzuki, B. Hie, S. Ermon, C. Ré, et al. Mechanistic design and scaling of hybrid architectures. *arXiv preprint arXiv:2403.17844*, 2024. 3.3.6

M. Raghu, C. Zhang, J. Kleinberg, and S. Bengio. Transfusion: Understanding transfer learning

for medical imaging. *Advances in neural information processing systems*, 32, 2019. 1.1

P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909*, 2019. 2.1.3

D. W. Romero, R.-J. Bruintjes, J. M. Tomczak, E. J. Bekkers, M. Hoogendoorn, and J. C. van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. *arXiv preprint arXiv:2110.08059*, 2021. 3.1

M. Sandler, J. Baccash, A. Zhmoginov, and A. Howard. Non-discriminative data or weak model? on the relative importance of data and model resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. (document), 2.1.2, 2.1.3, 2.2, 2.1.6

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013. 3.1

J. T. Smith, A. Warrington, and S. Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=Ai8Hw3AXqks`. 3.3

I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, D. Keysers, J. Uszkoreit, M. Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021. (document), 2.1, 2.1.1, 2.1.3, 2.2, 2.1.6

H. Touvron, P. Bojanowski, M. Caron, M. Cord, A. El-Nouby, E. Grave, A. Joulin, G. Synnaeve, J. Verbeek, and H. Jégou. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021a. (document), 2.1.3, 2.2, 2.1.4, 2.1.5, 2.1.6, 3.1.4

H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021b. (document), 2.1.2, 2.1.4, 2.2, 3.2.4, 3.4.2

H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 32–42, 2021c. 2.1.4, 3.2.1, 3.2.5

A. Trockman and J. Z. Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022. 3.1, 3.1.1, 3.1.3, 3.2.4, 3.4.1

A. Trockman and J. Z. Kolter. Mimetic initialization of self-attention layers. In *International Conference on Machine Learning*, pages 34456–34468. PMLR, 2023. 3.1.4, 3.3, 3.3.2, 3.4, 3.4, 3.4.1, 3.4.1, 3.4.2

A. Trockman, D. Willmott, and J. Z. Kolter. Understanding the covariance structure of convolutional filters. *arXiv preprint arXiv:2210.03651*, 2022. 3.2.1, 3.3, 3.4, 3.4.1, 3.4.2, 3.4.2

A. Trockman, H. Harutyunyan, J. Z. Kolter, S. Kumar, and S. Bhojanapalli. Mimetic initialization helps state space models learn to recall. *arXiv preprint arXiv:2410.11135*, 2024. 3.4

A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. 2.1.3, 3.3

R. Waleffe, W. Byeon, D. Riach, B. Norick, V. Korthikanti, T. Dao, A. Gu, A. Hatamizadeh, S. Singh, D. Narayanan, et al. An empirical study of mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024. 3.3, 3.3.6

W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021. 2.1.3

Z. Wang, Y. Bai, Y. Zhou, and C. Xie. Can cnns be more robust than transformers? *arXiv preprint arXiv:2206.03452*, 2022. 3.1

R. Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 2.1.2

R. Wightman, H. Touvron, and H. Jégou. Resnet strikes back: An improved training procedure in timm, 2021. (document), 2.1.2, 2.1.4, 2.2, 2.1.7, 3.1.3, 3.2.4

H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22–31, 2021. 3.2.1

L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402. PMLR, 2018. 1.1, 3.1

T. Xiao, M. Singh, E. Mintun, T. Darrell, P. Dollár, and R. Girshick. Early convolutions help transformers see better. *arXiv preprint arXiv:2106.14881*, 2021. 2.1.3

F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 3.1.4

K. Yuan, S. Guo, Z. Liu, A. Zhou, F. Yu, and W. Wu. Incorporating convolution designs into visual transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 579–588, 2021a. 2.1.3, 3.2.1

L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021b. 2.1.3

S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019. 2.1.2

H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 2.1.2

M. Zhang, K. Bhatia, H. Kumbong, and C. Re. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. In *The Twelfth International Conference on Learning Representations*. 3.3.3

Y. Zhang, A. Backurs, S. Bubeck, R. Eldan, S. Gunasekar, and T. Wagner. Unveiling transformers with lego: a synthetic reasoning task. *arXiv preprint arXiv:2206.04301*, 2022. 1.1, 3.1, 3.2, 3.2.1, 3.4

J. Zhao, F. Schäfer, and A. Anandkumar. Zero initialization: Initializing residual networks with only zeros and ones. *arXiv preprint arXiv:2110.12661*, 2021. 3.2.1, 3.2.4

J. Zheng, X. Li, and S. Lucey. Convolutional initialization for data-efficient vision transformers. *arXiv preprint arXiv:2401.12511*, 2024. 3.4

Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13001–13008, 2020. 2.1.2