UNIVERSITY OF CALIFORNIA

Los Angeles

Curricula-Driven Approaches for Efficient Model Training

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical and Computer Engineering

by

Elvis Nunez

2024

ABSTRACT OF THE DISSERTATION

Curricula-Driven Approaches for
Efficient Model Training

by

Elvis Nunez
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2024
Professor Stefano Soatto, Chair

As deep learning models and datasets continue to scale up, the financial and environmental costs associated with training these models have grown considerably over the past decade. While hardware accelerators—such as GPUs and TPUs—have made strides in efficiency, these gains are insufficient to offset the training demands of contemporary models. To move beyond the limits of hardware improvements alone, we pursue a complementary approach of developing more algorithmically-efficient training methods. In particular, we explore dynamically modulating the capacity of neural networks during training in service of improving training efficiency. Traditional training methods often assume a fixed model capacity and dataset complexity during the training process. However, we demonstrate that utilizing the model's full capacity for the entire duration of training is unnecessary and can lead to excessive resource consumption, overfitting, and slow convergence. To this end, we explore several approaches for dynamically modulating both model capacity and data complexity during training. For the model capacity, we consider techniques like dynamic model compression and modifications to the underlying architecture. For data complexity, we

consider dynamic adjustments to the input resolution. Central to our approaches are distinct curricula, each governing how the model's capacity and data complexity evolve throughout training. By modulating capacity in alignment with the model's learning stage, we reduce unnecessary computation while maintaining the model's performance. We demonstrate the effectiveness of this modulation across a wide range of vision and language tasks, as well as for various architectures.

The dissertation of Elvis Nunez is approved.

Jonathan Chau-Yan Kao

Cho-Jui Hsieh

Aditya Grover

Stefano Soatto, Committee Chair

University of California, Los Angeles

2024

To my parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

My interest in pursuing a PhD began during my undergraduate studies at Brown University. While there, I had the fortune of taking APMA1655 with Caroline Klivans, which proved to be a transformative experience. It was here that I became enthralled by how mathematical principles could elegantly describe and predict behavior of stochastic behavior, turning uncertainty into something quantifiable and understandable. Soon after, I took Nicolas Garcia-Trillos's courses on statistical learning, and Jerome Darbon's courses on convex analysis. The interplay between these disciplines sparked my interest in machine learning and set me on the path to a PhD. I am deeply grateful to Caroline, Nicolas, and Jerome for nurturing my curiosity and guiding me into these fields.

Following my time at Brown, I had the pleasure of meeting and working with Shantanu Joshi at UCLA. His kind and supportive mentorship has played an essential role throughout my PhD journey. Thanks to Shantanu, I was able to meet my advisor, Stefano Soatto, who welcomed me into his research group. Stefano's guidance and support have helped me navigate challenges, deepen my understanding of complex problems, and refine my research approach. His influence has been significant in my development, fostering my growth not only as a machine learning researcher but also in my personal resilience and confidence.

During my PhD, I had the pleasure of interning at both Apple and Amazon. At Apple, I learned a lot from my collaborators Max Horton, Sachin Mehta, Mohammad Rastegari, Ali Farhadi, Anish Prabhu, Yanzi Jin, Anurag Ranjan, Thomas Merth, and Mehrdad Farajtabar. At AWS, I had the privilege of being mentored by Luca Zancato, and I am grateful for the guidance I received from my collaborators Ben Bowman, Michael Kleinman, Aditya Golatkar, and Aditya Chattopadhyay. The support I received through these experiences has helped me develop into the researcher I am today and I will always be grateful for it.

In the development of this dissertation, insightful questions and comments from my committee, Jonathan Kao, Cho-Jui Hsieh, and Aditya Grover, have helped me refine and

strengthen this work. I am very grateful for their kind and compassionate guidance throughout this journey.

My PhD was supported by two fellowships: an NSF NRT fellowship (2020-2021) and an NSF GRFP fellowship (2021-2024). This funding allowed me to pursue my curiosities, for which I am deeply grateful. Moreover, I thank Andrea Bertozzi for helping me navigate the landscape of academic funding; her dedication to students has inspired me on both a professional and personal level.

Finally, I would like to thank my parents, Carlos and Margarita, and my brother, Galileo. Without their unwavering support, I would not be where I am today. Thank you to Sophie for supporting me during the last and most difficult leg of this journey. I appreciate all of you more than you know.

VITA

2014-2018	Sc.B. in Applied Mathematics - Computer Science, Brown University
2018-2019	PhD Student in Applied Mathematics and Statistics, Johns Hopkins University (left without degree)
2021	Machine Learning Research Intern, Apple
2022	Machine Learning Research Intern, Apple
2023	Machine Learning Research Intern, Apple
2024	Applied Scientist Intern, AWS AI Labs
2020-Present	PhD Student in Electrical and Computer Engineering, University of California, Los Angeles

CHAPTER 1

Introduction

1.1 Deep Learning Advancements and the Need for Efficient Training

The proliferation of deep learning models can be traced back to the AlexNet [60] model from 2012. AlexNet demonstrated that deep convolutional networks (CNNs) could be successfully trained on large-scale datasets such as ImageNet [20] and take advantage of GPU acceleration for faster training. Since then, model design has evolved from basic convolutional networks to more sophisticated architectures incorporating depth [96], modularity [102], and residual connections [38]. In parallel, advancements in the field of Natural Language Processing innovated on recurrent neural networks [91, 119] by developing sequence-to-sequence (seq2seq) models [101] built on top of long short-term memory (LSTM) modules [42]. These seq2seq models relied on encoding entire input sequences into a single fixed-size vector which the decoder would then use to generate the output sequence, making it challenging for the model to retain information from long sequences. The Bahdanau Attention Mechanism [7] addressed these issues by allowing the decoder to access the entire sequence of encoder outputs rather than just a single vector. The Self-Attention mechanism [111] is more general than Bahdanau Attention and operates on the input sequence itself, computing relationships between all tokens in parallel; this makes Self-Attention highly scalable. The coupling of Self-Attention layers with fully-connected layers and residual connections led to the development of the Transformer model, which facilitated a convergence between the Computer Vision and Natural Language Processing fields.

Since the introduction of the Transformer architecture in 2017, it has become the foundation for most modern neural networks. In vision, Vision Transformer models [21, 107] have been developed for object recognition and segmentation tasks [11, 12, 66]. Moreover, the coupling of language and vision modalities has led to innovations in generative modeling , such as diffusion models [85, 89]. Large Language Models (LLMs) have become a primary research focus as their ability to parse unstructured data such as text, images, and audio enables them to generate coherent insights and predictions across diverse tasks. This capability has proven especially impactful in natural language processing, where LLMs such as GPT [2], Claude [5], and Llama [22], have revolutionized applications ranging from machine translation, to question answering and content generation.

Scaling laws of LLMs [57] dictate that increasing compute leads to stronger-performing models. This increase in compute manifests itself in increasingly larger models with more parameters and larger training datasets. This trend is driving the increasing strain on both economic and ecological resources as model sizes expand and more compute-intensive tasks are undertaken. The cost of training OpenAI's GPT-4, for example, is estimated to be $40M, while Google's Gemini Ultra is estimated to be $30M [15]. Moreover, the CO_2 emissions associated with training a simple Transformer model having 10^8 parameters—far fewer than contemporary LLMs—are projected to be greater than those of several cars over their entire lifetime (when accounting for hyperparameter tuning) [98]. These realizations underscore the need for more efficient training of deep learning models.

A common critique of research in efficient training of deep learning models is that training costs are incurred only once and are then amortized over the model's deployment cycle. However, this perspective overlooks the iterative nature of model development. In practice, a model undergoes multiple rounds of training and extensive hyperparameter tuning before deployment to identify the optimal configuration. By applying efficient training techniques across these iterative cycles, we can achieve significant cost savings and reduce the environmental footprint of model development. Motivated by this need for sustainable and

scalable model training, our work explores approaches that dynamically modulate training complexity—in service of improving training efficiency—without sacrificing performance.

1.2 Curriculum-Based Training: Dynamic Modulation of Model and Data Complexity for Efficient Model Training

Human learning follows a structured progression and aligns with the natural development of the brain. From infancy to adulthood, educational complexity builds systematically, allowing foundational skills to precede more advanced concepts. This progression is not arbitrary; rather, individuals advance through a highly organized sequence of increasing difficulty. This mechanism that steers the difficulty of each subsequent step is referred to as a "curriculum," and the learning process it underpins is commonly referred to as "curriculum learning" [10]. In the context of machine learning, curriculum learning involves systematically increasing the complexity of tasks presented to a model; such training has been shown to improve generalization.

In our work, we adopt the term "curriculum" to describe a schedule that governs how the model's capacity and the training data's complexity evolve throughout training. By "model capacity," we mean the model's ability to effectively represent and fit its training data, which is largely determined by its architecture and the number of parameters it contains. By "data complexity," we mean the richness or variability of information present in the training data, which can be approximated by attributes such as sample resolution.

Traditional deep learning training methods generally maintain a fixed model architecture throughout training. Moreover, certain characteristics of the training dataset, such as batch size or image resolution (or sequence length for sequence models), are also kept fixed. In pursuit of algorithmically-efficient training methods, we explore the dynamic modulation of the model's architecture and/or training data during training.

A common approach for enhancing model efficiency, particularly during inference, is

network pruning. This technique involves removing redundant parameters from a trained model, reducing its size and computational requirements without significantly compromising accuracy. The standard pruning framework is typically iterative and involves training the model to convergence, pruning it, fine-tuning it to recover from pruning-induced accuracy loss, and then repeating this process until the desired model size and performance trade-off is reached [34]. Recent works [3] reveal that models undergo two phases during training: a memorization and a forgetting phase. During the memorization phase, the network encodes information about the training data in its weights, and then sheds this information during the forgetting phase. In *Timing Matters: Identifying When Neural Networks Are Most Amenable to Pruning* (Chapter 2), we reveal a correlation between the performance of a model pruned at different training stages, and the phase transition between the memorization and forgetting phases. In particular, we show that pruning a model via one-shot magnitude-based pruning near this transition phase yields a higher accuracy model than pruning at the end of training. An analysis on linear networks further elucidates this behavior, suggesting that models are most amenable to pruning after they have learned the most salient features of their training data, but before they begin to overfit. This insight allows us to strategically determine the optimal time to prune a model during training, allowing the remaining training steps to proceed on a compressed model and eliminating the need for training to convergence before pruning.

In *Dynamic Compression During Training for Real-Time Adaptive Inference* (Chapter 3), we tackle both training and inference efficiency. A common assumption when deploying a neural network onto a device is that the device will always have sufficient computational resources available to run the model. However, battery power and competing applications may preclude the device from running the model. In this work, we develop a training framework that produces a model capable of dynamically adapting to the device's resources. We accomplish this by compressing the model—via pruning or quantization—according to the device's available resources. This is done on-device without the need for recalibration

nor fine-tuning. To enable this, we propose a training method based on Neural Network Subspaces [121] that gradually compresses a linear subspace so that one end is optimized for efficiency, while the other end is optimized for accuracy, yielding a strong accuracy-efficiency trade-off. By gradually compressing the model during training, we reduce the cost of training while simultaneously producing a model amenable to a spectrum of compression levels after deployment.

In *Curriculum-Based Multi-Scale Training* (Chapter 4), we perform a thorough analysis on the effect of modulating the data resolutions during the training of vision models. We show that training with multiple resolutions acts as an effective regularizer, producing networks that are more robust to changes in input scales and adversarial inputs. We introduce a curriculum, MSc-VBSWC, that governs how sample resolutions expand throughout training and simultaneously adjust the batch size accordingly, ensuring full utilization of accelerators. We show that this training method reduces compute by $> 30\%$ while preserving or improving baseline accuracy. To further reduce training compute, we also explore growing the model architecture in tandem with the sample resolutions. Namely, we progressively increase the model's depth and width alongside the sample resolutions, effectively increasing model capacity and data complexity in parallel. We present a Pareto-optimal order for this expansion and demonstrate that this compound model/data scaling yields networks with a higher accuracy than baseline models trained within a similar compute budget.

Training deep learning models on video data presents significant challenges due to the inherent computational cost of processing high-dimensional, temporal information. Modeling such temporal dependencies leads to substantial resource consumption during training, and as the demand for more accurate and robust video models increases, the need for efficient training methods has become increasingly critical. To address this, in *Cross-Attention and Curriculum-Based Masking for Efficient Masked Autoencoders* (Chapter 5), we augment a multi-modal masked autoencoder for audio and video [50] with a diffusion process. Our model, DiffMAViL, leverages this diffusion process to replace self-attention layers with cross-attention

layers so that masked (diffused) tokens only need to attend to visible tokens. We couple this with a masking ratio that progressively decays throughout training. Similar to our MSc-VBSWC curriculum sampler, we dynamically adjust the batch size so that we process more samples when the masking ratio is high (fewer visible patches), and fewer samples when the masking ratio is low (more visible patches). We demonstrate that these changes reduce training FLOPS by $> 30\%$ while maintaining the baseline model's performance.

Transformer-based models have have become ubiquitous across many areas in deep learning. However, the computational complexity of Attention—the primary building block of Transformers—scales quadratically in the sequence length, making them difficult to train on long sequences without access to large-scale computational resources. A recent class of architectures that rivals Transformers is State Space Models (SSMs) [32, 31], whose computational complexity scales linearly in the sequence length. This linear scaling is achieved by maintaining a fixed-size compressed state representation of processed tokens. While SSMs offer improved computational and memory efficiency, their fixed-size state is inherently lossy. Recently, Hybrid SSMs [18, 19, 28, 63] have interleaved Attention layers with SSM layers in an effort to mitigate this "fading" memory. However, due to the quadratic scaling of Attention, training these models on sequences with many tokens remains costly. In *Efficient Adaptation of Hybrid State Space Models to Long Sequences of Tokens* (Chapter 6), we introduce a sparse Attention mechanism, Span-Expanded Attention (SE-Attn), which enables training on long context tasks by processing long sequences in chunks, and retrieving relevant information from previous chunks. Interestingly, we find that dynamically adjusting the chunk sizes during training leads to improved generalization. We show that this Attention mechanism improves performance on long context downstream tasks when used to fine-tune Hybrid SSMs on long sequences. Further, in order to efficiently adapt Hybrid SSMs to leverage this retrieval, we develop a variant of LoRA, HyLoRA, specific to Hybrid models and show that our training framework improves upon existing methods.

Our work explores a variety of methods for improving the efficiency of model training. The

common denominator across these methods is the modulation of model capacity and/or data complexity during training. Ultimately, our work aims to contribute to the ongoing effort to make deep learning more accessible and sustainable. We demonstrate that model capacity and data complexity need not remain static throughout training, and that dynamically modulating these factors offers a means to reduce training costs and improve training efficiency without compromising performance.

CHAPTER 2

Timing Matters: Identifying When Neural Networks Are Most Amenable to Pruning

The pruning of neural networks involves removing unnecessary weights from a trained network to create a compressed model with a reduced memory and compute footprint. "Unstructured" pruning is a technique used to remove individual weights according to some criteria, without regard to their arrangement or position within the model's structure. The most common criterion for pruning is magnitude pruning [34], where the weights with the smallest magnitudes are pruned; however, more sophisticated gradient-based methods have also been explored [62, 35]. Typically, pruning is applied after training in an iterative process, where a network is repeatedly pruned and fine-tuned to maintain performance.

Since pruning effectively reduces the model's capacity to fit its data, in this chapter we investigate the question: *How long during training does a model need to maintain a high capacity?* By "capacity," we mean the size of the model, i.e., its number of parameters, which we use as a proxy for the size of the hypothesis class that can be learned by the learning procedure. We are particularly interested in determining if there exists a point during training when a model's capacity can be reduced, allowing the remainder of training to be conducted on a compressed model, thereby improving efficiency without compromising performance.

We showcase a phenomenon in which an "early pruning period" occurs—a period during training where a model becomes amenable to pruning, prior to convergence. Specifically, we find that pruning the model during this period leads to better performance compared to pruning either earlier or later. To help understand this behavior, we draw inspiration from

recent work showing that model training undergoes two phases—the "memorization" and "forgetting" phases [3]—and we show that the early pruning period often correlates with these two phases, suggesting that a large model capacity is only needed for the transient period of training, after which the model can be effectively compressed. Further, we investigate this behavior in the case of deep linear networks and conclude that pruning should occur after the model has learned the most salient features of its training data, and before it begins to overfit.

2.1 Identifying Early Pruning Periods

We consider a neural network $f(x;\theta)$ parameterized by $\theta \in \mathbb{R}^d$ that takes as input x. We train f for a total of T epochs and denote by $\theta_{0\to k}$, $k \in \{1, 2, \ldots, T\}$ the parameters after training for k epochs when starting from random initialization θ_0. In this chapter, we consider the discriminative task of classification. We denote by $\text{Acc}\left(f(x;\theta_{0\to k}), D\right)$ the test accuracy of the model on a held-out dataset D. We are interested in compressing $\theta_{0\to k}$ by pruning some of its weights, i.e., setting some of its values to 0. We denote by $\text{Prune}\left(\theta_{0\to k}, r\right)$ the pruning of $r\%$ of the parameters of $\theta_{0\to k}$.

Our goal is to identify a point during training, $t < T$, such that $\text{Acc}\left(f(x;\text{Prune}\left(\theta_{0\to t}, r\right)_{t\to T}), D\right) \simeq \text{Acc}\left(f(x;\theta_{0\to T}), D\right)$ where $\text{Prune}\left(\theta_{0\to t}, r\right)_{t\to T}$ denotes the training of parameters $\text{Prune}\left(\theta_{0\to t}, r\right)$ for an additional $T - t$ epochs. In words, we want to train a model for t epochs, prune it, and then continue training it for the remaining $T - t$ epochs. Ultimately, we would like t to be such that the performance of the pruned model, $\text{Prune}\left(\theta_t, r\right)_{t\to T}$, is similar to the performance of the non-pruned model, θ_T. Below we describe our pruning methods and the metrics we use to help identify t. We emphasize that our pruning method does not entail fine-tuning the pruned model beyond the original training compute of T epochs.

Pruning. We compress our model using unstructured pruning [34], which simply zeros out some of the elements in $\theta_{0\to k}$ by performing an element-wise multiplication with binary mask

$\mathcal{M}_r$: Prune $(\theta_{0\to k}, r) = \theta_{0\to k} \odot \mathcal{M}_r$ where the percentage of zeros in $\mathcal{M}_r \in \mathbb{R}^d$ is r. $\mathcal{M}_r$ is constructed such that only the top $(100-r)\%$ of weights of $\theta_{0\to k}$ are kept, and all other values are set to 0. In this chapter, the "top $(100-r)\%$" of weights refers to the weights with the largest magnitude; however, as we discuss in the next section, we also consider another metric based on the Fisher Information in order to decide which parameters to prune. Our pruning is done in a one-shot manner, i.e., we only prune the model once at the target sparsity level. We consider both global and local unstructured pruning. In global pruning, we keep the top $(100-r)\%$ of parameters across the entire model. In local pruning, we keep the top $(100-r)\%$ of parameters within each layer.

Compression Signal. Our task at hand is to judiciously decide *when* to prune a model during training, and *which* parameters to prune. Toward this end, we begin by performing a perturbation analysis on the predicted posterior distribution $p(y|x;\theta)$ modeled by the network $f(x;\theta)$. Given a perturbation, $\bar{\theta} = \theta + \Delta\theta$, one way of measuring the discrepancy between posterior distributions parameterized by θ and $\bar{\theta}$ is via the KL divergence, whose second-order approximation is given by

$$\mathbb{E}_{x\sim\hat{Q}(x)}[KL(p(y|x,\bar{\theta})\|p(y|x,\theta))] = \Delta\theta^T F \Delta\theta + o(\Delta\theta^2) \tag{2.1}$$

where $\hat{Q}(x)$ denotes the training dataset and F is the Fisher Information Matrix (FIM) given by

$$F = \mathbb{E}_{x\sim\hat{Q}(x)}\left[\mathbb{E}_{y\sim p(y|x;\theta)}\left[\nabla_\theta \log p(y|x;\theta) \nabla_\theta \log p(y|x;\theta)^T\right]\right]. \tag{2.2}$$

The FIM can therefore be interpreted as a local measure that quantifies how much the perturbation to a set of weights affects the predicted distribution. As such, while we focus on

magnitude-based pruning due to its simplicity, we also consider pruning parameters with a low Fisher Information, which helps us decide *which* parameters to prune. However, computing the full FIM given by Eq. (2.2) is computationally expensive, so we compute only its trace, which is given by

$$\mathrm{tr}(F) = \mathbb{E}_{x\sim\hat{Q}(x)}\left[\mathbb{E}_{y\sim p(y|x;\theta)}\left[||\nabla_\theta \log p(y|x;\theta)||^2\right]\right]. \tag{2.3}$$

The trace of the FIM, denoted by $\mathrm{tr}(F)$, can be well-approximated using Monte Carlo estimation [3]. For a set of samples $\{x^{(i)}\}_{i=1}^N$ sampled from the training dataset, and corresponding predicted labels $\{\tilde{y}^{(i)}\}_{i=1}^N$, we use Monte Carlo sampling to estimate Eq. (2.3) by computing:

$$\mathrm{tr}(F) \approx \frac{1}{N}\sum_{i=1}^{N} ||\nabla_\theta \log p(\tilde{y}^{(i)}|x^{(i)};\theta)||^2. \tag{2.4}$$

To help us understand *when* a model is most amenable to pruning, we draw inspiration from [3], where it was shown that models undergo two phases during training: a "memorization" phase and a "forgetting" phase. These two phases can be observed by measuring $\mathrm{tr}(F)$ during training. During the memorization phase, which occurs first, the model's weights undergo an increase in the Fisher Information; afterwards, the model undergoes a "forgetting" phase, in which the Fisher Information of the weights decreases. Since the Fisher Information measures how much the predicted distribution is affected by a perturbation to the weights, weights with a small Fisher Information do not have a significant influence on outputs. The shift from the memorization phase to the forgetting phase therefore marks the period when parameters begin to become redundant. Hence, we explore whether $\mathrm{tr}(F)$, computed over the training set via Eq. (2.4), can be used as a metric for deciding when to prune a model.

Efficient Compression Signal. While the trace of the FIM can be approximated using Monte Carlo estimation as in Eq. (2.4), it requires additional gradient computations beyond those computed for training the model, which will slow down training. To remedy this, we propose an alternative gradient-free metric that can be computed efficiently during training. In particular, we propose tracking the expected KL divergence between the predicted distribution, $p(y|x;\theta)$, and the uniform categorical distribution over C classes, $p_u(y|x) = \frac{1}{C}$ for $y \in [C]$:

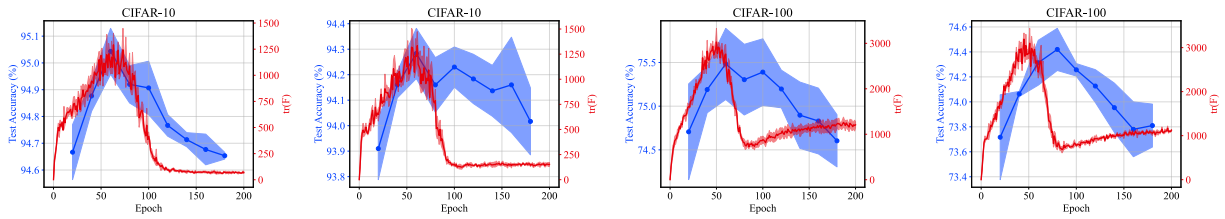
$$KL_{Uniform} \triangleq \mathbb{E}_{x\sim\hat{Q}(x)}\left[KL(p_u(y|x)\|p(y|x;\theta))\right]. \tag{2.5}$$

At initialization, the network's predicted distribution is nearly uniform; as training progresses, the learned distribution diverges from the uniform distribution, and computing Eq. (2.5) during training quantifies how this divergence evolves. We found that this metric also correlates with a model's early pruning period.

Learning Rate Scaling. In the standard train-prune-fine-tune framework [34], the learning rate used to fine-tune the pruned model is smaller than the learning rate used to train the dense model—typically $\frac{1}{10}$th of the original learning rate. However, since we apply pruning earlier in training, we found that simply thresholding the learning rate defined by the learning rate scheduler at the pruning epoch worked well. More precisely, the maximum learning rate used to train the pruned model at epoch t was $\max(\Lambda, LR(t))$ where $LR(t)$ is the learning rate schedule's value at epoch t and Λ is a lower bound on the learning rate.

2.2 Experiments

We conduct extensive experiments on ResNet [38] and VGG [96] models on the CIFAR-10 and CIFAR-100 datasets [59]. Training details are provided in Sec. 2.5.1. We conduct experiments with global pruning in Sec. 2.2.1, and local pruning in Sec. 2.2.2. For each of these settings,



(a) (b) (c) (d)

Figure 2.1: **Early pruning periods correlate with tr**(F). Blue points denote the test accuracies of models pruned at a 90% sparsity level at the corresponding epoch t and fine-tuned for $200 - t$ epochs. The red curve denotes the trace of the Fisher Information Matrix of the baseline (unpruned) models computed over the training set. The mean and standard deviation of three random seeds are plotted. (a): ResNet18 on CIFAR-10. Baseline accuracy: 95.3 ± 0.06. (b): VGG11 on CIFAR-10. Baseline accuracy: 94.7 ± 0.07. (c) ResNet18 on CIFAR-100. Baseline: 77.3 ± 0.1. (d): VGG11 on CIFAR-10. Baseline accuracy: 76.03 ± 0.11.

we ablate over model size, compression levels, learning rate schedules, and regularization.

2.2.1 Early (Global) Pruning Periods

In this section, we compress models via global unstructured magnitude-based pruning, where we keep the top $(100 - r)\%$ of parameters across the entire model.

Early pruning. Discriminative models can be effectively pruned after an initial transient period. We trained a ResNet18 and VGG11 model on both CIFAR-10 and CIFAR-100 for 200 epochs, saving a model checkpoint after each epoch. After training, we loaded the model at various saved checkpoints, t, pruned 90% of the model's parameters using unstructured magnitude-based pruning, and then resumed training for the remaining $200 - t$ epochs with the learning rate computed as explained in Sec. 2.1. Our results are provided in Fig. 2.1, where we load and fine-tune globally-pruned models at every 20th epoch. This procedure was repeated three times, and we show results for the mean $\pm$ the standard deviation across the three trials (illustrated with the low opacity bands). As illustrated by the blue curve, pruning the model early, i.e., around epoch 60-70, yields the highest-accuracy pruned model. The baseline (non-pruned) CIFAR-10 ResNet18 model achieves an average test accuracy of

95.3%, while the ResNet18 CIFAR-100 model achieve an average of 77.3%. Hence, we observe that the model can be pruned early while incurring a minor accuracy degradation of about 0.2% on CIFAR-10, and 1.8% on CIFAR-100—though this is a more challenging task with an aggressive pruning rate. A more moderate level of sparsity can better preserve accuracy, as shown in Fig. 2.5.

These results suggest that a large model capacity is needed for the model to memorize sufficient information about the training data, i.e., cross loss landscape bottlenecks as hypothesized in [3], but once this threshold is crossed, model capacity becomes expendable, and shedding some of these weights can yield more efficient training without a significant reduction in accuracy.

When to prune? The trace of the FIM [4] correlates with the performance of a pruned model, suggesting its potential use as a metric for deciding when to prune a model during training. The red curve in Fig. 2.1 depicts the trace of the FIM of the dense model computed on the training data after every training epoch. We observe the "memorization" and "forgetting" phases first discovered in [3]. In the memorization phase, we see an increase in the amount of information about the dataset that is stored in the model's weights; afterwards, in the forgetting phase, we see a decrease of information. Crucially, we observe that the early pruning period occurs around the peak of $\mathrm{tr}(F)$, that is, at the boundary between the memorization and forgetting phases. This suggests that a model may be most amenable to pruning when it is near the memorization/forgetting phase shift.

2.2.1.1 Fisher-based Pruning Criterion

As discussed in Sec. 2.1, the Fisher Information of a set of the model's weights quantifies how much the predicted posterior distribution is affected by a perturbation to these weights. Hence, the Fisher Information of a set of weights can be interpreted as a measure of their synaptic strength, whereby weights with a low Fisher Information can be considered irrelevant. Next, we experiment with pruning weights based on their Fisher Information, removing

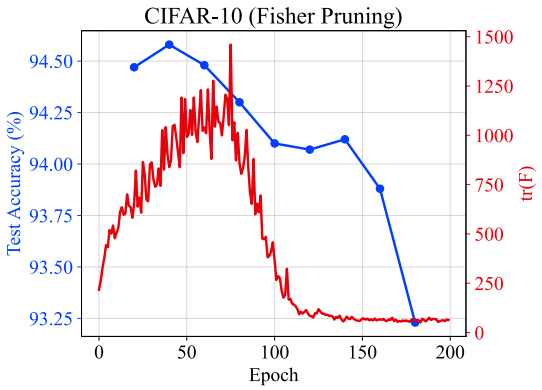
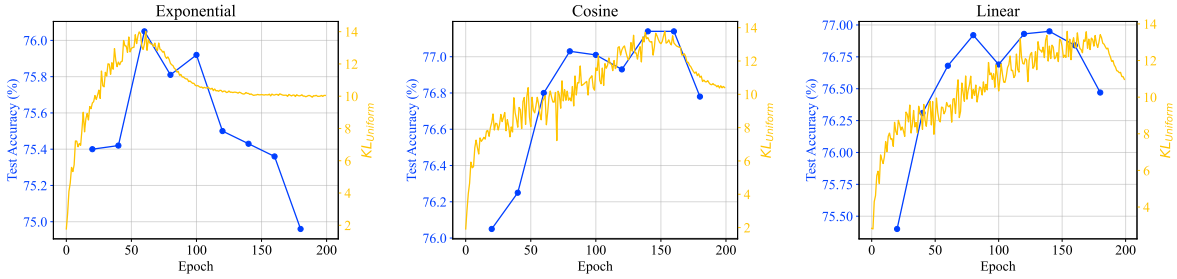


Figure 2.2: **Models pruned via a Fisher Information criterion exhibit an early pruning period.** Rather than prune weights via a magnitude selection criterion, here we use the weights' Fisher Information as measure of importance.

weights with a small Fisher Information. In Fig. 2.2, we prune a ResNet-18 model at a sparsity level of 90% and prune it according to a Fisher Information criterion. We again observe an early pruning period that correlates with the FIM trace. This suggests that $\text{tr}(F)$ can be an effective pruning criterion. However, comparing the performance of this model to the model pruned with the magnitude-based criterion, i.e., Fig. 2.6a in Sec. 2.2.1.3, we observe that the magnitude-based criterion achieves a higher test accuracy. While $\text{tr}(F)$ can often be a good proxy for assessing how much the model has memorized about its training data in aggregate, it may not be the best local measure for assessing parameter importance.

2.2.1.2 Efficient Compression Signal

To decide when to prune a model during training, we would like a metric that can be computed efficiently. As discussed above, $\text{tr}(F)$ can often be an effective signal for deciding when to prune; however, computing $\text{tr}(F)$ requires additional gradient computations beyond those computed to train the model, which will slow down training. To remedy this, we propose a more efficient metric, $KL_{Uniform}$ (see Eq. (2.5)), which forgoes gradient computations. This metric measures the divergence between the uniform distribution over the data's labels (the distribution that assigns equal probability to all labels) and the model's learned posterior



(a) (b) (c)

Figure 2.3: **The early pruning period correlates with the $KL_{Uniform}$ metric.** Models are trained on CIFAR-100 at a 90% sparsity rate. The yellow curve denotes the $KL_{Uniform}$ metric of the baseline (unpruned) models computed over the training set. We observe that $KL_{Uniform}$ correlates with the test accuracies of the pruned models. (a): Models trained with an exponential learning rate schedule. Baseline accuracy: 77.4%. (b): Models trained with a cosine schedule. Baseline accuracy: 78.4%. (c): Models trained with a linear schedule. Baseline accuracy: 78.4%.

distribution. In Fig. 2.3, we plot our $KL_{Uniform}$ metric of the unpruned model trained on CIFAR-100, along with the accuracies of the pruned models. Similar to the correlation observed with $\text{tr}(F)$ in Fig. 2.1, here we again see that the best performing pruned model tends to be the one pruned near the inflection point of the metric. While the inflection point of $\text{tr}(F)$ represents a shift from the memorization phase to the forgetting phase of training, the inflection point of $KL_{Uniform}$ marks a shift of the predicted posterior distribution toward the uniform distribution. This can also be interpreted as a "forgetting" phase, since a decrease in $KL_{Uniform}$ implies an attenuation of the model's overconfident predictions. Moreover, we observe that this metric also correlates with $\text{tr}(F)$, suggesting $KL_{Uniform}$ may be an efficient gradient-free proxy for $\text{tr}(F)$.

2.2.1.3 Ablations

In this section, we investigate the sensitivity of early global pruning periods to model size, compression strength, learning rate schedules, and regularization strengths.

Effect of Model Size. We observe that models across a spectrum of sizes (i.e., number of

parameters) can be effectively pruned early in training. In Fig. 2.1, we showed results for ResNet18, which has approximately 11.2 million (M) parameters, and was originally intended for use on the ImageNet dataset and adapted for CIFAR. We further experimented with early pruning on ResNet20/32/44/56/110, each of which was designed specifically for the CIFAR datasets and range from having 0.27M parameters to 1.7M—all of which are smaller than the previous ResNet18 model. We provide the performance of these models in Fig. 2.4. For all model sizes, we again observe that the performance of pruned models correlates with $\text{tr}(F)$, where the early pruning period occurs near the memorization/forgetting phase shift.

We also note that training a small model for a fixed number of epochs, T, does not perform as well as training a larger model for $t < T$ epochs, pruning it, and then training for $T - t$ epochs. As shown in Fig. 2.1a, pruning ResNet-18 after around 60 epochs of training and then fine-tuning it for 140 epochs yields a model with around 1.1M parameters with an accuracy around 95.04%. In contrast, training the ResNet110 model in Sec. 2.2.1.3 without any pruning yields a model with 1.7M parameters with an accuracy of 93.94% when trained for 200 epochs. Hence, we conclude that a large capacity early in training is beneficial for generalization.

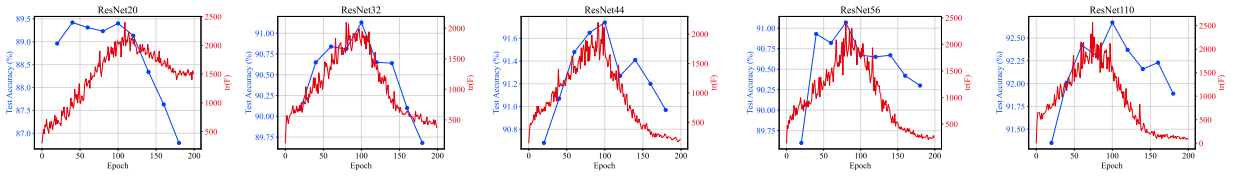


Figure 2.4: **Early pruning periods emerge across models of various sizes**. We train ResNet20/32/44/56/110 on CIFAR-10 for 200 epochs. At every 20th epoch t, we prune 90% of the models' parameters and fine-tune for $200 - t$ epochs. We observe that there is a period early in training when pruning the model performs best. Baselines accuracies: ResNet20: 92.2%, ResNet32: 93.1%, ResNet44: 93.3%, ResNet56: 92.5%, ResNet110: 93.94.

Varying Compression Levels. The benefits of pruning a model earlier in training are agnostic to the sparsity rate. Our experiments thus far have considered compressing a model by 90%—that is, removing 90% of its parameters with the smallest magnitudes. Next,

we investigate how the performance of early-pruned models varies with sparsity levels. In Fig. 2.5, we provide results for a ResNet18 model pruned at multiple compression levels: 80%, 90%, and 95%. Models trained with a more moderate compression level exhibit the highest performance. Nevertheless, the model pruned early at a 95% sparsity rate maintains a competitive performance compared to its baseline (94.7% vs. 95.3%). Additionally, we again observe that the early pruning period occurs near the memorization/forgetting phase shift.

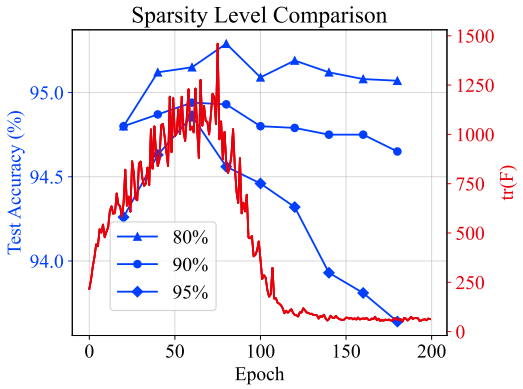
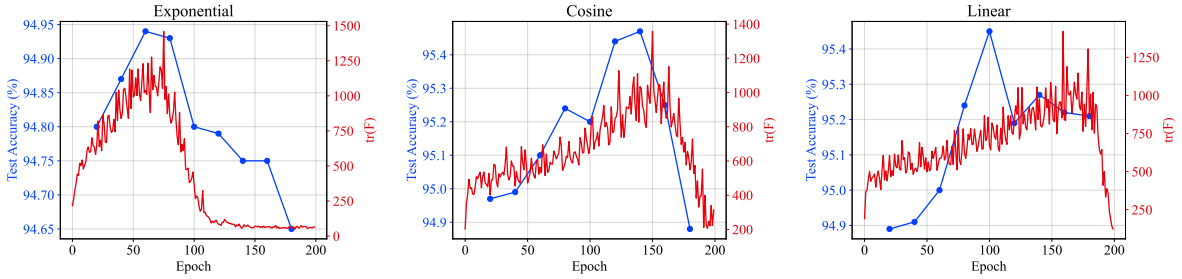


Figure 2.5: **Models pruned at various compression levels perform best when pruned near the peak of tr(F).** We train a ResNet18 model on CIFAR-10 for 200 epochs. At every 20th epoch t, we prune either 80%, 90%, or 95% of the model's parameters via unstructured magnitude-based pruning. Performance of pruned models correlates with the peak of the dense model's FIM trace.

Effect of Learning Rate Schedules. The learning rate schedule can modulate the duration of the memorization and forgetting phases during training, as depicted by the red curves in Fig. 2.6, and hence can control when a model becomes amenable to pruning. Our previous experiments utilized an exponential learning rate decay, which decayed the learning rate by 0.97 after every epoch. Here, we additionally train with cosine and linear decay schedules. In Fig. 2.6, we observe that the peak of the tr(F) curve is dependent on the learning rate, with cosine and linear schedules reaching the forgetting phase later in training. This suggests that learning rate schedules determine the length of the memorization phase. In particular, as depicted in Fig. 2.7, the exponential schedule decays much faster than the cosine and linear



(a) (b) (c)

Figure 2.6: **The learning rate schedule modulates the length of the memorization phase and when the early pruning period occurs**. Models are pruned at 90%. (a): Models trained with an exponential learning rate schedule. Baseline accuracy: 95.3%. (b): Models trained with a cosine schedule. Baseline accuracy: 95.0% (c): Models trained with a linear schedule. Baseline accuracy: 95.0%. When fine-tuning pruned models, we use the same schedule used to train the baseline models.

schedules, suggesting that models enter the forgetting phase once the model has memorized sufficient information and the learning rate has become sufficiently small. Nevertheless, we continue to observe a correlation between the peak of $\text{tr}(F)$ and when the pruned model performs best.

Weight Decay Regularization. The strength of correlation between a model's early pruning period and the trace of the FIM hinges on the model's ability to fit the data well. Our previous experiments employed L2 regularization ("weight decay") as a means of regularizing the network. The weight decay coefficient controls the strength of how much model parameters are decayed; larger coefficients impose stronger regularization, while smaller coefficients may lead to insufficient regularization. Our experiments have thus far used a coefficient of 5×10^{-4}, which we found to perform well. To study the effect of the regularization strength on the early pruning period, we retrained baseline models with coefficients of 5×10^{-3} and 5×10^{-5}, and applied early pruning to these models. Our results are summarized in Fig. 2.8. The baseline model trained with a coefficient of 5×10^{-3} (Fig. 2.8a) underfit the data, while the the model trained with a coefficient of 5×10^{-5} (Fig. 2.8c) overfit the data. All three cases exhibit an early pruning period; however, we observe a weak correlation between when the early

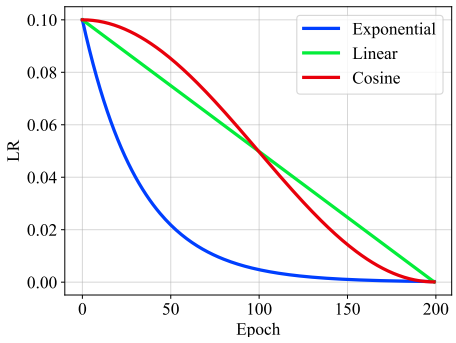
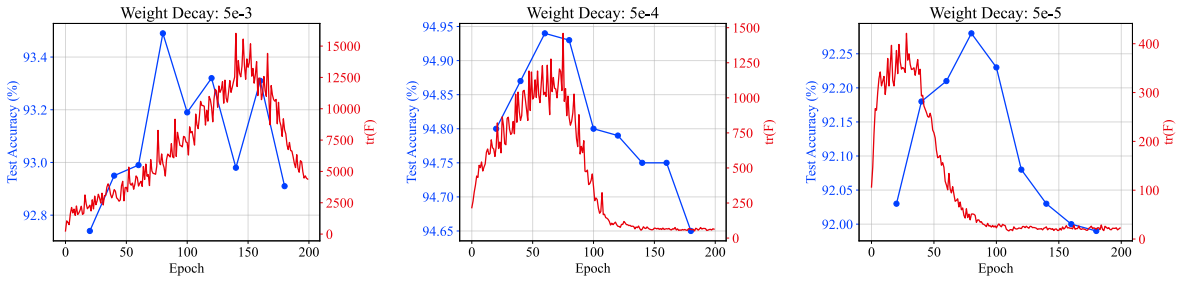


Figure 2.7: **We train models with exponential, cosine, and linear learning rate annealing schedules.** Pruned models follow the same learning rate schedule as their baseline model, with the max learning set as defined in Sec. 2.1.

pruning period occurs and the trace of the FIM. In the case of overfitting, we hypothesize that the weak regularization strength prompted the model to quickly memorize its training data and subsequently enter its forgetting phase early, leading to the discarding of parameters that may have become important later in training. Hence, the early pruning period occurs later—toward the end of the forgetting phase—once the appropriate superfluous weights can be identified. In the case of underfitting, the model receives larger gradient updates (as evidenced by the larger $\text{tr}(F)$ values, which are the norm of the gradients, as per Eq. (2.4)), and so $\text{tr}(F)$ may not begin to decrease until the learning rate is sufficiently small later in training. Hence, while the early weights to prune may have become evident earlier in training, $\text{tr}(F)$ was not able to capture when this occurred. Collectively, these results suggest that a model's early pruning period and its FIM trace are most strongly correlated when the model fits the data well.

2.2.2 Early (Local) Pruning Periods

In this section, we expand on Sec. 2.2.1 and show that models compressed via *local* unstructured magnitude-based pruning, where we keep only the top $(100 - r)\%$ of parameters on a per-layer basis, also exhibit early pruning periods.



(a) Underfitting (b) Good fit (c) Overfitting

Figure 2.8: **How well a model fits the data influences the correlation strength between a model's early pruning period and the trace of its FIM.** We trained a ResNet18 model on CIFAR-10 with varying weight decay coefficients. Each model was then pruned at varying points throughout training and fine-tuned for the remaining epochs. (a): Weight decay coefficient 5×10^{-3}; this baseline model underfit the data. Baseline: 93.4%. (b): 5×10^{-4}; this baseline model fit the data well. Baseline: 95.3%. (c): 5×10^{-5}; this baseline model overfit the data. Baseline: 92.8%.

In Fig. 2.9, we recreate Fig. 2.1 using local pruning and again observe an early pruning period. Moreover, we again see a correlation between the trace of the FIM, and the performance of the pruned model.

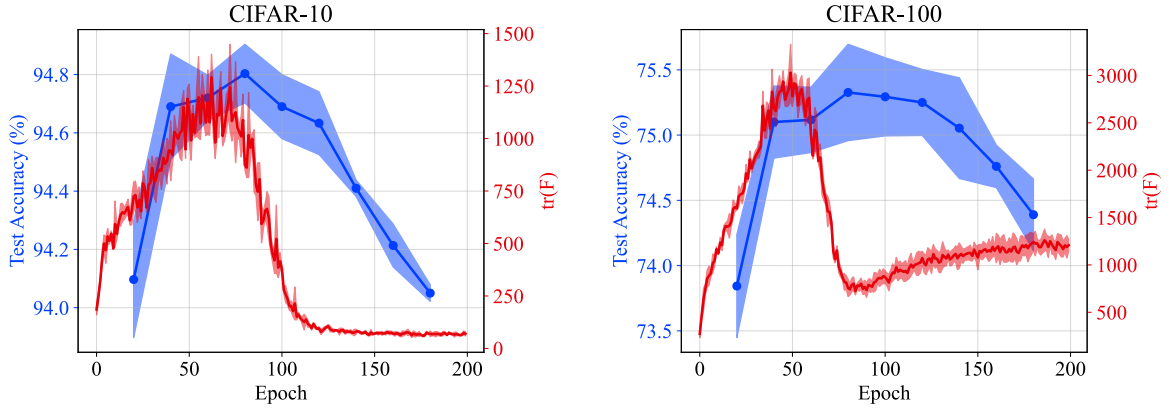
2.2.2.1 Ablations

Next, we investigate the sensitivity of early local pruning periods to model size, compression strength, learning rate schedules, and regularization strengths.

Effect of Model Size. Similar to global pruning, we observe that models across a spectrum of sizes can be effectively pruned via local pruning early in training. In Fig. 2.10, we plot the performance of various ResNet models and again observe that the performance of pruned models correlates with $\mathrm{tr}(F)$.

Varying Compression Levels. As was the case for global pruning, the early pruning period of models pruned via local pruning remain agnostic to the sparsity rate, as illustrated in Fig. 2.11.

Effect of Learning Rate Schedules. We plot the effect of the learning rate schedule on



(a) (b)

Figure 2.9: **Early pruning periods correlate with tr(F) when models are pruned at a local level**. Blue points denote the test accuracies of models pruned at a 90% sparsity level at the corresponding epoch t and fine-tuned for $200 - t$ epochs. The red curve denotes the trace of the Fisher Information Matrix of the baseline (unpruned) models. The mean and standard deviation of three random seeds are plotted. (a): ResNet18 on CIFAR-10. Baseline accuracy: 95.3 ± 0.06. (b) ResNet18 on CIFAR-100. Baseline: 77.3 ± 0.1.

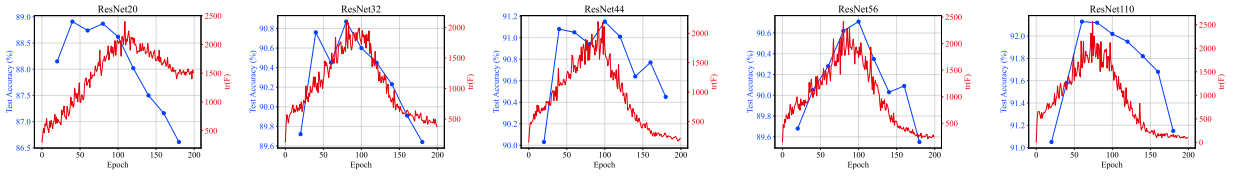


Figure 2.10: **Early pruning periods emerge across models of various sizes when pruned at a local level**. We train ResNet20/32/44/56 on CIFAR-10 for 200 epochs. At every 20th epoch t, we prune 90% of the models' parameters and fine-tune for $200 - t$ epochs. We observe that there is an optimal time to apply pruning early in training. Baselines accuracies: ResNet20: 92.2%, ResNet32: 93.1%, ResNet44: 93.3%, 56: 92.5%, 110: 93.94%.

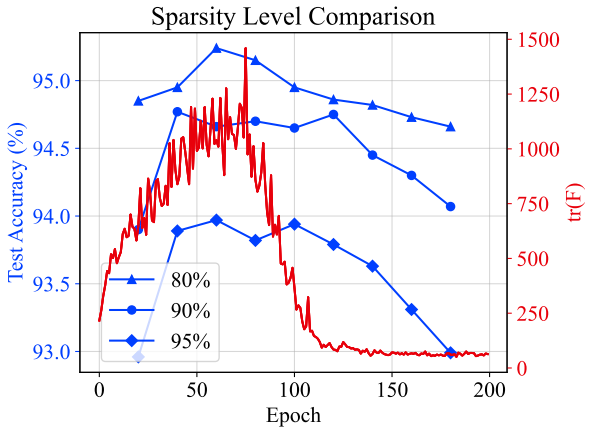
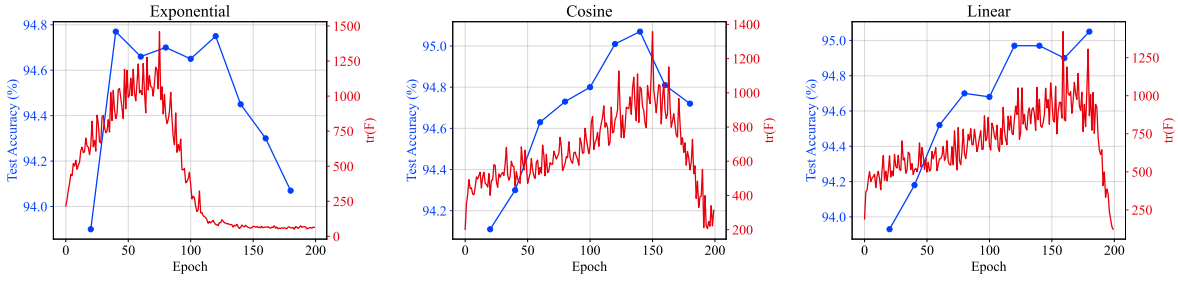
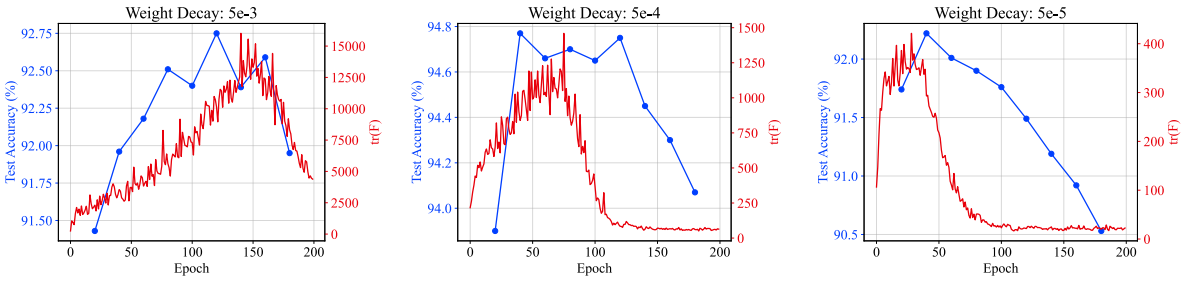


Figure 2.11: **Models pruned via local unstructured pruning at various compression levels perform best when pruned near the peak of tr(F).** We train a ResNet18 model on CIFAR-10 for 200 epochs. At every 20th epoch t, we prune either 80%, 90%, or 95% of the model's parameters via unstructured magnitude-based pruning. Performance of pruned models correlates with the peak of the dense model's FIM trace.



(a) (b) (c)

Figure 2.12: **The learning rate schedule modulates the length of the memorization phase and when the early pruning period occurs for models pruned at a local level**. Models are pruned at 90%. (a): Models trained with an exponential learning rate schedule. Baseline accuracy: 95.3%. (b): Models trained with a cosine schedule. Baseline accuracy: 95.0% (c): Models trained with a linear schedule. Baseline accuracy: 95.0%. When fine-tuning pruned models, we use the same schedule used to train the baseline models.



(a) Underfitting (b) Good fit (c) Overfitting

Figure 2.13: **The early pruning period and the trace of a model's FIM are correlated when the model is pruned at a local level**. We trained a ResNet18 model on CIFAR-10 with varying weight decay coefficients. Each model was then pruned at varying points throughout training and fine-tuned for the remaining epochs. (a): Weight decay coefficient 5×10^{-3}; this baseline model underfit the data. Baseline: 93.4%. (b): 5×10^{-4}; this baseline model fit the data well. Baseline: 95.3%. (c): 5×10^{-5}; this baseline model overfit the data. Baseline: 92.8%.

the early pruning period of models pruned with local pruning in Fig. 2.12. We observe that the time at which the memorization/forgetting phase transition occurs is largely dependent on the learning rate, as we saw for models pruned via global pruning. Moreover, we see that the early pruning period also shifts along with this phase transition.

Weight Decay Regularization. In contrast to global pruning (Fig. 2.13), early pruning periods of models pruned via local pruning correlate more strongly with the trace of the FIM, as illustrated in Fig. 2.13.

2.3 Linear Network Analysis[1]

In this section, we explore early pruning periods in the context of simple linear networks. This linear framework provides an analytically tractable model for understanding the learning dynamics observed in more complex neural networks. Our analysis follows the teacher-student paradigm considered in previous works [93, 94].

[1]Experiments in this section were done while interning at AWS in collaboration with Michael Kleinman.

2.3.1 Experimental Setup of Teacher-Student Network

Teacher Network. We begin by assuming there is some underlying target function that a "teacher" network aims to approximate. The teacher network takes $x \in \mathbb{R}^{d_{in}}$ and outputs $\tilde{y} = Wx + \epsilon$ where $W \in \mathbb{R}^{d_{out} \times d_{in}}$ and $\epsilon \sim \mathcal{N}\left(\mathbf{0}, \frac{1}{d_{out}} I_{d_{out}}\right)$. The noise, ϵ, represents the error bias in statistical learning theory. More precisely, we assume there is some underlying target function that has generated the data, and the student network aims to learn this function, but instead learns the mapping $\tilde{y} = Wx + \epsilon$ due to its limited capacity to fit the target function. W is constructed as a random matrix with a fixed rank, r. In particular, we construct W by considering its singular-value decomposition (SVD), $W = U_W \Sigma_W V_W^T$ where $U_W \in \mathbb{R}^{d_{out} \times d_{out}}$, $\Sigma_W \in \mathbb{R}^{d_{out} \times d_{in}}$, and $V_W \in \mathbb{R}^{d_{in} \times d_{in}}$. U_W and V_W are set by sampling form the Haar distribution. We set r elements along the diagonal of Σ_W to be nonzero so that the rank of W is r.

Training/Test Data. Having defined the teacher network, we then use it to construct synthetic data for the student network. The student network is trained to approximate the teacher network. We first create the set, $\{x_i\}_{i=1}^N$ where $x_i \in \mathbb{R}^{d_{in}}$ and $x_i \sim \mathcal{N}\left(0, \frac{1}{d_{in}} I_{d_{in}}\right)$. Next, we construct the dataset $\mathcal{D}_{train} = \{(x_i, \tilde{y}_i\}_{i=1}^N$ where $\tilde{y}_i = Wx_i + \epsilon$ are the outputs of the teacher network for input x_i. The "true" labels are the denoised $y_i = Wx_i$, which yields the test data $\mathcal{D}_{test} = \{(x_i, y_i)\}_{i=1}^N$.

Student Network. The student network is a two-layer fully-connected network. The first layer is parameterized by $W_1 \in \mathbb{R}^{d_h \times d_{in}}$, and the second layer by $W_2 \in \mathbb{R}^{d_{out} \times d_h}$. For input $x \in \mathbb{R}^{d_{in}}$, the network outputs $\hat{y} = W_2 W_1 x \in \mathbb{R}^{d_{out}}$.

Training Procedure. We construct $\mathcal{D}$ with $N = d_{in} = d_h = 100$. We set the rank of W to $r = 3$ with singular values 5, 3.5, and 2. We train the network using the dataset $\mathcal{D}_{train}$ with a learning rate of 0.002 for 10^4 epochs and minimize the mean squared error loss given by Eq. (2.6) via gradient descent. The test loss is measured over $\mathcal{D}_{test}$ and is given by Eq. (2.7).

$$\mathcal{L}_{train} = \sum_{i=1}^{N} ||\tilde{y}_i - W_2 W_1 x_i||_2^2 \tag{2.6}$$

$$\mathcal{L}_{test} = \sum_{i=1}^{N} ||y_i - W_2 W_1 x_i||_2^2 \tag{2.7}$$

2.3.2 Linear Network Analysis Results

We plot the train and test losses of our linear network after pruning at various epochs in Fig. 2.14. Specifically, we prune the network at epoch t by 90% (via magnitude-based pruning) and then continue training the model for an additional $10^4 - t$ epochs. We first observe that pruning right at the outset (epoch 0) yields the worst performance. This is expected since the pruning is done via magnitude-based pruning and no information about the training data has been used in the pruning selection criterion. Moreover, we see that pruning the model at the end of training (epoch 10^4) also yields a model with high error, as the model is not fine-tuned to compensate for the pruning.

Notably, we observe that the test error exhibits an early pruning period between epochs 4000 and 5200—pruning the model at these epochs yields the lowest test error. Beyond epoch 5200, the test error begins to increase, while the training error continues to decrease. This is indicative of overfitting, and suggests that the early pruning period of a model occurs before it begins to over fit its training data. To investigate this further, we examine the singular values of the student network's learned approximation to W, given by W_2W_1. As the student network fits noisy labels, as given by Eq. (2.6), we do not expect the learned W_2W_1 to capture the salient features of W, i.e., the singular values of W, perfectly. However, we expect the learned singular values to be similar to W.

In Fig. 2.15, we plot the singular values of W_2W_1 throughout training. When the network is pruned prior to any training as in Fig. 2.15a, we see that the top three singular values of W_2W_1 do not come close to the those of W; hence, the model is underfitting. Pruning at the

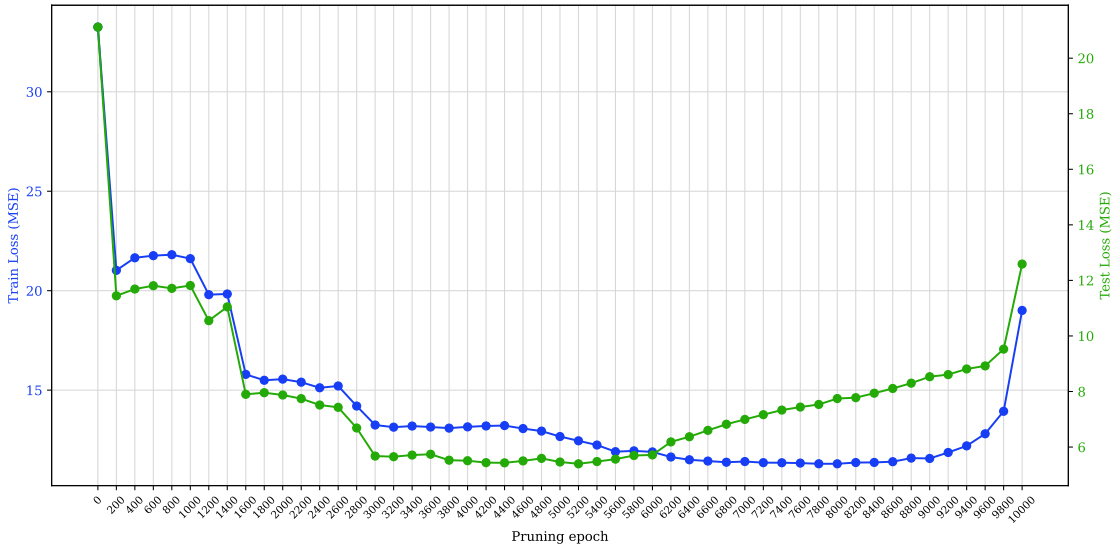


Figure 2.14: **Linear Networks Exhibit an Optimal Pruning Period.** We train a two-layer linear network on synthetic data for 10^4 epochs. Each point represents the train/test loss of the model pruned with 90% sparsity rate at epoch t, and then trained for the remaining $10^4 - t$ epochs. We observe that the early pruning period occurs near the epoch when the model begins to overfit.

end of training as in Fig. 2.15b yields a network whose top three singular values are close to those of W, but the network has also learned many noisy features (as evidenced by the band of smaller singular values); hence, this network has overfit to noise. Pruning too early in training, such as in Fig. 2.15c, yields a sub-optimal network where we see that, despite improving upon pruning at epoch 0, W_2W_1 has still not fit the salient features of W. Pruning a bit later in training, such as at epochs 3000 or 5200 as in Figs. 2.15d and 2.15e, yields a network that fits the salient features of W well, while mitigating the amount of learned noisy features. Pruning later in training, such as at epochs 7000 and 9000 as in Figs. 2.15f and 2.15g produces a network that overfits to the noise, as evidenced by the larger singular values besides the top three salient ones.

To conclude, our analysis on a linear network points to an early pruning of networks prior to when overfitting occurs, and after the salient features of the data have been learned by the dense model. This is consistent with our finding that the early pruning period correlates

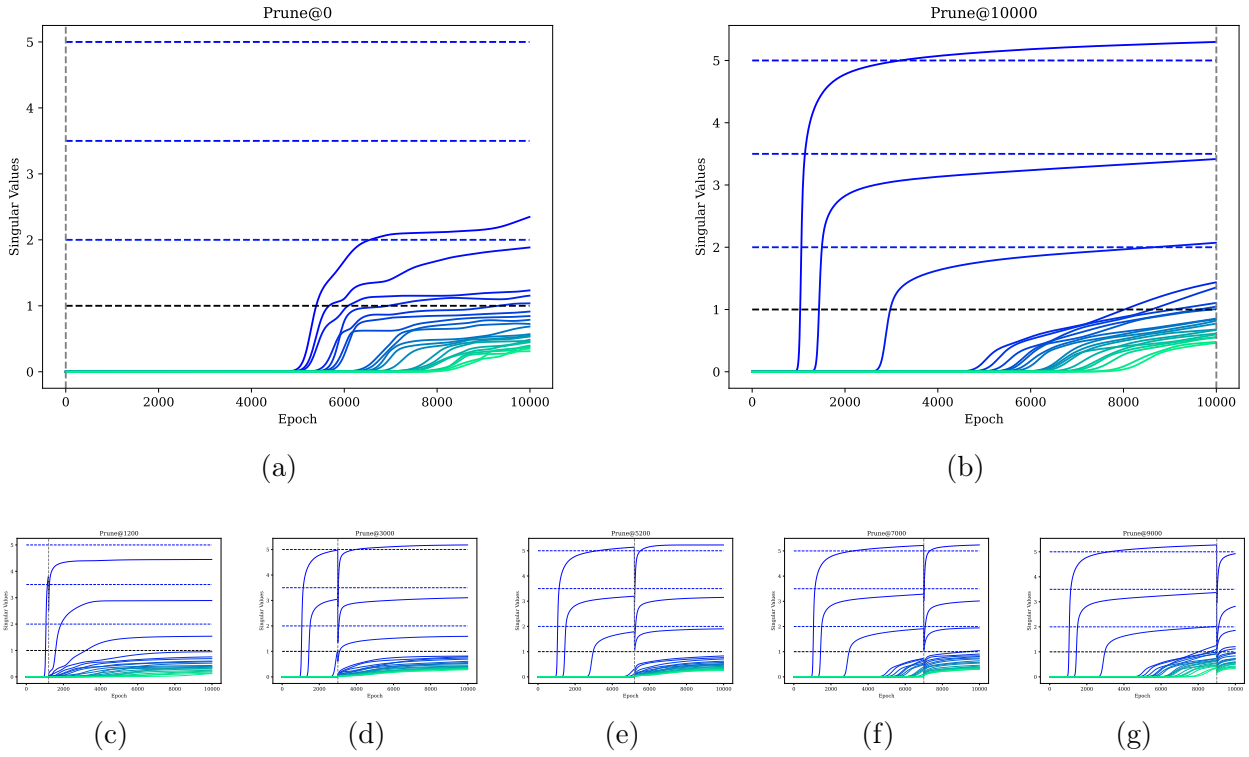
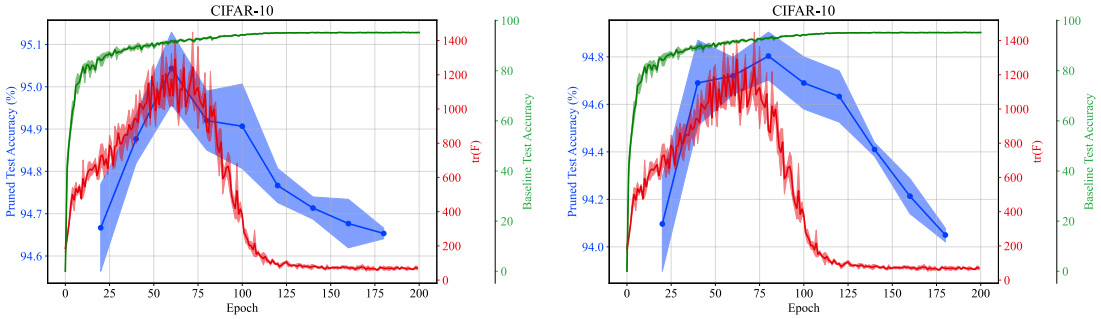


Figure 2.15: **The early pruning period of a linear network occurs after learning the most salient features, and before fitting noise.** We consider a two-layer fully connected linear network which learns the mapping $\hat{y} = W_2W_1x$ where W_2W_1 aims to approximate W which has rank 3 with singular values 5, 3.5, 2. Dashed blue lines are the singular values of W. We prune the network at different epochs t and continue training for $10^4 - t$ epochs. We plot the singular values of W_2W_1 throughout training. each subplot prunes the network at a different training epoch with a sparsity rate of 90%. (a): Pruning the network at epoch 0 (before training) yields to a network that does not learn the salient features of W. (b): Pruning the network at the end of training produces a model that captures salient features, but also a lot of noise in the data. (c)-(d): Pruning too early leads to a network that does not adequately learn the salient features of W. (e): Pruning around epoch 5000 yields a model that captures salient features well without fitting too much to noise. (f)-(g): Pruning too late in training yields a model that fits too much to noise.

with the Fisher Information. As shown in [3], the test error of a model typically plateaus toward the end of the Fisher Information. This plateau coincides with when the network would begin to overfit its training data, modulo proper regularization.



(a) (b)

Figure 2.16: **Similar to linear networks, the optimal time to prune a deep non-linear network occurs before convergence/overfitting.** We train a ResNet18 model on CIFAR-10 and plot the test accuracy throughout training in green. Moreover, we prune the model with 90% sparsity rate at various epochs and plot the final test accuracy in blue. We observe that the optimal performance of the pruned models occurs before the test accuracy of the saturates, suggesting that the optimal pruning period occurs before "overfitting," though due to regularization during training, we observe saturation rather than overfitting. (a): Pruning using global pruning. (b): Pruning using local pruning.

2.3.3 Extrapolating from Linear Networks To Deep Non-Linear Networks

Next, we validate our conclusions drawn in the linear network case to larger non-linear networks. In Fig. 2.16, we plot the test accuracies of baseline ResNet18 models trained on CIFAR-10 throughout training. In contrast to the linear setting, here the network is heavily regularized with data augmentations during training in an effort to mitigate overfitting. As such, we see that the baseline test accuracy (in green) eventually plateaus. However, when overlaying the final accuracies of the pruned models (in blue), we observe that models perform best when pruned before the test accuracy of the baseline model plateaus. This is consistent with our analysis in the linear network setting where we observed that pruning too late could cause the model to overfit to noise, and it is best to prune prior to this.

2.4 Discussion

In this chapter, we explored the question of *when* a model is most amenable to compression via pruning, particularly during training. We investigated model compression in the context of one-shot unstructured pruning, whereby we pruned the model once *during* training, and then fine-tuned the model for the remaining epochs, staying within the original computational budget, and yielding a pruned model without additional fine-tuning. We showed a correlation between the trace of a model's Fisher Information, and when the pruned model performs best—what we refer to as "early pruning periods." By tracking a model's Fisher Information, $\text{tr}(F)$, during training, we found that a pruned model's performance is often best if it is pruned near the transition from the memorization phase to the forgetting phase. We posit that the evolution of the Fisher Information throughout training may serve as a low-fidelity signal that quantifies a model's learned redundancy. Our experiments highlight the need for a large model capacity at the outset of training, but after the initial transient period (i.e., near the memorization/forgetting phases), this capacity can be reduced in order to expedite training. Our analysis on linear networks suggest that this transient period consists of the model learning the most salient features of the training data, and networks are most amenable to pruning after it has learned these features and before it overtfits to its training data.

In addition to showing a correlation between the trace of the Fisher Information and the accuracy of pruned models at different stages of training, we also showed a correlation between our $KL_{Uniform}$ metric and when pruned models performed best. $KL_{Uniform}$ measures the KL divergence between the uniform posterior distribution assigning equal probability to data labels, $p_u(y|x) = \frac{1}{C}$, and the predicted posterior distribution defined by the model, $p(y|x, \theta)$. This gradient-free metric can be computed efficiently during training and may serve as a proxy for $\text{tr}(F)$, making it suitable for deciding when to prune a model. While we have shown that this metric tends to correlate with when early pruning periods occur, it is not clear whether this will be suitable for larger models and datasets. Exploring efficient functions for

identifying early pruning periods remains as future work.

Due to computational constraints, the experiments in this chapter were performed on small-scale models and datasets. Validating the claims presented in this chapter on large-scale models and datasets—as well as on other forms of compression, such as structured pruning and quantization—remains an exciting future work with many practical applications that can make training more efficient and can democratize access to large-scale training.

2.5 Appendix

2.5.1 Training Details

Our baseline models are trained on either CIFAR-10 or CIFAR-100 for 200 epochs. We use random horizontal flipping and random cropping data augmentations. We use an SGD optimizer with 0.9 momentum and a maximum learning rate of 0.1 (without a warm-up). We use a batch size of 128. Unless stated otherwise, models are trained with a weight decay value of 5×10^{-4} and with an exponential learning rate schedule that decays the learning rate by 0.97 after each epoch. When a model is pruned at epoch t, the pruned model is trained for an additional $200 - t$ epochs with the same learning rate schedule as its base model. However, as explained in Sec. 2.1, the pruned model is trained with a different maximum learning rate, namely $\max(\Delta, LR(t))$ where $LR(t)$ is the learning rate set by the learning rate schedule at epoch t, and $\Delta = 0.001$ is a lower bound hyperparameter. We choose this learning rate so that models pruned later in training are trained with a sufficiently large learning rate to recover performance, but not too large to compromise generalization; models pruned earlier will be trained with a learning rate similar to that which would be used by the base model.

CHAPTER 3

Dynamic Compression During Training for Real-Time Adaptive Inference[1]

In Chapter 2, we showed that a model's large capacity is needed during the transient period of training, but the model becomes amenable to pruning after it has learned the salient features of the training data. In this chapter, we leverage dynamic model compression during training for improved training efficiency, and to enable a model to adapt to arbitrary compression levels in real-time during inference. We expand our compression during training framework to include structured pruning and quantization. Structured pruning removes contiguous groups of weights from a network, such as entire columns or rows in linear layers, or channels from convolution layers. Quantization reduces memory and compute requirements by representing model weights with lower bit-width precision. We explore these compression methods because, in contrast to unstructured pruning, they provide practical efficiency once deployed: models pruned in a structured manner can be deployed without requiring special hardware, and quantized networks can be deployed to devices that support multiple bit-width configurations (e.g., most modern smartphones).

Models are deployed to a variety of computing platforms, including phones, tablets, and watches. When deploying models to a device, it is traditionally assumed that available computational resources (compute, memory, and power) remain static. However, real-world computing systems do not always provide stable resource guarantees. Computational resources need to be conserved when load from other processes is high, or available memory is low.

[1]Work completed during an internship at Apple.

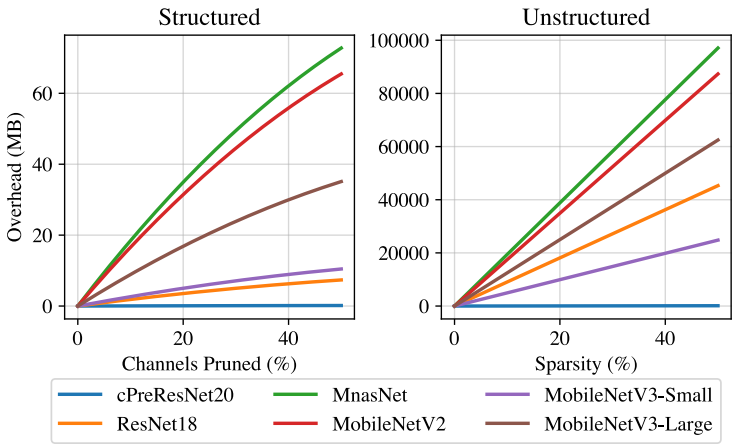


Figure 3.1: **Storing a set of BatchNorm parameters for every compression configuration is expensive.** We visualize the parameter overhead in Megabytes (MB) for storing an extra set of pre-calibrated BatchNorm statistics for every possible sparsity configuration between 0% sparsity and the given compression level. Our method avoids this overhead by eliminating the need for storing BatchNorm statistics (Sec. 3.2.5).

In this chapter, we present a dynamic compression training procedure to produce models that, once deployed, can be compressed in real-time to arbitrary compression levels entirely on-device. This enables the deployment of a single model that can efficiently adapt to its host device's available resources. We formulate this problem as learning an adaptively compressible network subspace [121], where we gradually optimize one end for accuracy, and the other for efficiency. Notably, our subspace models do not require recalibration nor retraining when changing compression levels.

3.1 Overhead of Adaptive Inference by Deploying Multiple Models

Models are generally designed to consume a fixed budget of resources, but the compute resources available on a device can vary over time. Computational burden from other processes, as well as battery life, may influence the availability of resources to a model. Adaptively adjusting inference-time load is beyond the capabilities of traditional neural networks, which are designed with a fixed architecture and a fixed resource usage. A simple approach to the

problem of providing an accuracy-efficiency trade-off is to train multiple neural networks of different sizes. Multiple networks are then stored on the device and loaded into memory when needed. There is a breadth of research in the design of efficient architectures that can be trained with different capacities, then deployed on a device [44, 92, 43]. However, there are a few drawbacks to using multiple networks to provide an accuracy-efficiency trade-off: (1) it requires training and deploying multiple networks (which induces training-time computational burden and on-device storage burden), (2) it requires all compression levels to be specified before deployment, and (3) it requires new networks to be loaded into memory when changing the compression level, which prohibits real-time model switching on memory-constrained edge devices.

Previous methods such as Network Slimming [126] and Universal Slimming [125] address the first issue in the setting of structured sparsity by training a single network conditioned to perform well when varying the number of channels pruned. However, these methods require BatchNorm [52] statistics to be recalibrated for every accuracy-efficiency configuration before deployment. This requires users to know every compression level in advance. If a large number of compression levels are chosen, the storage burden of BatchNorm statistics is significant (Fig. 3.1), especially for low-compute devices. If only a few compression levels are chosen, a user will have to sacrifice accuracy and underutilize available resources by choosing a smaller model if the desired compression level is not available.

In the next section, we introduce a training framework, Learning Compressible Subspaces (LCS), designed to produce models that adapt to a device's resources in real-time, guided by a progressive compression algorithm.

3.2 Compressible Subspaces for Real-Time Adaptive Inference

3.2.1 Compressible Lines

Our method involves training a neural network subspace [121] that contains a spectrum of networks that each have a different accuracy-efficiency trade-off. We recast the subspace formulation of [121] to train a linear subspace with high accuracy solutions at one end and high efficiency solutions at the other end, as depicted in Fig. 3.2.

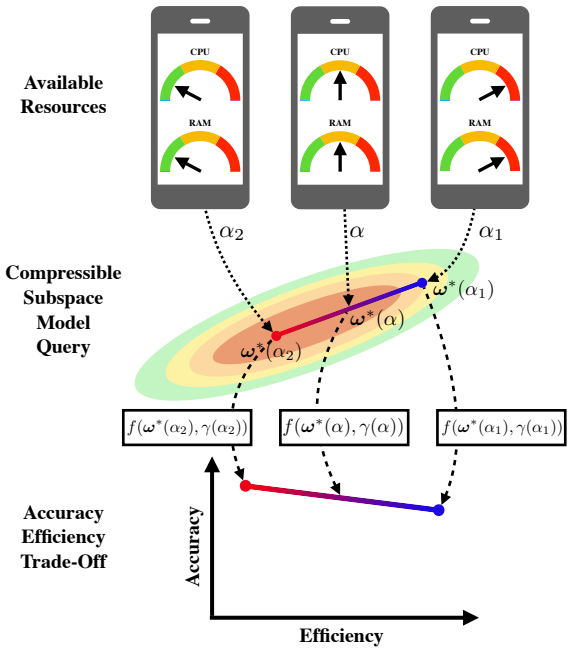


Figure 3.2: **Depiction of our method for learning a compressible linear subspace.** Our linear subspace of networks $\boldsymbol{\omega}^*$ is parameterized by $\alpha \in [\alpha_1, \alpha_2]$. Networks with $\alpha \approx \alpha_2$ exhibit high accuracy and low efficiency, while networks with $\alpha \approx \alpha_1$ trade off accuracy in favor of high efficiency. By varying $\alpha \in (\alpha_1, \alpha_2)$, we obtain a spectrum of networks which exhibit a strong accuracy-efficiency trade-off.

To learn a compressible subspace, we choose a model architecture and denote its collection of weights by $\boldsymbol{\omega}$. We randomly initialize two sets of network weights, $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$, to define the endpoints of our subspace. Our network subspace spans the line between $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ and is defined by $\boldsymbol{\omega}^*(\alpha) = \alpha\boldsymbol{\omega}_1 + (1-\alpha)\boldsymbol{\omega}_2$, where $\alpha \in [\alpha_1, \alpha_2]$, and $0 \leq \alpha_1 < \alpha_2 \leq 1$. In other words, by varying our subspace parameter, α, we can obtain a set of weights $\boldsymbol{\omega}^*(\alpha)$ through

interpolation.

We now adjust our subspace so that one end (e.g., $\alpha \approx \alpha_1$) yields highly compressed networks, and the other end (e.g., $\alpha \approx \alpha_2$) yields highly accurate networks. Intermediate values (e.g., $\alpha \approx (\alpha_1 + \alpha_2)/2$) should exhibit moderate compression. In other words, tuning $\alpha \in [\alpha_1, \alpha_2]$ allows us to move along the subspace, and we would like different points along the subspace to exhibit different accuracy-efficiency trade-offs. To achieve this, we introduce a function $\gamma(\alpha)$ which determines how much to compress the network at α, and a compression function $f(\boldsymbol{\omega}, \gamma)$ which compresses $\boldsymbol{\omega}$ with compression level γ.

To train our subspace, we first sample a position on the subspace by randomly choosing some $\alpha \in [\alpha_1, \alpha_2]$, yielding a network with weights $\boldsymbol{\omega}^*(\alpha)$. We then compute $\gamma(\alpha)$, which determines how much to compress the network. Finally, f compresses the network, obtaining a network with weights $f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha))$. We then perform a standard forward and backward pass of gradient descent with it, backpropagating gradients to $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$. We continue training in this manner until convergence.

Once our model is trained, a user can deploy $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ on-device to allow efficient, adaptive, real-time compression, as depicted in Fig. 3.2. To change compression levels in real-time, the user first determines how many resources are available on the device; this step is application-dependent, and may involve looking at the amount of currently available memory or the current CPU load. The user then chooses the compression level γ_0 based on currently available resources. From this quantity, the user calculates the appropriate $\alpha_0 = \gamma^{-1}(\gamma_0)$ value corresponding to the desired compression level. Next, the user computes the compressed network, $f(\boldsymbol{\omega}^*(\alpha_0), \gamma_0)$. This network is used until a new compression level is desired by the user. Note that computing f is negligible compared to the cost of a network forward pass in all our experiments (see Sec. 3.2.6).

Table 3.1: **Our compressible subspaces require no retraining, no BatchNorm recalibration, and are adaptive.** We compare our method with a linear subspace (LCS+L) and a point subspace (LCS+P) to LEC [67], NS [126], and US [125]. Note that "Adaptive" refers to post-deployment compression at any compression level. $|\boldsymbol{\omega}|$ denotes the number of network parameters, $|b|$ denotes the number of BatchNorm parameters, and n denotes the number of compression levels for models that do not support arbitrary compression levels.

	LCS+P	LCS+L	LEC	NS	US
No Retraining	✔	✔	✘	✔	✔
No Norm Recalibration	✔	✔	✘	✘	✘
Adaptive	✔	✔	✘	✘	✘
Stored Parameters	$\|\boldsymbol{\omega}\|$	$2\|\boldsymbol{\omega}\|$	$n\|\boldsymbol{\omega}\|$	$\|\boldsymbol{\omega}\| + n\|b\|$	$\|\boldsymbol{\omega}\| + n\|b\|$

3.2.2 Compressible Points

In Sec. 3.2.1, we discussed formulating our subspace as a line connected by two endpoints in weight space. This formulation requires additional storage resources to deploy the subspace (Table 3.1), since an extra copy of network weights is stored. For many cost-efficient computing devices, this overhead may be significant. To eliminate this need, we propose training a degenerate subspace with a single point in weight-space (rather than two endpoints). We still use $\alpha \in [\alpha_1, \alpha_2]$ to control our compression ratio, but our subspace is parameterized by a single set of weights, $\boldsymbol{\omega}^*(\alpha) = \boldsymbol{\omega}$. The compressed weights are now expressed as $f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha)) \equiv f(\boldsymbol{\omega}, \gamma(\alpha))$. This corresponds to applying varying levels of compression during each forward pass.

This method still produces a subspace of models in the sense that, for each value of α, we obtain a different compressed network $f(\boldsymbol{\omega}, \gamma(\alpha))$. However, we no longer use different endpoints of a linear subspace to specialize one end of the subspace for accuracy and the other for efficiency. Instead, we condition one set of network weights to tolerate varying levels of compression.

3.2.3 Curriculum-Based Compression

When training our compressible subspaces, we need to sample our subspace parameter, α, at each iteration of training. In [125], a "sandwich method" for training with varying levels of structured sparsity is proposed. This method involves performing n rounds of forward and backward passes in each iteration of training. One round uses the maximum sparsity level, another round uses the minimum sparsity level, and the remaining $n-2$ rounds use randomly chosen sparsity levels. After all n rounds of forward and backward passes, the gradient update is applied.

To incorporate this method into our algorithm, we introduce a stochastic function, $\boldsymbol{\alpha}_n : \boldsymbol{\Omega} \to [\alpha_1, \alpha_2]^n$, where $\boldsymbol{\Omega}$ represents the state of the stochastic function (e.g., the internal state of a random number generator). For each training batch, we sample $\boldsymbol{\alpha} \in [0,1]^n$ from $\boldsymbol{\alpha}_n$. We perform n forward and backward passes using compressed networks $f(\boldsymbol{\omega}^*(\alpha_i), \gamma(\alpha_i))$ for $i \in \{1, ..., n\}$, where α_i is the i^{th} element of $\boldsymbol{\alpha}$. Then, the gradient update is applied. An overview of our overall algorithm is given in Algorithm 1.

Line 7 of Algorithm 1 involves sampling a point on our subspace, which determines the compression level of the corresponding network. For structured sparsity experiments, this sampling follows the sandwich rule with $n = 4$. For experiments on unstructured sparsity and quantization, we omit the sandwich rule (by setting $n = 1$) because we found the benefit to be marginal compared to the increased training cost. Thus, in this setting, our sampling function becomes $\alpha : \Omega \to [\alpha_1, \alpha_2]$, and follows a curriculum that gradually compresses the model during training. In particular, we consider a warm-up period aimed at allowing the model to learn the data's most salient features prior to applying more aggressive compression. When learning a compressible line, our pruning and quantization strengths gradually increase stochastically during training; our algorithm for this setting is provided in Algorithm 2. When learning a compressible point, our pruning and quantization strengths remain fixed at the lowest compression level during the warm-up period, and then are randomly chosen for

the remainder of training; our algorithm for this setting is provided in Algorithm 3. A toy example of what this progressive scaling looks like for LCS+L and LCS+P in the unstructured pruning setting is provided in Fig. 3.3. We expand on these curricula in the next section.

Algorithm 1 LCS Training Algorithm

Input: Network subspace $\boldsymbol{\omega}^*(\alpha)$, compression level calculator $\gamma(\alpha)$, compression function $f(\boldsymbol{\omega}, \gamma)$, stochastic sampling function $\boldsymbol{\alpha}_n \in [\alpha_1, \alpha_2]^n$, dataset $\mathcal{D}$, loss function $\mathcal{L}$.
Replace BatchNorm layers with GroupNorm.
while $\boldsymbol{\omega}^*$ has not converged **do**
    for batch $\mathcal{B}$ in $\mathcal{D}$ **do**
        Sample a vector $\boldsymbol{\alpha} \sim \boldsymbol{\alpha}_n$
        $l \leftarrow 0$
        for $\alpha \in \boldsymbol{\alpha}$ **do**
            # compute loss on batch
            $l \mathrel{+}= \mathcal{L}(f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha)), \mathcal{B})$
        end for
        backpropagate loss l
        apply gradient update to $\boldsymbol{\omega}^*$
    end for
end while
return $\boldsymbol{\omega}^*$

3.2.4 Compression Methods

We experiment with three different formulations for our compression function $f(\boldsymbol{\omega}, \gamma)$. These correspond to unstructured sparsity, structured sparsity, and quantization.

Unstructured Sparsity. In our unstructured sparsity experiments, our compression function $f(\boldsymbol{\omega}, \gamma)$ is TopK sparsity [131], which prunes a fraction γ of the weights with the smallest absolute value from each layer (we ignore the input and output layers). Our compression level calculator is $\gamma(\alpha) = 1 - \alpha$. Our stochastic sampling function $\boldsymbol{\alpha}_n$ samples a single α value uniformly along an interval $[\alpha_1, \alpha_2]$. We experiment with different settings of $[\alpha_1, \alpha_2]$ corresponding to the wide-sparsity regime and high-sparsity regime. See Sec. 3.3 for experimental details.

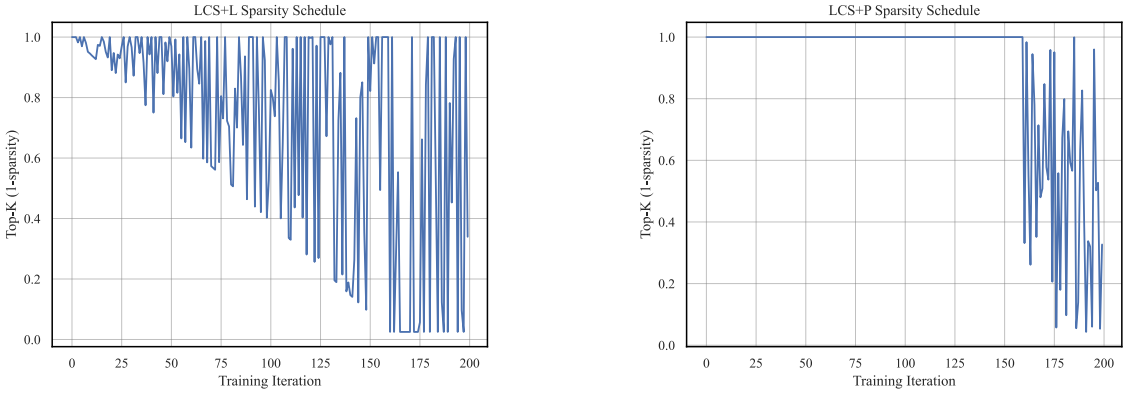
It is typical to include a warm-up phase when training models with TopK sparsity [131].

Algorithm 2 Sampling of subspace parameter for unstructured pruning and quantization for compressible lines (LCS+L).

Input: Smallest subspace parameter α_L, largest subspace parameter α_U, number of warm-up iterations i_w, current iteration i_c, biased sampling probability p.
Sample $X \sim Uniform(0, 1)$
if $X \leq p$ **then**
    $\alpha \sim Uniform(\{\alpha_L, \alpha_U\})$
else
    $\alpha \sim Uniform(\alpha_L, \alpha_U)$
end if
$df = \max(1 - i_c/i_w, 0)$
$\alpha_{out} = \alpha + (1 - \alpha)df$
return α_{out}

Algorithm 3 Sampling of subspace parameter for unstructured pruning and quantization for compressible points (LCS+P).

Input: Smallest subspace parameter α_L, largest subspace parameter α_U, number of warm-up iterations i_w, current iteration i_c.
if $i_c \leq i_w$ **then**
    $\alpha_{out} = \alpha_U$
else
    $\alpha_{out} \sim Uniform(\alpha_L, \alpha_U)$
end if
return α_{out}



(a) (b)

Figure 3.3: **LCS progressive compression for unstructured pruning.** (a) When training a compressible line via unstructured pruning, our LCS algorithm progressively increases the compression strength following Algorithm 2. (b): When training a compressible point via unstructured pruning, our LCS algorithm trains the model at the weakest sparsity level (typically zero compression) for a specified warm-up period, and then compresses the model at random compression levels as in Algorithm 3. Compression via quantization follows a similar curriculum, but chooses random bit widths rather than sparsity levels. Compression via structured pruning follows the sandwich rule [125].

In our baselines in Sec. 3.3.1, we increase the sparsity level from 0% to its final value over the first 80% of training epochs. For our method, sparsity values fall within a range, so there is no single target sparsity value to warm up to. For our point method (LCS+P), we simply train for the first 80% of training with the lowest sparsity value in our sparsity range. We finish training by sampling uniformly between the lowest and highest sparsity levels. For our line method (LCS+L), our choice of sparsity level is tied to our choice of weight-space parameters through α. We implement our warm-up by simply adjusting $\gamma(\alpha)$ to apply less sparsity early in training, warming up to our final sparsity rates over the first 80% of training.

For transformer models in particular, our LCS+P training resembles our LCS+L method whereby $\gamma(\alpha)$ applies less sparsity early in training and gradually warms up to our final sparsity rates over a fraction of the training epochs. Moreover, we do not apply sparsity to the patch embedding layer.

Structured Sparsity. In our structured sparsity experiments, our compression function $f(\boldsymbol{\omega}, \gamma)$ retains a fraction γ of the input and output channels in each layer and prunes away the rest (we ignore the input channels in the first layer, and the output channels in the last layer). Our compression level calculator $\gamma(\alpha) : [\alpha_1, \alpha_2] \to [a, b]$ is the unique affine transformation over its domain and range. Here, $[a, b]$ is the width factor range where a and b are the minimum and maximum fraction of channels retained (see Sec. 3.3 for model-specific parameter settings). Our stochastic sampling function $\boldsymbol{\alpha}_n$ samples $n = 4$ values as $[a, b, U(a, b), U(a, b)]$, where $U(a, b)$ samples uniformly in the range $[a, b]$. This choice mirrors the "sandwich rule" used in [125].

When training our method with a line in the structured sparsity setting, we do not use two sets of weights (e.g., $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$, Sec. 3.2.1) for convolutional filters. Instead, we only use two sets of weights for affine transforms in GroupNorm [122] layers. For the convolutional filters, we instead use a single set of weights, similar to our point formulation (and similar to US [125] and NS [126]). By contrast, we use two sets of weights for convolutional filters as well as for affine transforms when experimenting with unstructured sparsity (Sec. 3.3.1) and quantization (Sec. 3.3.3).

The reason for only using one set of convolutional filters in the structured sparsity setting is that the filters themselves are able to specialize, even without an extra copy of network weights. Some filters are only used in larger networks, so they can learn to identify different signals than the filters used in all subnetworks. Note that this filter specialization argument does not apply to our unstructured or quantized settings.

In preliminary experiments, we found that using a single set of weights for convolutions in our structured sparsity experiments gave a slight improvement over using two sets of weights (roughly 2% for cPreResNet20 [38] on CIFAR-10 [59]). We hypothesize that this slight difference may be attributed to the ease of learning fewer network parameters.

Quantization. In quantization experiments, our compression function $f(\boldsymbol{\omega}, \gamma)$ is affine quantization as described in [53]. Our compression level calculator is $\gamma(\alpha) = 2+6\alpha$. Our stochastic

sampling function $\boldsymbol{\alpha}_n$ samples a single α value uniformly over the set $\{1/6, 2/6, ..., 6/6\}$. This corresponds to training with bit widths 3 through 8. We avoid lower bit widths to circumvent training instabilities we encountered in baselines.

Our method applied to quantization trains without quantizing the activations for the first 80% of training, and then adds activation quantization for the remainder of training. Weights are quantized throughout training. The number of groups in our GroupNorm layers is the same as described in the unstructured setting.

3.2.5 Circumventing BatchNorm Recalibration

Previous works that train a compressible network require an additional training step to calibrate BatchNorm [52] statistics at each compression level [125, 126]. This precludes these methods from evaluating at arbitrarily fine-grained compression levels after deployment (Table 3.1). We seek to eliminate the need for recalibration or storage of statistics.

To understand the need for recalibration in previous works, recall that BatchNorm layers store the per-channel mean of the inputs, $\boldsymbol{\mu}$, and the per-channel variance of the inputs, $\boldsymbol{\sigma}^2$. The recalibration step is needed to correct $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$, which are corrupted when a network is adjusted. Adjustments that corrupt statistics include applying sparsity and quantization.

In Fig. 3.4, we analyze the inaccuracies of BatchNorm statistics for two models trained with specific compression levels and tested with a variety of inference-time compression levels. We calculate the differences between stored BatchNorm means $\boldsymbol{\mu}$ and the true mean of a batch $\hat{\boldsymbol{\mu}}$ during the test epoch of a cPreResNet20 [38] model on CIFAR-10 [59]. In the (a) and (b) rows of Fig. 3.4a, we show the distribution of mean absolute differences (MAD), $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$, across all layers of the models. Models have lower BatchNorm errors when evaluated near compression levels that match their training-time compression level. Applying mismatched levels of compression shifts the distribution of these errors away from 0. In the (c) and (d) rows, we show the average of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ across the test set for each of the BatchNorm layers.

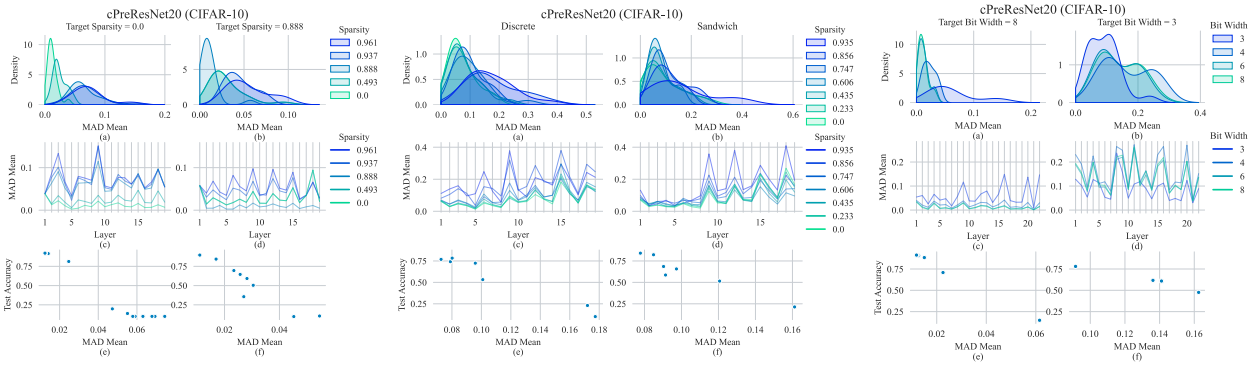


Figure 3.4: **Compressing networks corrupts their BatchNorm statistics.** *Left*: Analysis of observed batch-wise means $\hat{\boldsymbol{\mu}}$ and stored BatchNorm means $\boldsymbol{\mu}$ during testing for models trained with TopK unstructured sparsity. The models are trained with different target sparsities and evaluated with various inference-time sparsities. *Middle*: Models trained in the structured sparsity setting with the *Discrete* and *Sandwich* sampling rules (see Sec. 3.3.2). *Right*: models trained with different quantization bit widths. (a)-(b): The distribution of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ across all layers. (c)-(d): The average value of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ for individual layers. (e)-(f): The correlation between the average of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ and test set error.

Across layers, the lowest error is achieved when the compression level matches training the one used during training. In the (e) and (f) rows, we show the average of $|\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}|$ and the corresponding test set accuracy for various levels of inference-time compression. We find that the increased error in BatchNorm is correlated with decreased accuracy.

Thus, BatchNorm layers' stored statistics can become inaccurate during inference-time compression, which can lead to accuracy degradation. To circumvent the need for BatchNorm, we adjust our networks to use GroupNorm [122]. This computes an alternative normalization over g groups of channels rather than across a batch. It does not require maintaining a running average of the mean and variance across batches of input, so there are no stored statistics that can be corrupted if the network changes.

GroupNorm typically uses $g = 32$ groups, but it also includes InstanceNorm [110] (in which $g = c$, where c is the number of channels) as a special case. We use $g = c$ in structured sparsity experiments, since the number of channels is determined dynamically and is not always divisible by 32. For all other experiments, we use $g \in \{1, 8, 32\}$ depending on the

architecture as discussed in Sec. 3.2.4.

3.2.6 Real-Time Compression Analysis

Next, we justify our claim that our network compression functions $f(\boldsymbol{\omega}, \gamma)$ can be computed in real-time. Consider a network layer with weights $\boldsymbol{\omega}_l$. The computational cost of a forward pass is $\mathcal{O}(|\boldsymbol{\omega}_l|L)$, where L is the spatial dimension (in the case of a 2D convolution, $L = HW$, where H, W are the height and width of the output buffer). In the LCS+L algorithm, the cost of computing $\boldsymbol{\omega}^*$ is $\mathcal{O}(|\boldsymbol{\omega}_l|)$, which is far less than the cost of a forward pass. In the case of LCS+P, no computation is needed. Similarly, the cost of computing $\gamma(\alpha)$ is $O(1)$.

Now, consider the cost of computing $f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha))$. Our unstructured sparsity method takes $\mathcal{O}(|\boldsymbol{\omega}_l|)$ for each layer (to calculate the threshold, then discard parameters below the threshold). Our structured sparsity method takes $\mathcal{O}(1)$ for each layer, since we only need to mark each layer with the number of filters to ignore. Since most inference libraries support tensor slicing operations, we can simply pass a subset of the filters to the underlying matrix multiplication or convolution. Our quantization method [53] takes $\mathcal{O}(|\boldsymbol{\omega}_l|)$ for each layer (to calculate the affine transform parameters and apply them). In all cases, the complexity of computing $f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha))$ is at least L times lower than the cost of the convolutional forward passes, meaning our compression methods can be considered real-time.

3.3 Experiments

We present results in the domains of unstructured sparsity, structured sparsity, and quantization. We train using Pytorch [79] on NVIDIA GPUs. On CIFAR-10 [59], we experiment with the pre-activation version of ResNet20 [38] presented in the PyTorch version of the open-source code provided by [67]. We abbreviate it as "cPreResNet20."

We additionally experiment with a variety of architectures on the ImageNet [20] dataset. In particular, we present results using standard convolutional neural network (CNN) architec-

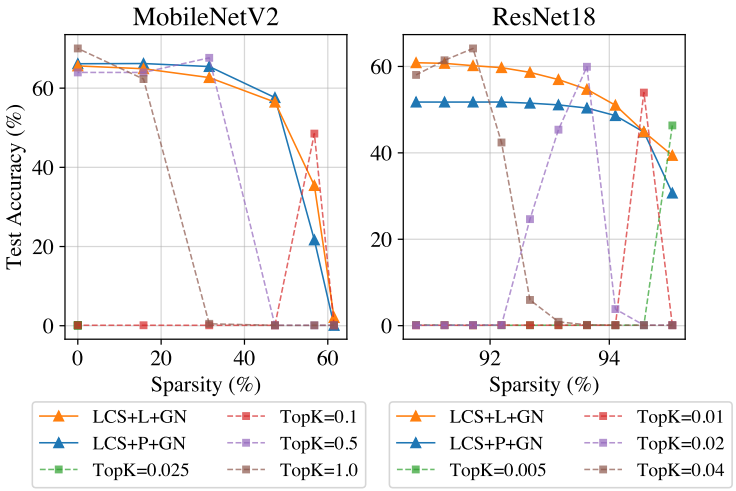


Figure 3.5: **Learning compressible subspaces with unstructured sparsity.** We compare our method for unstructured sparsity using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) to networks trained for a particular TopK target. The TopK target refers to the fraction of weights that remain unpruned during training. We train models to perform in the wide sparsity regime (left) and high sparsity regime (right).

tures: ResNet18 [38], and VGG19 [96]; lightweight CNNs: MnasNet-B1 [103], MobileNetV2 [92], MobileNetV3-Small, and MobileNetV3-Large [43]; and vision transformer models: DeiT-Ti, DeiT-S [107], and CaiT-XXS [108]. All models are trained using an input resolution of 224×224. The accuracies of our baseline models are provided in Tab. 3.3.

We train cPreResNet20 for 200 epochs and ImageNet CNNs for 90 epochs. We follow hyperparameter choices in [121] for our methods and baselines (though we do not use the β regularization they describe), with a few architecture-dependent parameters detailed in Sec. 3.5.1. For transformer models, we train for 300 epochs and follow the hyperparameter settings in [107]. Our baselines for each architecture always use the same training hyperparameters as our own methods.

3.3.1 Unstructured Sparsity

We present results for our method using MobileNetV2 and ResNet18 in Fig. 3.5. For MobileNetV2, we use an α range of $[0.025, 1]$, corresponding to a wide sparsity training regime, while for ResNet18 we use a range of $[0.005, 0.05]$, corresponding to a high sparsity training regime (because ResNet18 is overparameterized, we operate over a high sparsity range to make the accuracy-efficiency trade-off clearer).[2] Our methods use GroupNorm in the unstructured setting. For cPreResNet20, ResNet18, and VGG19, we set the number of groups to $g = 32$ (we set $g = c$ for the first few layers of cPreResNet20, because it has fewer than 32 channels). For MnasNet, MobileNetV2, MobileNetV3-Small, and MobileNetV3-Large, we use $g = 8$. Transformer models use LayerNorm (equivalent to $g = 1$).

Our method achieves a strong accuracy-efficiency trade-off in both cases. Our line subspace (LCS+L+GN) achieves a higher accuracy at high sparsities, at the expense of a lower accuracy at low sparsities. To our knowledge, efficient, adaptive, real-time compression has not been previously explored for unstructured sparsity. Thus, our baselines are networks that are trained to perform at a particular TopK sparsity level, and each network is evaluated at a variety of sparsity targets. These methods peak in accuracy near their target sparsity, but decrease sharply at higher sparsities.

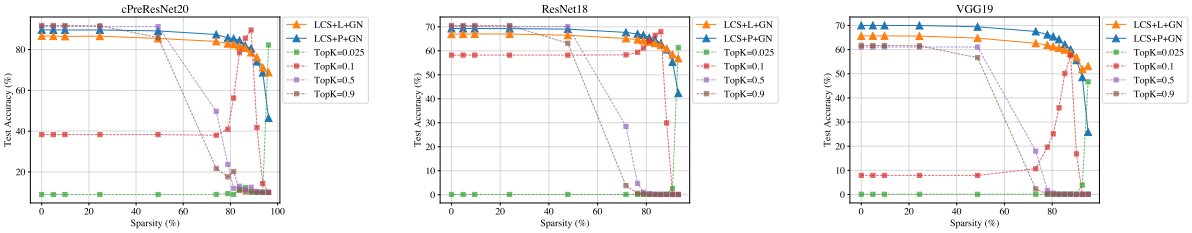
We present additional results in the unstructured wide sparsity regime in Fig. 3.6 and Fig. 3.7. For MnasNet, our α range is $[0.0325, 1]$. For the remaining models, our α range is $[0.025, 1]$. All other training details are unchanged. We find that our method is able to produce a higher accuracy over a wider range of sparsities than our baselines.

We present additional results for our method using transformer architectures in Fig. 3.8. Transformers contain LayerNorm [6] rather than BatchNorm, which does not require recalibration. Hence, we do not need to modify normalization layers in this case. As before, our

[2] In the unstructured setting, we do not compress the first and last layers of our models. Hence a compressed model's sparsity rate may not be exactly $1 - \alpha$.

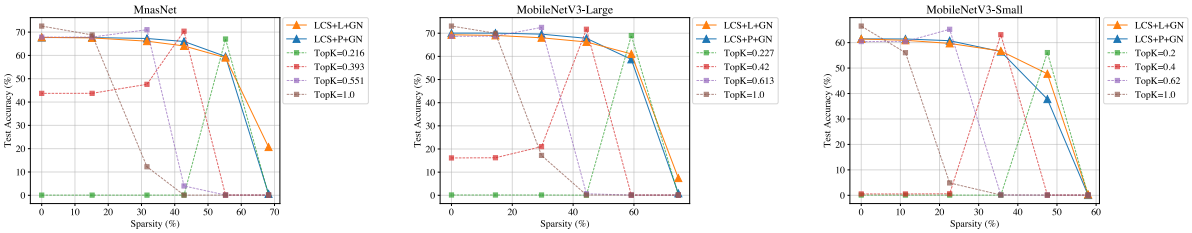
method produces a strong accuracy-efficiency trade-off across a variety of sparsity levels. Our LCS+L+LN method underperforms on DeiT models relative to LCS+P+LN, but still achieves stronger results than baselines at high sparsities. We hypothesize that the benefits of learning fewer parameters outweighs the benefits of increased capacity in this case, but we leave more investigation to future work.

We also present results for vision transformer models in the high sparsity regime in Fig. 3.9. For these models, our α range is $[0.05, 0.2]$. Additionally, for these models we use a 65% warm-up phase instead of 80% as we used for CNNs.



(a) (b) (c)

Figure 3.6: **Learning compressible subspaces with unstructured sparsity.** We compare our method for unstructured sparsity using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) to networks trained for a particular TopK target. Here we, consider compression in the wide sparsity regime, where our models can tolerate a wide range of compression ratios.



(a) (b) (c)

Figure 3.7: **Learning compressible subspaces with unstructured sparsity.** Our method for unstructured sparsity using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) compared to networks trained for a particular TopK target. Here, we apply out method to efficient networks designed to operate on edge devices.

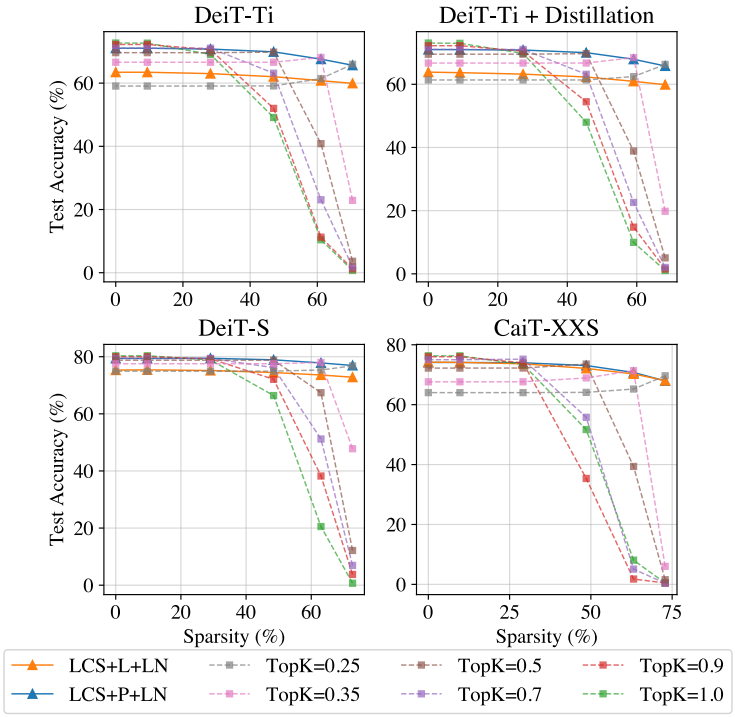


Figure 3.8: **Learning compressible subspaces with unstructured sparsity.** We compare our method for unstructured sparsity using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) to networks trained for a particular TopK target. Here, we train vision transformer networks to operate in the wide sparsity regime.

3.3.2 Structured Sparsity

We present results for our method using structured sparsity in Fig. 3.10. For all structured sparsity experiments, we use an α range of $[0, 1]$. We use a width factor range of $[0.25, 1]$ for both VGG19 and ResNet18. As discussed in Sec. 3.2.5, we use a special case of GroupNorm [122] known as InstanceNorm [110] since the number of channels in the network varies. We performed preliminary experiments with LayerNorm, but InstanceNorm achieved stronger results in our case. In the case of structured sparsity, filters are able to specialize without the need for an extra copy of network weights, since some filters are only used when the model is lightly pruned. Therefore, we only create extra copies of our InstanceNorm parameters when using LCS+L+IN, as an extra copy of network weights was unnecessary. We provide a table

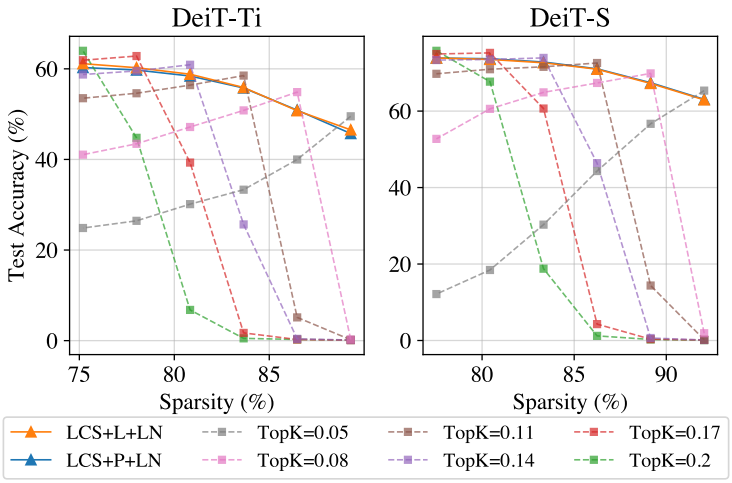


Figure 3.9: **Learning compressible subspaces with unstructured sparsity.** We compare our method for unstructured sparsity using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) compared to networks trained for a particular TopK target. Here, we train vision transformer networks to operate in the high sparsity regime.

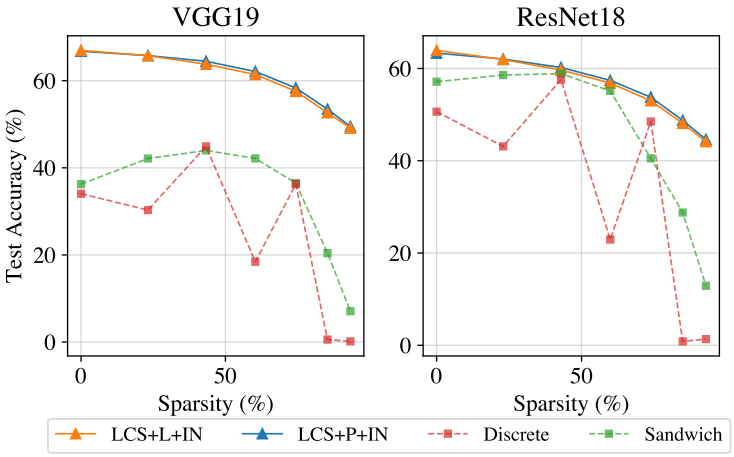


Figure 3.10: **Learning compressible subspaces with structured sparsity.** We compare our method for structured sparsity using a linear subspace (LCS+L+IN) and a point subspace (LCS+P+IN) to networks trained with *Sandwich* and *Discrete*.

demonstrating the memory, flops, and runtime of structured sparsity models in Table 3.2.

To our knowledge, efficient, adaptive, *real-time* compression has not been explored before for structured sparsity. As such, we compare our methods to two baselines that train a

network to operate at different sparsity levels. In the first method, which we denote *Discrete*, we train a network at four discrete width factors of $\{0.25, 0.5, 0.75, 1\}$. In the second method, which we denote *Sandwich*, we train a network using the sandwich rule (Sec. 3.2.3). At test time, both methods are evaluated at arbitrary sparsities. We do not perform BatchNorm calibration for either method. Note that our baselines are similar to NS and US, but our baselines operate at arbitrary width factors without BatchNorm calibration.

Models trained with our method demonstrate a strong accuracy-efficiency trade-off. By contrast, the trade-off produced by *Sandwich* peaks in the middle. We hypothesize that this is due to the sandwich rule training formulation in which sparsity levels are randomly sampled. This could cause the BatchNorm statistics to be more accurate (on average) near the middle of the sparsity range. The trade-off produced by *Discrete* contains peaks and troughs. This method trains only at discrete width factors of $\{0.25, 0.50, 0.75, 1\}$ and produces stronger accuracies at these sparsities than at sparsities that it was not explicitly trained for.

In preliminary experiments with transformers, we found that adaptive compression for structured sparsity did not converge to high accuracies. We hypothesize this may be due to inter-channel variation of transformers described in [65], but we leave more investigation to future work.

We provide additional results on lightweight networks in the structured sparsity setting in Fig. 3.11 and Fig. 3.12. Our width factor ranges for MnasNet, MobileNetV2, MobileNetV3-Small, and MobileNetV3-Large are $[0.5, 1]$, $[0.55, 1]$, $[0.57, 1]$, and $[0.44, 1]$, respectively. We observe that training with our method yields a better accuracy-efficiency trade-off for a wider range of sparsity levels.

3.3.3 Quantization

We also provide experiments for quantization. Note that in the quantization setting, there are a small number of discrete compression levels. As such, it is usually feasible to simply store

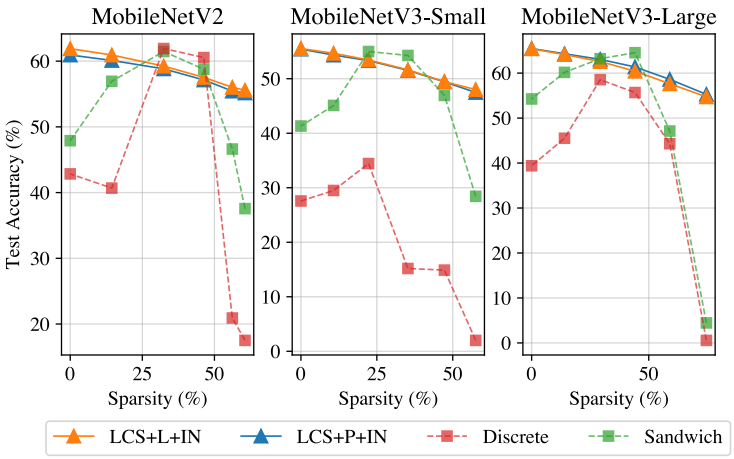


Figure 3.11: **Learning compressible subspaces with structured sparsity.** We compare our method for structured sparsity using a linear subspace (LCS+L+IN) and a point subspace (LCS+P+IN), to *Sandwich* and *Discrete*. Here, we apply our method to efficient networks designed to operate on edge devices.

extra BatchNorm parameters for all desired parameter settings before model deployment. Thus, our main purpose for experimenting in this setting is to characterize the behavior of our method under another compression technique besides pruning and to verify the versatility of our method.

We present results for our method in Fig. 3.13, comparing to baseline models trained at a fixed bit width and evaluated at a variety of bit widths. Generally, baselines achieve high accuracy when they are evaluated with the bit width used for training, and reduced accuracy at other bit widths. In contrast, our method using a linear subspace (LCS+L+GN) achieves high accuracy at all bit widths, matching or exceeding accuracies of individual networks trained for target bit widths. In the case of VGG19, we found that our accuracy even exceeded the baselines. We believe part of the increase is due to GroupNorm demonstrating improved results on this network compared to BatchNorm (which does not happen with ResNets, as reported in [122]). We provide quantization results for ResNet18 in Fig. 3.14. We find that our models approach the accuracy of models trained at a single bit width, and our models generalize better to other bit widths.

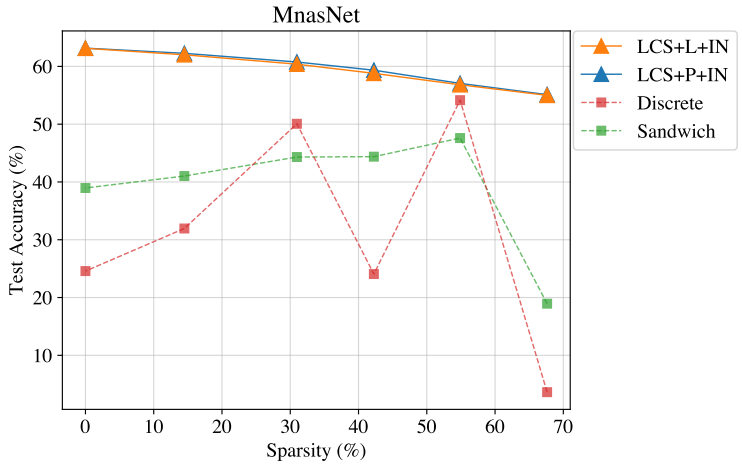


Figure 3.12: **Learning compressible subspaces with structured sparsity.** We compare our method for structured sparsity using a linear subspace (LCS+L+IN) and a point subspace (LCS+P+IN) applied to MnasNet, to MnasNet trained with *Sandwich* and *Discrete*.

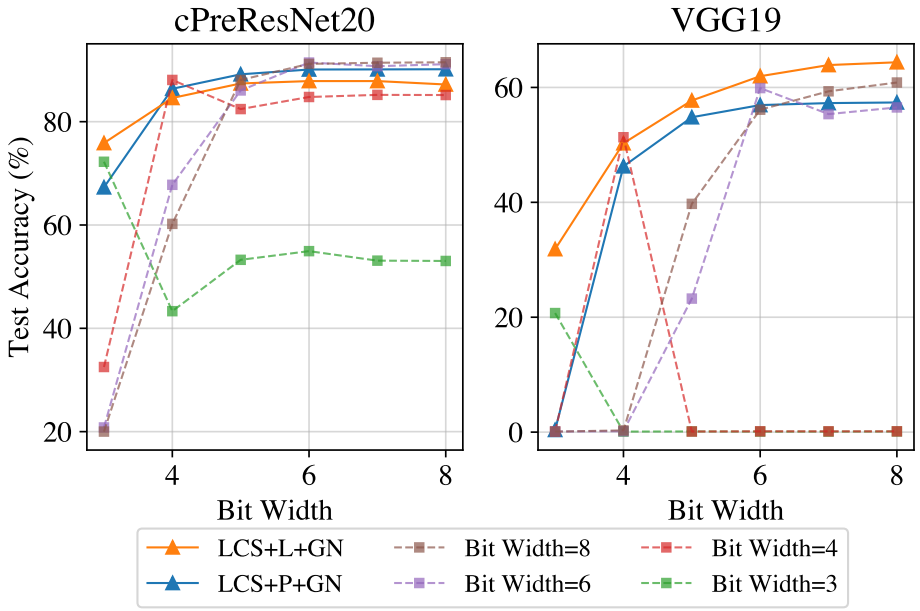


Figure 3.13: **Learning compressible subspaces with quantization.** We compare our method for quantization using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) to networks trained to operate at a particular bit width target.

Table 3.2: **Runtime characteristics for structured sparsity.** Note that models of a particular architecture and sparsity level all have the same memory, FLOPS, and runtime, so we only report one value. Runtime was measured on a MacBook Pro (16-inch, 2019) with a 2.6 GHz 6-Core Intel Core i7 processor and 16GB 2667 MHz DDR4 RAM. Memory consumption refers to the size of model weights in the currently executing model.

cPreResNet20 (CIFAR-10)	Sparsity (%)	0	43.491	60.614	74.655	85.614	93.491
	FLOPS ($\times 10^6$)	33.75	19.07	13.29	8.55	4.85	2.2
	Memory (MB)	0.87	0.49	0.34	0.22	0.12	0.06
	Runtime (ms)	3.13	2.64	2.09	1.83	1.64	1.28
	Acc (LCS+P+IN)	87.51	86.07	84.46	**82.02**	78.39	**75.96**
	Acc (LCS+L+IN)	**88.49**	**86.22**	**84.54**	81.92	**78.73**	75.25
	Acc (*Sandwich*)	70.62	83.13	81.11	62.18	40.04	21.81
	Acc (*Discrete*)	72.87	75.46	57.09	70.07	16.86	19.76
ResNet18 (ImageNet)	Sparsity (%)	0.0	42.91	59.89	73.88	84.89	92.91
	FLOPS ($\times 10^6$)	1814.1	1042.66	736.42	483.16	282.89	135.61
	Memory (MB)	46.72	26.67	18.74	12.2	7.06	3.31
	Runtime (ms)	45.85	30.34	22.51	14.31	9.84	6.02
	Acc (LCS+P+IN)	63.32	60.21	**57.42**	**53.77**	**48.75**	**44.62**
	Acc (LCS+L+IN)	**63.93**	59.66	56.84	53.00	48.11	44.14
	Acc (*Sandwich*)	58.91	**60.39**	53.76	44.72	22.51	8.34
	Acc (*Discrete*)	50.63	57.58	22.93	48.52	0.84	1.34
VGG19 (ImageNet)	Sparsity (%)	0.0	43.28	60.35	74.37	85.35	93.28
	FLOPS ($\times 10^6$)	19533.52	11008.56	7656.48	4911.33	2773.1	1241.81
	Memory (MB)	82.12	46.58	32.56	21.04	12.03	5.52
	Runtime (ms)	388.49	246.81	172.64	105.77	60.0	29.55
	Acc (LCS+P+IN)	66.77	**64.47**	**62.11**	**58.35**	**53.45**	**49.5**
	Acc (LCS+L+IN)	**66.97**	63.79	61.42	57.57	52.66	49.11
	Acc (*Sandwich*)	36.27	43.99	42.17	36.5	20.42	7.07
	Acc (*Discrete*)	34.05	44.91	18.44	36.26	0.57	0.14

3.3.4 Confirming the Linear Subspace Accuracy-Efficiency Trade-Off

We conclude our experiments by providing additional experimental evidence that our linear subspace method (LCS+L) trains a subspace specialized for high accuracy at one end and high efficiency at the other end. In Fig. 3.15, we plot the validation accuracy along our subspace, as well as the validation accuracy along our subspace when compressing with $\tilde{f}(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha)) \equiv f(\boldsymbol{\omega}^*(\alpha), \gamma(1-\alpha))$. In other words, the weights that were trained for low compression levels are evaluated with high compression levels, and vice versa. We see that this leads to a large drop in accuracy, confirming that our method has conditioned one side of

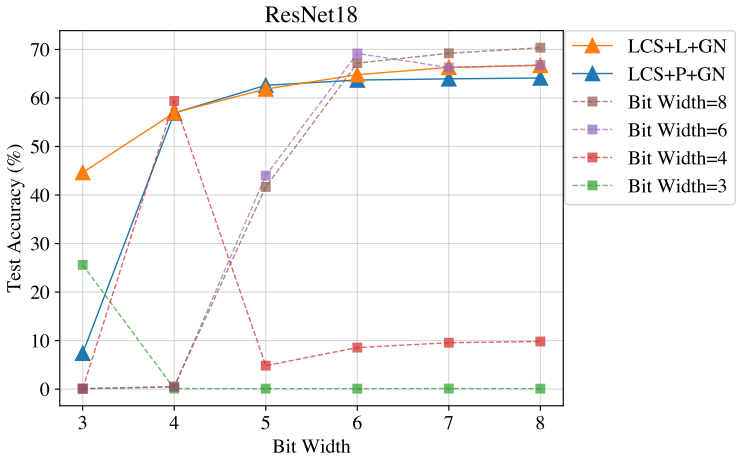


Figure 3.14: **Learning compressible subspaces with quantization.** We compare our method for quantization using a linear subspace (LCS+L+GN) and a point subspace (LCS+P+GN) to networks trained to operate at a particular bit width target.

the line to achieve high accuracy at high sparsities, and the other side of the line to achieve high accuracy at low sparsities.

3.4 Discussion

In this chapter, we presented a method for learning a compressible subspace of neural networks. Our method produces models that can be deployed on-device and used for efficient, adaptive, real-time model compression. Once deployed, our model can be compressed in real-time, to any compression level, without retraining, and without specifying the compression levels before deployment. Additionally, our LCS+P method incurs no parameter overhead. We show that our generic algorithm outperforms baselines in the domains of unstructured sparsity, structured sparsity, and quantization. To train these models, we employ a dynamic compression scheme during training that adapts the model to operate at different compression levels.

Our compressible subspaces yield several positive real-world impacts. Devices equipped

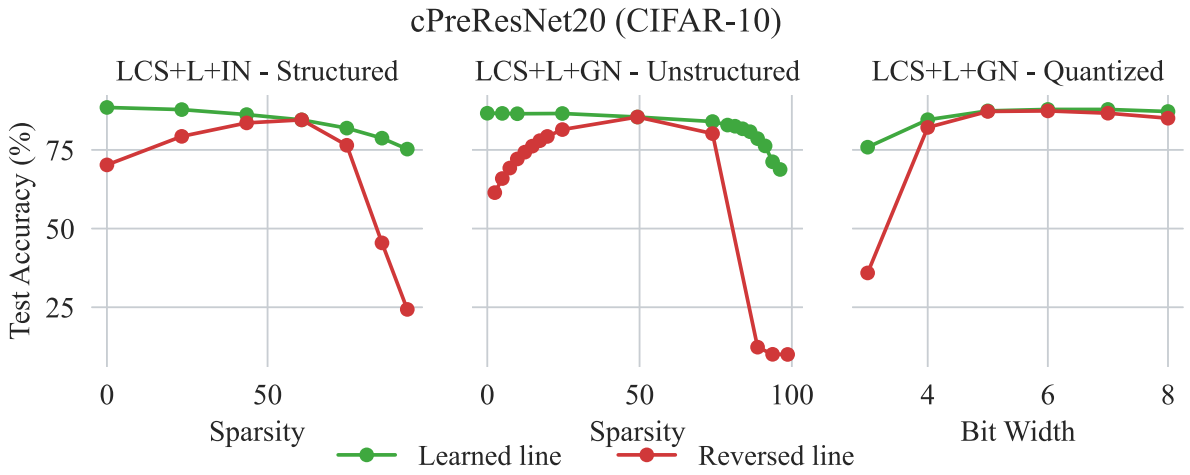


Figure 3.15: **Our linear compressible subspace optimizes one end for accuracy and one end for efficiency,**. We perform standard evaluation of a linear subspace with network $f(\boldsymbol{\omega}^*(\alpha), \gamma(\alpha))$ (Learned line), and evaluation when evaluating with reversed compression levels, $f(\boldsymbol{\omega}^*(\alpha), \gamma(1-\alpha))$ (Reversed line). Evaluation with the reversed line performs significantly worse, indicating that one end is optimized for accuracy, while the other for efficiency.

with our models can dynamically adjust their energy consumption by efficiently compressing models according to the device's available resources. Additionally, our method circumvents the need for training multiple models tailored to multiple devices. Increasingly larger DNNs are inaccessible to older devices and devices with lower compute, and training models specific to each device's hardware constraints would be prohibitively expensive. Using our method, users can train a single model and efficiently compress it to a particular device's hardware constraints prior to deployment.

While we have focused on CNNs and Transformers for vision, an exciting future work is extending these results to large language models.

3.5 Appendix

3.5.1 Training Details

Our CNNs warm up to an initial learning rate of 0.1 (0.045 for MobileNetV2) over 5 epochs, which then decays to 0 over 85 epochs (or 195 for cPreResNet20) using a cosine schedule. We use a batch size of 128 on a single GPU for cPreResNet20 and ResNet18. We use the version of VGG19 provided by [67]. This implementation modifies VGG19 slightly by adding BatchNorm layers and removing the last two fully connected layers. For VGG19, we use a batch size of 256 with 4 GPUs. We train MnasNet, MobileNetV2, MobileNetV3-Small, and MobileNetV3-Large with a batch size of 128 using two GPUs. We train transformer models using a batch size of 1024 with 8 GPUs. For cPreResNet20, VGG19, and ResNet18, we use a weight decay of 5×10^{-4}. For MobileNetV2, we use a weight decay of 4×10^{-5}, and 10^{-5} for MnasNet, MobileNetV3-Small, and MobileNetV3-Large.

3.5.2 Baseline Model Accuracies

As a reference, we provide the final test accuracies of our models trained with BatchNorm and without any compression in Tab. 3.3.

3.5.3 Memory and FLOPS Footprint of Compressed models

In Tabs. 3.4 to 3.6 we provide memory and compute measurements of our models compressed under different compression levels in the unstructured high sparsity regime, unstructured wide sparsity regime, and quantization, respectively.

Table 3.3: **Accuracies of baseline (non-compressed) models with BatchNorm.** We provide the accuracies of all models used when no compression is applied and are trained with BatchNorm. cPreResNet20 is trained on CIFAR-10 and all other models on ImageNet.

Model	BatchNorm Baseline Accuracy (%)
cPreResNet20	91.69
ResNet18	70.72
VGG19	62.21
MnasNet-B1	72.58
MobileNetV2	70.03
MobileNetV3-Small	66.5
MobileNetV3-Large	73.09
DeiT-Ti	72.7
DeiT-Ti + Distillation	73.1
DeiT-S	80.3
CaiT-XXS	76.02

Table 3.4: **Runtime characteristics for unstructured sparsity in the high sparsity regime**. Note that models of a particular architecture and sparsity level all have the same runtime characteristics (memory and FLOPS), so we only report one value. Runtime was not measured because it requires specialized hardware. So, we follow the standard practice of only reporting memory and flops. Memory consumption refers to the size of *nonzero* model weights in the currently executing model.

cPreResNet20 (CIFAR-10)	Sparsity (%)	95.66	96.15	96.64	97.14	97.63	98.12
	FLOPS ($\times 10^6$)	1.46	1.29	1.13	0.96	0.79	0.63
	Memory (MB)	0.04	0.03	0.03	0.02	0.02	0.02
	Acc (LCS+P+GN)	70.18	69.22	67.74	**63.78**	54.91	24.92
	Acc (LCS+L+GN)	**75.53**	**72.30**	67.02	52.86	39.63	34.12
	Acc (TopK=0.04)	14.83	12.83	10.8	9.66	10.01	9.79
	Acc (TopK=0.02)	10.25	10.53	**78.7**	10.56	10.05	10.09
	Acc (TopK=0.01)	10.02	9.97	9.96	10.64	**59.2**	10.79
	Acc (TopK=0.005)	10.0	10.08	9.81	10.03	10.0	**41.44**
ResNet18 (ImageNet)	Sparsity (%)	92.67	93.15	93.62	94.1	94.58	95.06
	FLOPS ($\times 10^6$)	169.42	160.94	152.46	143.98	135.51	127.03
	Memory (MB)	3.42	3.2	2.98	2.76	2.53	2.31
	Acc (LCS+P+GN)	51.5	51.1	50.37	48.62	44.8	30.69
	Acc (LCS+L+GN)	**58.63**	**56.96**	54.70	**51.02**	44.87	39.41
	Acc (TopK=0.04)	5.96	0.9	0.18	0.11	0.1	0.1
	Acc (TopK=0.02)	24.66	45.37	**59.92**	3.84	0.1	0.11
	Acc (TopK=0.01)	0.12	0.1	0.1	0.11	**53.95**	0.11
	Acc (TopK=0.005)	0.1	0.12	0.11	0.1	0.11	**46.35**

Table 3.5: **Runtime characteristics for unstructured sparsity in the wide sparsity regime**. Note that models of a particular architecture and sparsity level all have the same runtime characteristics (memory and FLOPS), so we only report one value. Runtime was not measured, because it requires specialized hardware (so most unstructured pruning works report memory and flops). Memory consumption refers to the size of *nonzero* model weights in the currently executing model.

cPreResNet20 (CIFAR-10)	Sparsity (%)	0.0	49.31	86.29	91.22	93.68	96.15
	FLOPS ($\times 10^6$)	33.75	17.11	4.62	2.96	2.13	1.29
	Memory (MB)	0.87	0.44	0.12	0.08	0.05	0.03
	Acc (LCS+P+GN)	89.64	89.34	**82.34**	75.24	67.55	47.1
	Acc (LCS+L+GN)	86.65	85.45	80.78	**76.27**	**71.22**	68.78
	Acc (TopK=0.9)	**91.66**	83.17	10.54	10.0	9.76	10.0
	Acc (TopK=0.5)	91.16	**91.17**	10.64	10.14	10.0	10.0
	Acc (TopK=0.1)	40.74	40.74	78.56	39.54	11.67	10.26
	Acc (TopK=0.025)	9.64	9.64	10.65	10.0	9.98	**82.15**
ResNet18 (ImageNet)	Sparsity (%)	0.0	47.77	83.59	88.37	90.76	93.15
	FLOPS ($\times 10^6$)	1814.1	966.32	330.49	245.72	203.33	160.94
	Memory (MB)	46.72	24.4	7.66	5.43	4.32	3.2
	Acc (LCS+P+GN)	69.25	69.12	64.53	60.36	55.58	41.48
	Acc (LCS+L+GN)	66.94	66.49	63.20	**60.96**	**58.44**	56.83
	Acc (TopK=0.9)	**70.57**	63.17	0.1	0.1	0.1	0.1
	Acc (TopK=0.5)	70.15	**70.15**	0.24	0.17	0.1	0.12
	Acc (TopK=0.1)	58.21	58.21	**66.44**	29.93	0.24	0.1
	Acc (TopK=0.025)	0.12	0.12	0.13	0.17	2.64	**61.33**
VGG19 (ImageNet)	Sparsity (%)	0.0	48.75	85.31	90.19	92.62	95.06
	FLOPS ($\times 10^6$)	19533.52	9822.65	2539.51	1568.42	1082.88	597.34
	Memory (MB)	82.12	42.09	12.06	8.06	6.06	4.06
	Acc (LCS+P+GN)	**70.0**	**69.45**	**62.11**	55.62	48.58	25.87
	Acc (LCS+L+GN)	65.64	64.76	59.93	**56.80**	**51.89**	**53.15**
	Acc (TopK=0.9)	61.72	56.67	0.1	0.0	0.0	0.1
	Acc (TopK=0.5)	61.07	61.07	0.15	0.09	0.06	0.1
	Acc (TopK=0.1)	7.91	7.91	50.13	16.8	0.13	0.09
	Acc (TopK=0.025)	0.09	0.09	0.1	0.16	3.91	46.66

Table 3.6: **Runtime characteristics for quantized models.** Note that models of a particular architecture and quantization bit width all use the same memory, so we only report one value. Runtime was not measured, because it requires specialized hardware. Memory consumption refers to the size of model weights in the currently executing model.

	Bit Widths	8	7	6	5	4	3
	Memory (MB)	0.22	0.19	0.17	0.14	0.11	0.08
	Acc (LCS+P+GN)	89.97	90.0	89.88	**89.26**	86.25	65.26
cPreResNet20	Acc (LCS+L+GN)	87.20	87.86	87.86	87.40	84.59	**75.86**
(CIFAR-10)	Acc (Bit Width=8)	**91.36**	**91.02**	90.47	87.98	65.91	16.65
	Acc (Bit Width=6)	91.07	90.89	**91.26**	87.12	63.1	18.78
	Acc (Bit Width=4)	84.93	84.65	84.77	82.37	**88.22**	25.19
	Acc (Bit Width=3)	55.71	55.56	57.01	55.66	44.83	73.89
	Memory (MB)	11.69	10.23	8.77	7.31	5.84	4.38
ResNet18	Acc (LCS+P+GN)	63.59	63.51	63.15	**61.82**	55.89	5.48
(ImageNet)	Acc (LCS+L+GN)	66.72	66.30	64.80	61.84	56.96	**44.63**
	Acc (Bit Width=8)	**70.36**	**69.2**	67.18	41.67	0.54	0.08
	Acc (Bit Width=6)	66.8	66.26	**69.17**	44.0	0.43	0.1
	Acc (Bit Width=4)	9.82	9.57	8.57	4.84	**59.39**	0.1
	Acc (Bit Width=3)	0.1	0.12	0.09	0.1	0.12	25.61
	Memory (MB)	20.542	17.974	15.406	12.839	10.271	7.703
VGG19	Acc (LCS+P+GN)	57.83	57.74	57.24	55.64	47.11	0.25
(ImageNet)	Acc (LCS+L+GN)	**64.38**	**63.89**	**61.95**	**57.72**	50.28	**31.85**
	Acc (Bit Width=8)	60.85	59.32	56.13	39.73	0.3	0.09
	Acc (Bit Width=6)	56.52	55.35	59.89	23.24	0.16	0.1
	Acc (Bit Width=4)	0.13	0.13	0.14	0.11	**51.33**	0.1
	Acc (Bit Width=3)	0.12	0.1	0.1	0.12	0.09	20.72

CHAPTER 4

Curriculum-Based Multiscale Training[1]

In Chapter 2 and Chapter 3, we explored modulating the capacity of models during training by dynamically compressing them via pruning or quantization. In this chapter, we shift our attention to modulating the data complexity during training. In particular, we perform a detailed analysis of multiscale variable batch size data samplers [70], which randomly sample an input resolution at each training iteration and dynamically adjust their batch size accordingly. Such samplers have been shown to improve model accuracy beyond standard training with a fixed batch size and resolution, though it is not clear why this is the case. We explore the properties of these data samplers by performing extensive experiments on ResNet-101 and validate our conclusions across multiple architectures, tasks, and datasets. We show that multiscale samplers behave as implicit data regularizers and accelerate training speed. Compared to models trained with single-scale samplers, we show that models trained with multiscale samplers retain or improve accuracy, while being better-calibrated and more robust to scaling and data distribution shifts.

We additionally introduce a multiscale variable batch sampler with a simple curriculum that progressively grows resolutions throughout training, which further reduces training compute while retaining accuracy. We further show that the benefits of multiscale training extend to detection and instance segmentation tasks.

Building on the effectiveness of curriculum-based multiscale resolution training for enhancing training efficiency, we explore additional improvements by imposing a curriculum on the

[1]Work completed during an internship at Apple.

model architecture. Drawing inspiration from Neural Architecture Search (NAS) and network morphisms [117, 30], we explore progressive model growth during training, combined with sample resolution expansion, as a means of improving training efficiency further. This is in contrast to the dynamic model compression we considered in Chapter 2 and Chapter 3, where we gradually compressed the model throughout training. In this setting, as we progressively increase the data complexity by expanding sample resolutions, we simultaneously expand the model's capacity to handle the growing task complexity. Our results demonstrate that this compound scaling yields models with higher accuracy compared to baseline models trained within a similar budget.

4.1 Overview of Image Data Samplers

When training a deep neural network $\mathcal{N}$ for vision tasks, gradient descent is performed over a dataset $\mathcal{D}$ for a specific number of epochs, E. Throughout training, $\mathcal{N}$ is fed batches $\mathcal{B} \subset \mathcal{D}$ with shape $|\mathcal{B}| \triangleq (B, C, H, W)$ where B denotes the batch size, C is the number of input channels, and H, W are the image height and width, respectively. Typically, the batch shape $|\mathcal{B}|$ is fixed throughout training so that at each iteration, the model is fed a fixed number of samples at a fixed resolution. Throughout this chapter, we refer to the protocol that fetches data batches at each training iteration as the *data sampler*. We discuss several data sampling strategies and their trade-offs below.

Single-scale fixed batch size. The single-scale fixed batch size (SSc-FBS) data sampler is the default sampling procedure that is implemented across many deep learning libraries and frameworks [120, 80, 1]. In this setting, the batch shape at iteration t, $|\mathcal{B}|_t$, remains fixed so that $|\mathcal{B}|_t = (B, C, H, W)$ for all iterations t.

Multiscale fixed batch size. SSc-FBS allows the model to learn representations at single scale. However, in the real world, objects appear at different scales. Multiscale samplers

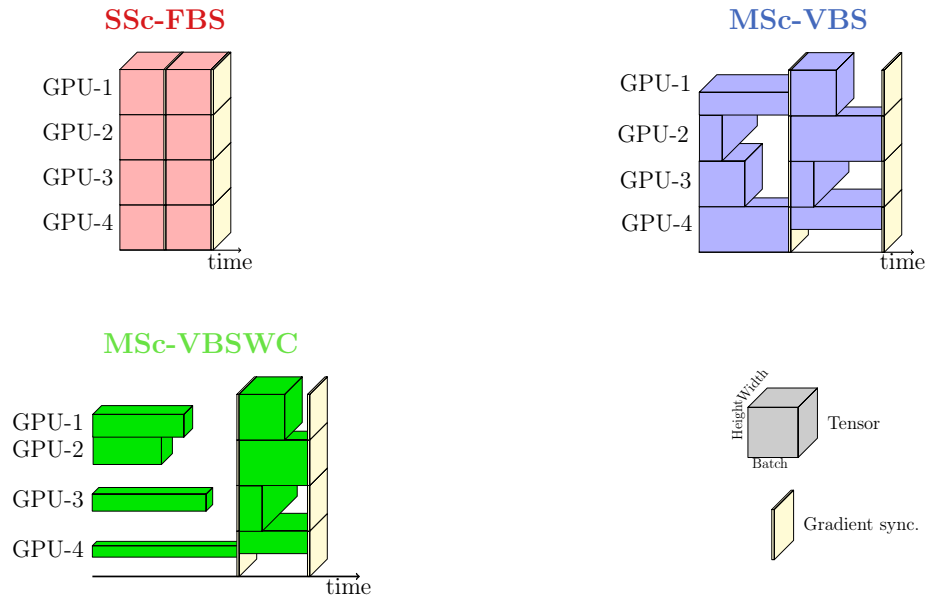


Figure 4.1: **Single-scale fixed batch size (SSc-FBS) vs multiscale variable batch size (MSc-VBS) vs multiscale variable batch size with curriculum (MSc-VBSWC) samplers.** In the SSc-FBS sampler, at each training iteration, each GPU receives a batch of data that is the same shape throughout training. In the MSc-VBS sampler, at each training iteration, each GPU will randomly sample a training resolution and dynamically adjust the batch size to use a large batch for small resolutions, and a small batch for large resolutions. In the MSc-VBSWC sampler, the sample resolutions expand throughout the course of training while leveraging the dynamic batch sizes of the MSc-VBS sampler.

with a fixed batch size (MSc-FBS) [88, 70] address this shortcoming by training a model at multiple input scales. More specifically, MSc-FBS randomly samples a batch with shape $|\mathcal{B}|_t = (B, C, H_t, W_t)$ at each training iteration t, where (H_t, W_t) is drawn randomly from a set of possible spatial resolutions $\mathcal{S} = \{(H_1, W_1), (H_2, W_2), \ldots, (H_n, W_n)\}$.

Multiscale with variable batch size. One drawback of MSc-FBS is its fixed batch size selection. Because the batch size B is fixed, if a large resolution $(H_t, W_t) \in \mathcal{S}$ is chosen, an out-of-memory (OOM) error can occur. Thus, the choices of (H_t, W_t) are limited by B. Conversely, a small spatial resolution may lead to under-utilization of computational resources. To address this, multiscale variable batch size samplers (MSc-VBS) were introduced in [72, 70]. This sampler circumvents the compute bottleneck of the MSc-FBS sampler by

dynamically adjusting the batch size B according to the resolution (H_t, W_t). In particular, a "reference" batch shape (B, C, H, W) is first defined. Then, at training iteration t, the MSc-VBS sampler samples a batch with shape $|\mathcal{B}|_t = (B_t, C, H_t, W_t)$ where $(H_t, W_t) \in \mathcal{S}$ are sampled randomly and $B_t = \frac{HW}{H_t W_t} B$. Thus, MSc-VBS will use a larger batch size B_t when the resolution (H_t, W_t) is small, and a smaller B_t when the resolution is large. The MSc-VBS sampler dynamically controls the batch size to offset the change in compute due to the spatial resolution, allowing it to efficiently utilize the underlying hardware and avoid OOM errors during training. Figure 4.1 visualizes single-scale and multiscale variable batch samplers. Next, we augment MSc-VBS with a curriculum to progressively expand training resolutions during training.

4.1.1 Curriculum-Based Multi-Scale Variable Batch Size Sampler

Motivated by curriculum and progressive learning [10, 45, 105], we extend MSc-VBS with a curriculum which we call the Multiscale Variable Batch Size Sampler with curriculum (MSc-VBSWC). Similar to MSc-VBS, we first consider a set of possible spatial resolutions $\mathcal{S}$ and a reference batch shape (B, C, H, W). In this setting, $\mathcal{S}$ serves as a "reference" pool of spatial resolutions that our sampler expands to. In particular, at each training epoch e, we consider a pool of sample resolutions $s(e) = \{(\rho(e) \cdot H_1, \rho(e) \cdot W_1), \ldots, (\rho(e) \cdot H_n, \rho(e) \cdot W_n)\}$ where $\rho(e)$ is a monotonically increasing function such that $\rho(0) = \rho_0 \in (0, 1]$, $\tau \in (0, 1]$, and $\rho(\tau E) = 1$. E is the total number of training epochs, ρ_0 represents the initial compression factor, and τ determines the fraction of epochs it takes for $s(e)$ to expand to $\mathcal{S}$. In words, at the start of training, we sample from a "compressed" form of $\mathcal{S}$ where each spatial resolution is scaled by ρ_0, then, as training progresses, the set of spatial resolutions we sample from, $s(e)$, expands to $\mathcal{S}$ over the first τE epochs. For example, for $\rho_0 = 0.75$, $\tau = 0.5$, and $E = 600$, training begins by sampling from $s(0) = \{(0.75H, 0.75W) : (H, W) \in \mathcal{S}\}$, and for epochs $e \geq 300$, $s(e) \equiv \mathcal{S}$. We found $\rho_0 = 0.75$ and $\tau = 0.5$ to work well, and use these hyperparameters for all MSc-VBSWC experiments.

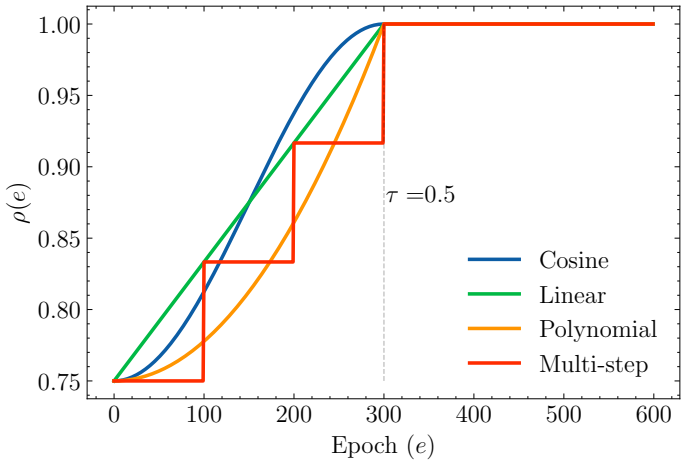


Figure 4.2: **MSc-VBSWC expansion schedules.** We experiment with four expansion schedules, $\rho(e)$, for MSc-VBSWC that control the rate at which spatial resolutions expand throughout training. We use initial compression factor $\rho_0 = 0.75$, and expansion period $\tau = 0.5$.

Sampler	Peak GPU Memory	Training FLOPs	Optimization updates	Training time	Top-1 Accuracy
SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	81.31
MSc-VBSWC (Linear)	1.72×	0.70×	0.66×	0.91×	81.34
MSc-VBSWC (Cosine)	1.75×	0.70×	0.66×	**0.84**×	**81.53**
MSc-VBSWC (Polynomial)	1.61×	**0.67**×	**0.63**×	0.88×	81.29
MSc-VBSWC (Multi-step)	1.32×	**0.67**×	0.64×	0.87×	81.22

Table 4.1: **MSc-VBSWC with a cosine schedule has a stronger accuracy-efficiency trade-off**. We train a ResNet-101 architecture on ImageNet with linear, cosine, polynomial, and multi-step expansion schedules. Due to its stronger accuracy-efficiency trade-off, we adopt the cosine schedule for all of our MSc-VBSWC experiments.

The rate at which $s(e)$ expands is controlled by the function $\rho(e)$, which we call the "curriculum." We consider four schedules for $\rho(e)$: 1) linear, 2) cosine, 3) polynomial, and 4) multi-step, as illustrated in Fig. 4.2. We compare the performance of these different schedules in Tab. 4.1, where we find that a cosine schedule offers the strongest accuracy-efficiency trade-off. As such, all our MSc-VBSWC experiments expand the training resolutions according to this schedule.

4.2 Why Train With Multiscale Samplers?

In this section, we analyze the properties of the different samplers in Sec. 4.1 and compare their performances. We first establish their benefits over single-scale training and show that multiscale variable batch size samplers train faster and lead to more accurate models (Sec. 4.2.1). To further understand the properties of multiscale samplers, we study their robustness and empirically show that they are better calibrated (Sec. 4.2.2). Finally, we also study their regularization behavior (Sec. 4.2.3).

We study these properties on the ImageNet [20] dataset using the ResNet-101 architecture [38] and follow the recipes in [70, 71] for training and evaluating the models. For results on other architectures, see Sec. 4.3. Additionally, in Sec. 4.6.2, we provide a discussion on the effect that the number of GPUs has on the performance of single and multiscale samplers, as well as uncertainty measurements.

4.2.1 Faster Training

Metrics: We use the following metrics to measure the training speed: (1) **Training FLOPs** measures computational complexity during training while being hardware-agnostic. Generally, a data sampler with fewer training FLOPs is preferable. (2) **Optimization updates** measures the number of training iterations it takes to train a model. Fewer updates is generally more efficient and favorable. (3) **Peak GPU memory** measures the maximum memory that is required for training a model. Data samplers with lower memory are preferable. (4) **Training time** measures the wall-clock time taken to train the model. Data samplers with lower training times are desirable. Note that, in these experiments, we used the same hardware for training all models, as well as the same number of data workers and GPUs. However, a careful tuning of hardware resources may vary the training time significantly. Such experiments are beyond the scope of this chapter.

Sampler	Peak GPU Memory	Training FLOPs	Optimization updates	Training time	Top-1 Accuracy (%)
SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	81.31
MSc-FBS	2.23×	1.15×	1.00×	1.15×	81.01
MSc-VBS	1.22×	0.77×	0.77×	0.92×	**81.66**
MSc-VBSWC	1.75×	**0.70**×	**0.66**×	**0.84**×	81.53

Table 4.2: **Training with a multiscale variable batch size sampler promotes faster training.** Here we train a ResNet-101 model on ImageNet with single-scale (SSc-FBS) and multiscale (MSc-FBS, MSc-VBS, MSc-VBSWC) samplers. Multiscale training retains the accuracy of the model trained with SSc-FBS while training faster and being more computationally efficient.

Results: Table 4.2 compares the performance of different samplers. In general, multiscale samplers (MSc-FBS, MSc-VBS, and MSc-VBSWC) are able to match the performance of the single-scale sampler (SSc-FBS). Multiscale samplers with variable batch sizes (MSc-VBS and MSc-VBSWC) also reduce the training FLOPs and optimization updates significantly, resulting in faster training. This is because multiscale variable batch samplers adjust the batch size depending on the input spatial resolution (larger batch sizes are used for smaller spatial resolutions and vice-versa). Compared to training with SSc-FBS, training with MSc-VBSWC reduces training FLOPs, optimization updates, and wall clock time by 30%, 34%, and 16% respectively, without compromising accuracy. Due to the large computational requirements of the MSc-FBS sampler, we focus our multiscale sampler analysis on the MSc-VBS and MSc-VBSWC samplers in the rest of our experiments.

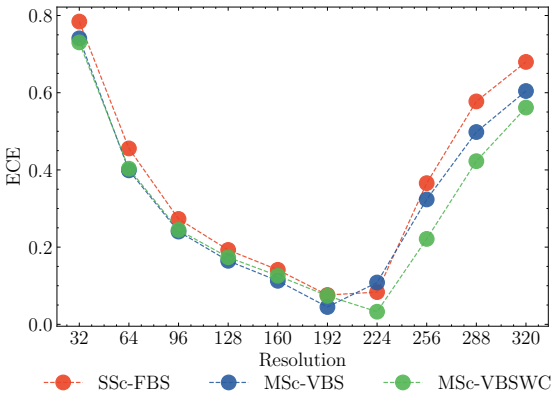
4.2.2 Robustness

In Sec. 4.2.1, we observe that multiscale samplers can preserve or slightly improve accuracy and accelerate training. In this section, we assess the robustness of models trained with multiscale samplers compared to those trained with single-scale samplers.

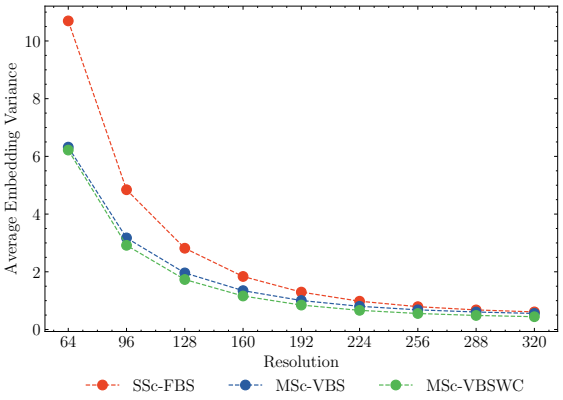
Metrics: We study the robustness of models trained with multiscale samplers in five ways. (1) **Accuracy on standard robustness benchmarks.** We evaluate the top-1 accuracy on three standard benchmarks for estimating a model's robustness: ImageNet-A [40], ImageNet-R

[39], and ImageNetV2 [87]. (2) **Expected calibration error.** A classification model is well-calibrated if its predicted class probabilities capture the true underlying distribution correctly. This is a desirable property as it allows users of such models to more reliably accept or reject the models' outputs. A standard metric to measure the degree of a model's miscalibration is the expected calibration error (ECE) [73, 61], which measures the discrepancy between the model's confidence and the accuracy. Hence, a well-calibrated model will have a low ECE. (3) **Image embedding variance.** An image embedding is a vector representation of an image that captures its semantic information. A model that produces low-variance embeddings is robust and selective [29, 95]. (4) **Robustness to scale changes.** Generally speaking, a discriminative model should be robust to scale changes. To assess this, we evaluate our models across a broad range of input resolutions. (5) **Entropy Skewness.** The entropy of a classification model's predicted class distribution reflects the model's uncertainty in its prediction. By computing the entropy for each image in the validation set at each training epoch, we can obtain empirical entropy distributions that we can monitor over the course of training. Measuring the skewness of these distributions allows us to investigate the rate at which a model becomes more certain in its predictions.

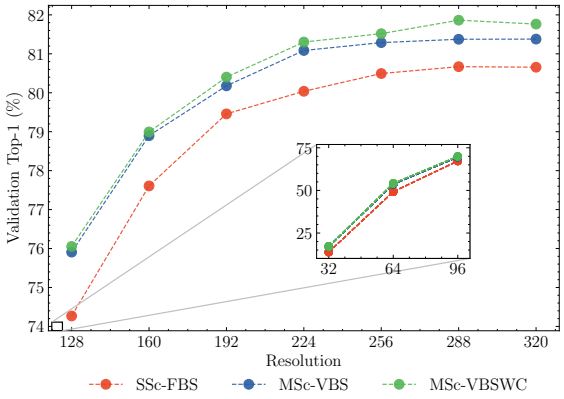
Results: (1) Table 4.3 shows results on different robustness datasets. We observe that ResNet-101 trained with multiscale samplers consistently achieves a higher accuracy across all benchmark datasets. Furthermore, models trained with multiscale samplers have (2) lower ECE (Figure 4.3a), (3) low image embedding variance (Figure 4.3b), and (4) are less sensitive to scale changes (Figure 4.3c) as compared to single scale samplers. These results suggest that models trained with multiscale samplers are able to more effectively extract the salient semantic information in the image (e.g., changes in image scale and rotation). Moreover, these models are more robust and have a greater discriminative power. Our findings regarding embedding variance match [54, 90], which also show that a reduction in embedding variance accelerates training. (5) Additionally, the large variation in the entropy skewness of the



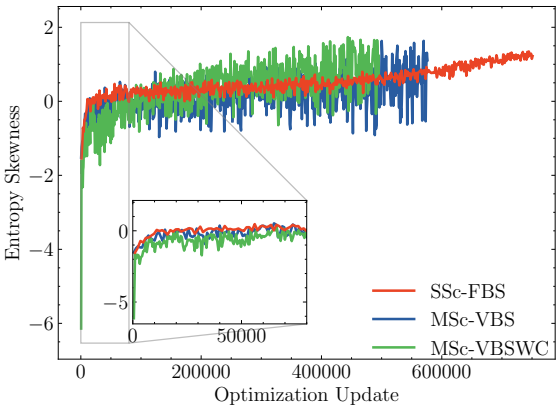
(a) Calibration error



(b) Embedding variance



(c) Accuracy across resolutions



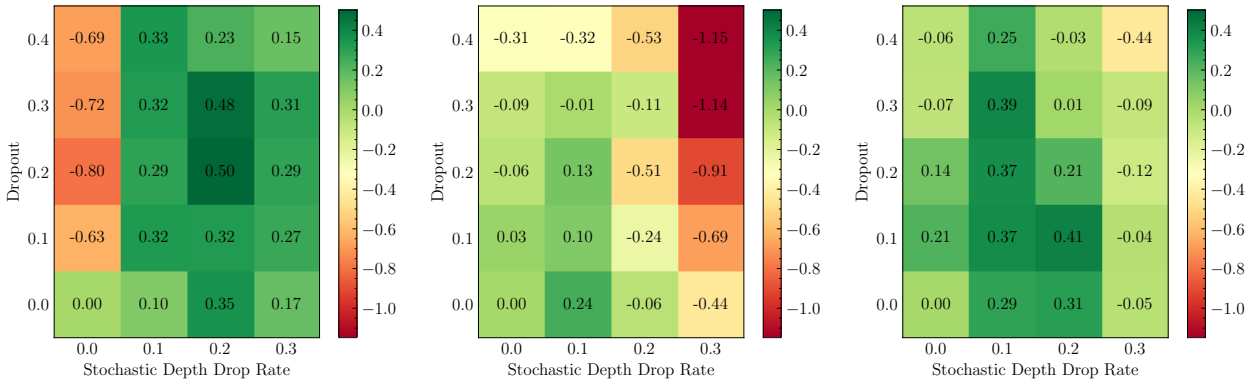
(d) Skewness of empirical entropy distributions

Figure 4.3: **Multiscale samplers improve model calibration, reduce embedding variance, and improve robustness.** Here we train a ResNet-101 model with single-scale and multiscale samplers. We find that models trained with multiscale samplers are better calibrated, learn embeddings with lower variance, and are more robust to changes in input scale. Moreover, multiscale samplers exhibit a larger shift in entropy, suggesting a larger exploration of the weight space.

multiscale samplers (Figure 4.3d) suggests that multiscale samplers explore more of the weight space. Coupled with the gradual shift toward large skewness values, this suggests that the increased exploration steers the model towards more robust minima.

Dataset	SSc-FBS	MSc-VBS	MSc-VBSWC
ImageNet-A	$16.05_{(0.0)}$	$17.91_{(+1.86)}$	$\mathbf{19.17}_{(+3.12)}$
ImageNet-R	$39.77_{(0.0)}$	$\mathbf{41.61}_{(+1.84)}$	$41.57_{(+1.80)}$
ImageNetV2-MF	$68.75_{(0.0)}$	$69.86_{(+1.11)}$	$\mathbf{70.05}_{(+1.30)}$
ImageNetV2-Th	$76.82_{(0.0)}$	$\mathbf{77.97}_{(+1.15)}$	$77.66_{(+0.84)}$
ImageNetV2-TI	$81.30_{(0.0)}$	$81.75_{(+0.45)}$	$\mathbf{82.03}_{(+0.73)}$

Table 4.3: **Training with multiscale samplers improves robustness.** Here we evaluate our ResNet-101 models that were trained with single-scale and multiscale samplers on ImageNet. We report the top-1 accuracy (%) across multiple datasets and observe that models trained with multiscale samplers consistently outperform the single-scale model. ImageNetV2-MF refers to the "matched frequency" subset of ImageNetV2, ImageNetV2-Th refers to the "threshold 0.7" subset, and ImageNetV2-TI refers to the "top images" subset (see [87]).



(a) SSc-FBS (b) MSc-VBS (c) MSc-VBSWC

Figure 4.4: **Multiscale samplers are implicit data regularizers.** We train ResNet-101 on the ImageNet dataset at different values of classifier dropout (y-axis) and stochastic depth drop rate (x-axis) with three samplers (SSc-FBS, MSc-VBS, and MSc-VBSWC). ResNet-101 trained with multiscale samplers (MSc-VBS and MSc-VBSWC) requires less regularization as compared to SSc-FBS. Here, the values in each cell are relative to the bottom left cell. The top-1 accuracy of ResNet-101 for bottom left cell (i.e., the values of classifier dropout and stochastic depth are 0.0) for SSc-FBS, MSc-VBS, and MSc-VBSWC are 81.31%, 81.66%, and 81.53% respectively.

4.2.3 Regularization

Multiscale data augmentation and sampling methods have been adopted in previous works for improving model accuracy [72]. However, to the best of our knowledge, the pairing of multiscale data sampling with explicit regularization methods (e.g., dropout [97], stochastic

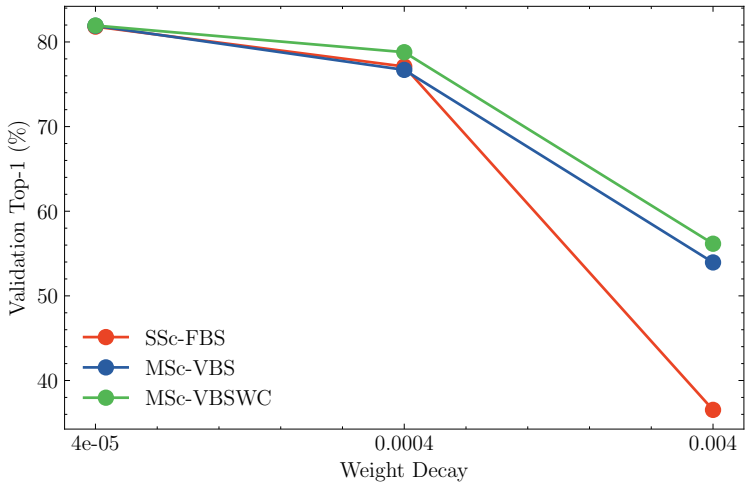


Figure 4.5: **Models trained with multiscale samplers are less sensitive to weight decay strength.** Here we train ResNet-101 models on ImageNet with varying degrees of weight decay. Models trained with multiscale samplers enjoy a lower drop in accuracy due to stronger weight decay.

depth drop rate [49], and L2 regularization) has not been extensively studied. It may be the case that multiscale samplers offer implicit regularization, and as a consequence, require little or no explicit regularization and hence can lead to reduced parameter tuning. In this section, we study the regularization aspect of different samplers.

Metrics: To assess the regularization, we consider three widely-used regularization methods in state-of-the-art models: (1) classifier dropout, (2) stochastic depth drop rate, and (3) L2 weight regularization.

Results: Figure 4.4 shows the effect of varying the values of classifier and stochastic dropouts during training of ResNet-101 on the ImageNet dataset. In the case of SSc-FBS training, we observe that the best top-1 accuracy of 81.81% is obtained when strong regularization (classifier dropout: 0.2 and stochastic depth drop rate: 0.2) is used (Figure 4.4a). On the other hand, similar or better accuracy with smaller regularization coefficients is achieved when we instead train with multiscale samplers. For example, MSc-VBS (Figure 4.4b) achieves

Model	Sampler	GPU Memory	Training FLOPs	Optimization updates	Training time	Top-1 Accuracy (%)
ResNet-50 [38]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	79.43
	MSc-FBS	2.16×	1.15×	1.00×	1.05×	80.01
	MSc-VBS	1.31×	0.77×	0.77×	0.95×	80.03
	MSc-VBSWC	1.03×	**0.70**×	**0.66**×	**0.85**×	**80.25**
SE-ResNet-50 [48]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	80.24
	MSc-FBS	2.21×	1.15×	1.00×	1.13×	80.64
	MSc-VBS	1.32×	0.77×	0.77×	0.99×	**80.71**
	MSc-VBSWC	1.04×	**0.70**×	**0.66**×	**0.88**×	80.56
RegNetY-16GF [83]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	79.64
	MSc-FBS	2.29×	1.15×	1.00×	1.11×	80.52
	MSc-VBS	1.11×	0.77×	0.77×	0.79×	**80.96**
	MSc-VBSWC	1.45×	**0.69**×	**0.66**×	**0.72**×	80.64
EfficientNet-B3 [104]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	81.20
	MSc-FBS	OOM	-	-	-	-
	MSc-VBS	1.01×	0.66×	0.75×	0.78×	81.47
	MSc-VBSWC	1.03×	**0.58**×	**0.65**×	**0.64**×	**81.86**

Table 4.4: **Training models with multiscale samplers reduces compute and training time while improving accuracy.** We train multiple CNN architectures using single-scale and multiscale samplers on the ImageNet dataset. Multiscale samplers are able to consistently match the performance of the single-scale models while reducing training time, optimization updates, and FLOPs.

a top-1 accuracy of 81.91% with a stochastic dropout of 0.1 (and no classifier dropout). Moreover, when there is no explicit regularization (i.e., the value of classifier and stochastic dropouts are 0.0), the multiscale samplers increase the accuracy of ResNet-101 by 0.2-0.3%.

Furthermore, as shown in Figure 4.5, when the value of the L2 regularization coefficient is varied from $4e^{-5}$ to $4e^{-3}$, the drop in top-1 accuracy of models trained with multiscale samplers is significantly lower than the top-1 accuracy drop of the single-scale sampler. These results, in conjunction with Figure 4.4, suggest that ResNet-101 trained with multiscale samplers requires less explicit regularization and is less sensitive to the choice of weight decay. This is likely because multiscale samplers act as data dependent regularizers.

4.3 Multiscale Training Beyond ResNet and Classification

In Sec. 4.2, we analyzed the efficacy of multiscale samplers using ResNet-101 on the ImageNet dataset. In this section, we validate that the benefits of multiscale training extend to other architectures and tasks.

4.3.1 CNNs, Transformers, and Lightweight Image Classification Models

CNNs. We study three different models on the ImageNet dataset using different data samplers: (1) ResNet-50, (2) Se-ResNet-50 [48], (3) RegNetY-16GF [83], and (4) EfficientNet-B3 [104]. Results are given in Table 4.4. We observe that multiscale samplers consistently improve the performance of different models and accelerate training.

Transformers Our analysis of multiscale samplers has centered around CNNs. In this section, we show that multiscale samplers improves training efficiency of vision transformers while being competitive to SSc-FBS. We train ViT-B [21] and Swin-S [66] models using single and multiscale samplers. Results are summarized in Table 4.5. We observe that training ViT-B with MSc-VBS reduces compute by 24% while improving accuracy. MSc-VBS can also be used to train Swin-S more efficiently with comparable accuracy. We observe a larger drop in accuracy when training ViT-B and Swin-S with MSc-VBSWC. We hypothesize that the smaller input resolutions used by MSc-VBSWC at the start of training adversely affect the patch embeddings. We used a standard patch size of 16×16 and a minimum resolution of 128×128 with an initial compression factor of $\rho_0 = 0.75$. Hence, the smallest input resolution at the start of training was 96×96 for MSc-VBSWC. While MSc-VBSWC may still be used to train a vision transformer, larger min/max spatial resolutions may be required and is the subject of future work.

Lightweight Models Our analysis thus far has only considered relatively "heavyweight" networks. In Table 4.6, we train MobileNetv1 [44], MobileNetv2 [92], and MobileNetv3-Large [43]—all of which are lightweight networks designed for computational efficiency—using single-scale and multiscale samplers. We observe that training with multiscale samplers can improve efficiency with similar or better accuracy compared to SSc-FBS.

Model	Sampler	GPU Memory	Training FLOPs	Optimization updates	Training time	Top-1 Accuracy
ViT-B [21]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	79.76
	MSc-VBS	1.75×	0.76×	0.76×	0.80×	**80.12**
	MSc-VBSWC	2.19×	**0.67**×	**0.67**×	**0.74**×	78.44
Swin-S [66]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	**82.93**
	MSc-VBS	1.58×	0.91×	0.76×	0.89×	82.35
	MSc-VBSWC	1.56×	**0.82**×	**0.67**×	**0.81**×	81.78

Table 4.5: **Training vision transformer models with multiscale samplers reduces compute and training time without a significant drop in accuracy.** We train ViT-B [21] and Swin-S [66] using single-scale and multiscale samplers on the ImageNet dataset. Training with multiscale samplers reduces training time, optimization updates, and FLOPs. We observe a drop in accuracy when training with MSc-VBSWC which we hypothesize is due to the small training resolutions at the beginning of training which may affect the patch embeddings.

Model	Sampler	GPU Memory	Training FLOPs	Optimization updates	Training time	Top-1 Accuracy
MobileNetv1 [44]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	73.92
	MSc-VBS	1.01×	0.77×	0.77×	0.98×	**74.16**
	MSc-VBSWC	1.24×	**0.70**×	**0.66**×	**0.97**×	74.05
MobileNetv2 [92]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	**73.36**
	MSc-VBS	1.21×	0.77×	0.77×	0.98×	73.06
	MSc-VBSWC	1.13×	**0.70**×	**0.66**×	**0.91**×	73.09
MobileNetv3 [43]	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	74.74
	MSc-VBS	1.19×	0.77×	0.77×	0.94×	**75.24**
	MSc-VBSWC	1.42×	**0.71**×	**0.66**×	**0.88**×	74.77

Table 4.6: **Multiscale samplers reduce training FLOPs of lightweight CNNs without a significant drop in performance.** We train MobileNet models on ImageNet using the recipes in [70]. We observe that training lightweight networks with multiscale samplers produces models with accuracies that are competitive to the SSc-FBS model while being more efficient to train.

4.3.2 Object Detection with Mask R-CNN

In this section, we study the effect of different samplers on a standard detection model, Mask R-CNN [37] with a ResNet-101 image backbone. We train and evaluate the model on the MS-COCO dataset [64]. To understand the effect of different samplers, we train the Mask R-CNN model with and without pre-training across different sampler configurations. Our results are summarized in Table 4.7. We first note that batch sizes for object detection tasks are typically much smaller than for classification tasks (e.g., a common batch size per GPU for object detection is 4, whereas 256 is commonly used for classification). Therefore, at

Pre-training Sampler	Mask R-CNN Sampler	Peak GPU Memory	Training FLOPs	Optimization updates	Training time	bbox mAP	segm mAP
SSc-FBS	SSc-FBS	**1.00**×	1.00×	1.00×	1.00×	42.91	38.10
	MSc-VBS	3.44×	0.70×	0.63×	0.72×	46.23	41.27
	MSc-VBSWC	3.35×	**0.63**×	**0.54**×	**0.66**×	**46.48**	**41.28**
MSc-VBS	SSc-FBS	**1.09**×	1.00×	1.00×	1.00×	43.19	38.30
	MSc-VBS	3.29×	0.70×	0.63×	0.72×	**46.82**	**41.67**
	MSc-VBSWC	2.68×	**0.63**×	**0.54**×	**0.66**×	46.02	41.12
MSc-VBSWC	SSc-FBS	**1.05**×	1.00×	1.00×	1.00×	42.71	38.12
	MSc-VBS	3.50×	0.7×	0.63×	0.72×	**45.46**	40.60
	MSc-VBSWC	3.30×	**0.63**×	**0.54**×	**0.66**×	45.42	**40.74**
None	SSc-FBS	**0.91**×	1.00×	1.00×	1.00×	37.91	34.21
	MSc-VBS	3.46×	0.7×	0.63×	0.74×	**40.83**	**36.74**
	MSc-VBSWC	3.38×	**0.63**×	**0.54**×	**0.68**×	39.00	35.74

Table 4.7: **Training with multiscale samplers increases mAP while decreasing training time and compute.** We report bounding box and instance segmentation mAP@IoU of 0.50:0.05:0.95 for a Mask R-CNN [37] model with a ResNet-101 backbone. Compared to single-scale training, multiscale samplers achieve better performance while reducing optimization updates by 46% and training FLOPs by 37%. Models in the lower part of the table were trained without pre-training the backbone model. We also note that effective batch sizes in object detection tasks are much smaller than classification tasks (e.g., 4 vs. 256 per GPU). Consequently, effective batch sizes at high resolutions are similar to the batch sizes at the base/reference resolution, causing multiscale variable batch size samplers to behave like multiscale fixed batch size samplers. Hence, we observe a large peak GPU memory.

high resolutions, the effective batch size for multiscale samplers is similar to the batch size of the base/reference resolution, causing multiscale variable batch samplers to behave like the multiscale fixed batch size sampler. Therefore, we observe a larger peak GPU memory when training object detection/instance segmentation tasks with variable batch samplers as compared to classification tasks (Table 4.4).

Our experiments show that the Mask R-CNN model with a multiscale sampler significantly improves performance both when training from scratch and when pre-training. For example, a pre-trained MSc-VBS model attains a 0.47 mAP in detection when training the Mask R-CNN model with MSc-VBS, compared to 0.43 when training the Mask R-CNN model with SSc-FBS. We also observe a commensurate improvement in instance segmentation mAP, where training with MSc-VBS yields an mAP of 0.42, while training with SSc-FBS yields a 0.38 mAP. In addition to substantially improving the mAP, training with multiscale samplers significantly reduces training time and compute. We also note that training with multiscale samplers improves performance for both detection and instance segmentation tasks regardless of the

pre-training method. Our results are consistent with large-scale jittering data augmentation [27] for object detection models. However, a key difference between models trained with large-scale jittering and multiscale samplers is that large-scale jittering uses a fixed batch size during training. As a result, training is slower. On the other hand, multiscale samplers serve the dual purpose of improving accuracy as well as training efficiency.

4.4 Efficient Training via Progressive Compound Scaling[2]

In Sec. 4.2.1, we showed that progressive multiscale training can improve training efficiency of classification models by more than 30% while preserving accuracy. To further improve efficiency, we turn our attention toward progressively expanding the model's capacity by incrementally increasing its width and depth. At the start of training, we compress a model by scaling down its width and depth along with the input resolution. As training progresses, the model's width, depth, and input resolution are gradually expanded following a curriculum. This progressive learning allows us to improve the training efficiency significantly while delivering the same performance as the dense model when trained with a similar compute budget.

4.4.1 Progressive Compound Scaling

We begin by developing the notation used when discussing our model and data expansion during training. Afterwards, we present a grid search analysis on ResNet-50 which allows us to establish a Pareto-optimal relationship between our three scaling dimensions.

[2]This section is based on unpublished work completed during an internship at Apple.

Model Scaling

Let $\mathcal{N} : \mathbb{R}^{C \times H \times W} \to \mathbb{R}^m$ denote an image classification network that operates on images of size (C, H, W) where C denotes channels and H, W are the spatial dimensions. Many CNNs [38, 44, 92, 43] are designed as a sequence of s modules, $\mathcal{F}_i$, followed by a classification head, $\mathcal{H}$. Hence, we can write $\mathcal{N}(X) = (\mathcal{H} \circ \mathcal{F}_s \circ \cdots \circ \mathcal{F}_1)(X)$. Each module $\mathcal{F}_i$ is typically constructed with a sub-module, f_j^i, e.g., a ResNet Bottleneck block, repeated L_i times. In other words, $\mathcal{F}_i(X_i) = (f_{L_i}^i \circ f_{L_i-1}^i \circ \cdots \circ f_1^i)(X_i)$. Each f_j^i can further be analyzed in terms of its constituent convolution layers. Let K_j^i denote the number of convolution layers in f_j^i and $\{C_l^{ij}\}_{l=1,\ldots,K_j^i}$ the corresponding output channels of each convolution layer.

Given network $\mathcal{N}$, we compress it by modulating its width and depth independently. Let $w, d \in (0, 1]$ denote width and depth scaling factors, respectively. Compressing the width of $\mathcal{N}$ is equivalent to scaling $\{C_l^{ij}\}_{l=1,\ldots,K_j^i}$ by w for all i, j. In particular, we introduce a width compression function, g_w, used to uniformly scale each layer's channels:

$$g_w : (\{C_l^{ij}\}_{l=1,\ldots,K_j^i}; w) \mapsto \{\lceil w \cdot C_l^{ij} \rceil_8\}_{l=1,\ldots,K_j^i}, \tag{4.1}$$

where $\lceil \cdot \rceil_8$ rounds to ensure divisibility by 8. We compress the network's depth by scaling each L_i (the number of times module $\mathcal{F}_i$ is repeated) by d. We similarly introduce a depth compression function, g_d, to uniformly scale the number of sub-modules in each module:

$$g_d : (\{L_i\}_{i=1,\ldots,s}; d) \mapsto \{\max(1, \lfloor d \cdot L_i \rceil\}_{i=1,\ldots,s}, \tag{4.2}$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer.

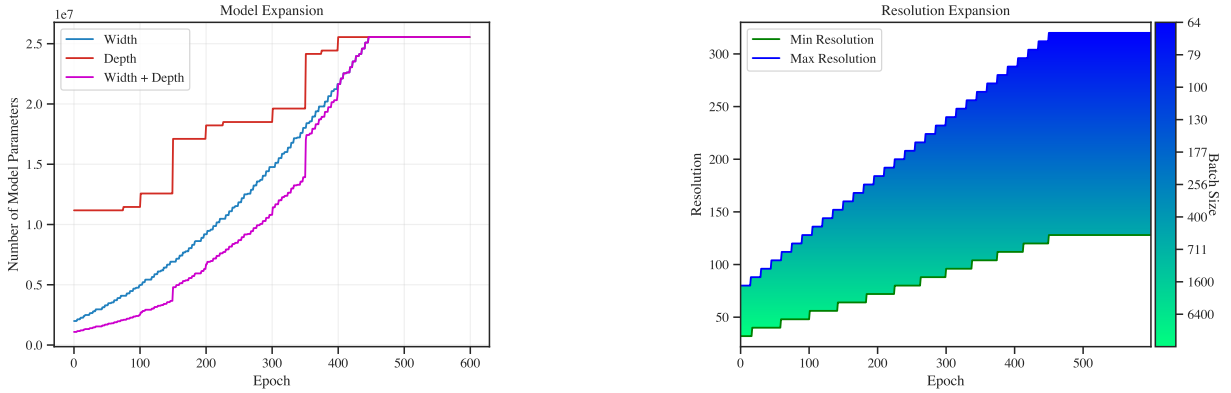
Resolution Scaling

In addition to compressing the model along its width and depth dimensions, we also progressively grow the resolution of the samples used to train the model. Our scaling follows our MSc-VBSWC sampler as defined in Sec. 4.1. In particular, the resolution scaling factor, ρ, progressively grows throughout training.

Expansion Schedules

Our training method begins by compressing the network using initial factors w_0 and d_0, yielding the network $\mathcal{N}_0 = g_d(g_w(\mathcal{N}, w_0), d_0)$. At each training epoch, e, we train a compressed network $\mathcal{N}_e = g_d(g_w(\mathcal{N}, w_e), d_e)$, where $w_e \geq w_{e-1}$ and $d_e \geq d_{e-1}$. Trained parameters from $\mathcal{N}_{e-1}$ are copied over to $\mathcal{N}_e$. Motivated by [118], the new parameters of $\mathcal{N}_e$ are initialized randomly. Hence, our training method begins training with compressed network $\mathcal{N}_0$ and progressively grows it to its full size $\mathcal{N}$ over a specified expansion period. A model trained for a total of E epochs has an expansion period of $E_\tau = \tau \cdot E$, where $\tau \in (0, 1]$ is a hyperparameter that controls the fraction of training epochs dedicated to growing the model. After the model reaches its full size, we continue training for the remaining $(1 - \tau)E$ epochs using the full model. The rate at which the model grows follows a linear schedule. Namely, the width and depth multipliers at epoch e are computed using Eq. (4.3), where c_0 represents an initial width, depth, or resolution multiplier. We illustrate the effect of our width and depth scaling on ResNet-50 in Fig. 4.6a, and resolution scaling in Fig. 4.6b.

$$h_l(e; c_0, E_\tau) = \begin{cases} c_0 + \frac{1-c_0}{E_\tau} e, & e < E_\tau \\ 1, & e \geq E_\tau. \end{cases} \tag{4.3}$$



(a) (b)

Figure 4.6: **ResNet-50 progressive compound model and resolution scaling schedule.** We consider a ResNet-50 model trained for 600 epochs with an expansion period of $\tau = 0.75$. (a) Effect of linearly scaling only the width, depth, or both of ResNet-50 following a linear expansion curriculum. (b): The progressive resolution scaling following our MSc-VBSWC data sampler.

4.4.2 ResNet-50 Progressive Compound Scaling

To shed some light on the effect that w_0, d_0, r_0, and τ have on the performance of a model versus its training compute, we perform a grid search on ResNet-50 over these four parameters with the following search space: $w_0 \in \{0.25, 0.5, 0.75, 1\}, d_0 \in \{0.25, 0.5, 0.75, 1\}, r_0 \in \{0.25, 0.5, 0.75, 1\}$, and $\tau \in \{0.5, 0.75, 0.95\}$. Each of these models is trained for 150 epochs and with our compound scaling method discussed in Sec. 4.4.1. Additionally, we train eight baseline dense models with the SSc-FBS data sampler (i.e., we do not apply model compression and use a fixed sample resolution) for $150, 120, 105, \ldots, 30$ epochs, corresponding to baselines with $100\%, 80\%, 70\%, \ldots, 20\%$ of the total compute budget. For SSc-FBS baselines, we train with a batch size of 128 and a resolution of 224×224. For our MSc-VBSWC data sampler, we use a reference batch size of 256 and resolution 160×160. The minimum (pre-scaled) resolution is 128×128 and the maximum (pre-scaled) resolution is 320×320.

In Fig. 4.7a, we plot the results of our grid search as a function of the fraction of total training FLOPS used compared to the 100% SSc-FBS model. We make the following

observations:

- Keeping the model size fixed and varying only the resolution with $\tau = 0.95$ and $r_0 \in \{1, 0.75, 0.5, 0.25\}$ (green points) yields the optimal accuracy-efficiency tradeoff with up to a 60% reduction in total training FLOPS[3].
- Keeping the resolution scaling fixed ($r_0 = 1$) and scaling only the model width and depth dimensions (blue and red points, respectively) yields a suboptimal tradeoff in terms of training efficiency vs. accuracy.
- To reduce training FLOPS beyond what is enabled by resolution scaling alone, we must perform a compound scaling where we vary width, depth, and resolution (gray points).

In particular, we observe that the optimal accuracy-efficiency trade-off consists of three phases as depicted in Fig. 4.7b. This observation allows us to develop a Pareto-optimal ordering in which compound scaling should be performed during training in order to improve efficiency:

- **Resolution Phase (R)**: Keep the model width and depth fixed at $w_0 = 1, d_0 = 1$. Then, gradually reduce the initial resolution factor r_0 until the desired training budget is reached or r_0 reaches its minimum value (0.25 for our experiments).
- **Width Phase (W)**: If the resolution compression limits have been reached, then keep r_0 fixed at its minimum value, keep $d_0 = 1$ fixed, and gradually reduce the width w_0 until the desired training budget is reached or w_0 reaches its minimum value (0.25 for our experiments).
- **Depth Phase (D)**: If the resolution and width phases have not yielded the desired training budget, keep w_0 and r_0 fixed at their minimum values, then gradually decrease

[3]The fraction of compute reduction will vary depending on the resolutions and batch sizes used for both SSc-FBS and MSc-VBSWC.

d_0 until the desired training budget is reached or d_0 reaches its minimum value (0.25 for our experiments).

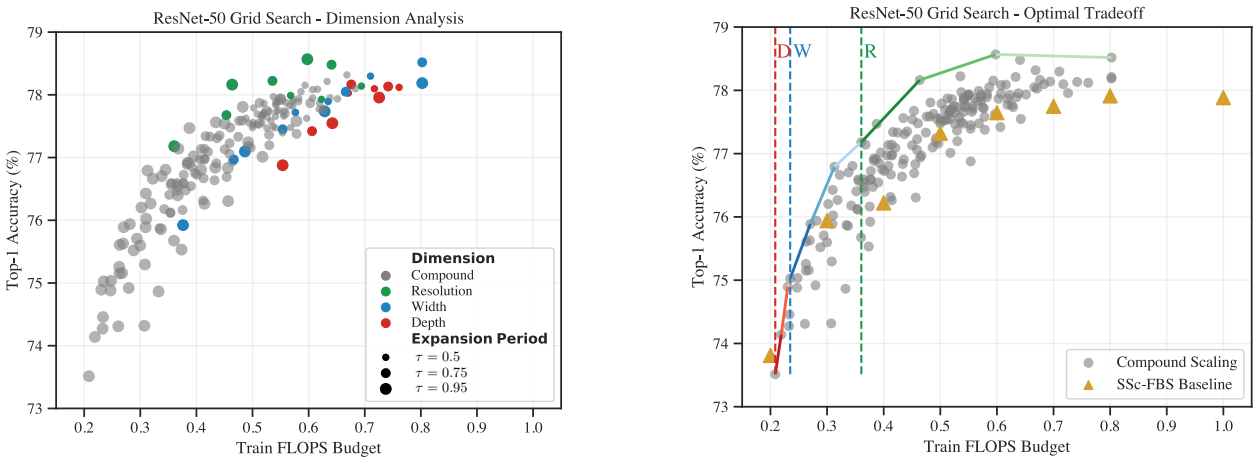


Figure 4.7: **ResNet-50 progressive compound model and resolution scaling grid search.** We progressively expand the width and depth of ResNet-50 while also expanding the sample resolutions. The initial compression factors for width, depth, and resolution are $w_0 \in \{0.25, 0.5, 0.75, 1\}$, $d_0 \in \{0.25, 0.5, 0.75, 1\}$, and $r_0 \in \{0.25, 0.5, 0.75, 1\}$, respectively. The expansion period search space is $\tau \in \{0.5, 0.75, 0.95\}$. (a) Expanding only the model depth and width yields a sub-optimal accuracy-efficiency trade-off, while resolution scaling alone provides a strong trade-off. To improve efficiency beyond that of resolution scaling, compound scaling works best. (b): The Pareto-optimal ordering for expansion is: resolution scaling, followed by joint width and resolution scaling, followed by joint depth, width, and resolution scaling.

Thus, progressive resolution expansion as performed by our MSc-VBSWC data sampler gives a stronger accuracy-efficiency trade-off than model width and depth expansion. As such, for the purpose of training efficiency, this method should be performed first. If one wishes to further reduce training compute by modulating the capacity of the model, the next best expansion method is joint model width and resolution scaling. Finally, jointly scaling the model's depth, width, and resolution should be used if training budget constraints have not been met.

4.5 Discussion

In this chapter, we presented an empirical study of multiscale samplers for training deep neural networks in visual recognition tasks. Compared to single-scale training, we showed that multiscale training maintains or improves accuracy, while reducing training time and FLOPs, enhancing robustness, and producing better-calibrated models. We analyzed three different sampling strategies, including a novel curriculum-based multiscale sampler that enforces a gradual increase in the resolution of images during the course of training, while adapting the batch size for smarter utilization of compute and memory at different stages of learning. Based on our experiments and analysis, we have demonstrated that using multiscale samplers yields significant advantages compared to single-scale samplers in various aspects of training and model properties. Furthermore, we extended our study to other network architectures and visual recognition tasks, namely object detection and instance segmentation, demonstrating similar benefits of multiscale training in these settings. We have shown that multiscale samplers, across a wide range of tasks, can be more robust and efficient alternatives to single-scale samplers.

To further reduce training compute beyond what is achieved by multiscale samplers alone, we explored compound scaling, where we progressively increase both the model's width and depth in parallel with the sample resolutions. In this setting, we find that there is a Pareto-optimal ordering in which resolution scaling alone provides the best accuracy-efficiency trade-off. To further improve efficiency, joint model width and resolution expansion should be performed. Finally, parallel expansion of all three, model depth, width, and resolution scaling should be performed.

Our conclusions in this chapter provide a solid foundation for further exploration and development of multiscale samplers in the context of deep neural network training. Extensions to this work should investigate large-scale foundation models (e.g., CLIP [82]) in both visual and language domains. Related to this, training with multi-modal multi-task data, wherein

different modalities and batch sizes are consumed, is a timely and exciting future direction.

4.6 Appendix

4.6.1 Training Details

In this section we provide additional training details of our experiments. For a particular model, all sampling schedules (SSc-FBS, MSc-FBS, MSc-VBS, MSc-VBSWC) are trained using the same recipe. Hence, multiscale samplers are drop-in replacements for single-scale samplers, requiring no modifications to the existing SSc-FBS recipe. All models are trained with the corresponding recipes implemented in the CVNets library [70]. RegNet [83] models are trained with the ResNet recipe. Though CVNets recipes consider the exponential moving average (EMA) of training checkpoints, all of our experimental results are reported for the non-EMA model. All of our CNN architectures are trained on a single node with four NVIDIA A100 GPUs. Vision Transformer models are trained on a single node with eight NVIDIA A100 GPUs.

When training with a multiscale sampler (MSc-FBS, MSc-VBS, MSc-VBSWC), we must first specify a "reference" batch shape as discussed in Sec. 4.1. Additionally, we define a minimum spatial resolution, (H_1, W_1), and a maximum, (H_n, W_n). To construct the spatial resolutions in $\mathcal{S}$, we interpolate between the minimum and maximum spatial resolutions, imposing divisibility by either 8 or 32. The batch sizes in $\mathcal{S}$ are computed based on the batch size of the reference batch shape. We report the reference batch shapes as well as the min/max spatial resolution used for each of our models and samplers in Table 4.8.

4.6.2 Multi-GPU vs. Single-GPU Training With Multiscale Samplers

The data samplers we considered in this chapter typically assume multi-GPU training, however, they may still be utilized using a single GPU. Table 4.9 compares the top-1 (%)

Model	MSc-VBS Ref.	MSc-VBSWC Ref.	Min (H,W)	Max (H,W)
ResNet [38]	(256, 3, 224, 224)	(512, 3, 160, 160)	128	320
RegNet [83]	(256, 3, 224, 224)	(512, 3, 160, 160)	128	320
EfficientNet [83]	(256, 3, 300, 300)	(256, 3, 300, 300)	160	448
ViT [21]	(256, 3, 224, 224)	(256, 3, 224, 224)	128	320
Swin [66]	(256, 3, 224, 224)	(256, 3, 224, 224)	128	320
Mask R-CNN [37]	(4, 3, 1024, 1024)	(4, 3, 1024, 1024)	512	1280

Table 4.8: **Reference batch shapes and min/max spatial resolutions for MSc-VBS and MSc-VBSWC**. The reference batch shapes for each sampler are used to determine the batch size of the spatial resolutions sampled at each training iteration. Reference batch shapes maintain a similar compute within each model and are reported as (B, C, H, W). Spatial resolutions at each iteration are sampled from resolutions interpolated between the min/max spatial resolutions.

accuracy of a ResNet-101 model trained using single and multiscale samplers on platforms with either 1 or 4 GPUs. For each sampler and GPU configuration, we report the mean and standard deviation for three seeds (due to computational constraints, we are unable to provide such measurements for all experiments). We observe that, for a given number of GPUs, the multiscale sampler performance is slightly lower than the single-scale sampler performance when trained on a single GPU. However, the use of multiple GPUs for training enables multiscale samplers to exceed the performance of the single-scale sampler. We hypothesize that the aggregation of gradients across different batch sizes and resolutions facilitates learning.

# GPUs	Sampler	Top-1 Accuracy (%)
4	SSc-FBS	81.03 ± 0.32
	MSc-VBS	81.79 ± 0.09
	MSc-VBSWC	81.56 ± 0.10
1	SSc-FBS	80.15 ± 0.11
	MSc-VBS	79.01 ± 0.23
	MSc-VBSWC	79.68 ± 0.50

Table 4.9: **Multiscale samplers may benefit from multi-GPU training.** We train ResNet-101 on ImageNet either on a single GPU or 4. We report the accuracy aggregated over three seeds. 4-GPU models were trained for 600 epochs while 1-GPU models were trained for 150 epochs. We observe that training with multiple GPUs enables multiscale samplers such as MSc-VBS and MSc-VBSWC to outperform single scale samplers; however, when training on a single GPU, single scale samplers (SSc-FBS) outperform multiscale samplers.

CHAPTER 5

Cross-Attention and Curriculum-Based Masking for Efficient Masked Autoencoders[1]

In this chapter, we extend our focus beyond the image modality to include audio and video, while also broadening our training approach to the self-supervised setting. Leveraging the abundant availability of unlabeled videos, many unsupervised training frameworks have demonstrated impressive results in various downstream audio and video tasks. Recently, Masked Audio-Video Learners (MAViL) [50] has emerged as a state-of-the-art audio-video pre-training framework. MAViL couples contrastive learning with masked autoencoding to jointly reconstruct audio spectrograms and video frames by fusing information from both modalities.

Pre-training of masked autoencoder models [36, 106, 113] on video is costly, often requiring hundreds of GPUs. In an effort to mitigate this, in this chapter we study the potential synergy between diffusion models and MAViL, seeking to derive mutual benefits from both of these two frameworks. By incorporating diffusion into MAViL, we are able to leverage efficient Attention mechanisms to reduce training costs. Moreover, motivated by our work on multi-scale samplers (Chapter 4), here we train our masked autoencoder with a curriculum imposed on the masking ratio. This curriculum guides the model to learn global features early in training, followed by fine-grained features in later stages. By also adapting the batch size according to the masking ratio, we enhance training efficiency while not compromising the model's performance on downstream audio-classification tasks when compared to MAViL.

[1]Work completed during an internship at Apple.

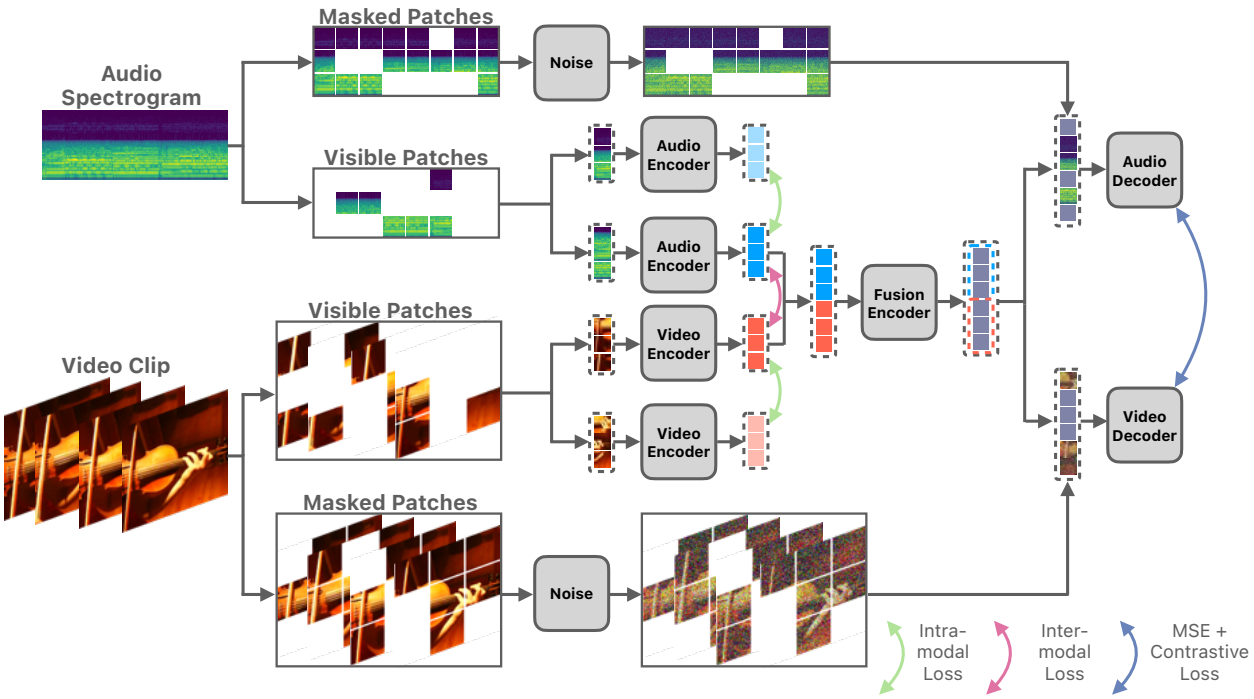


Figure 5.1: **DiffMAViL architecture.** Similar to the audio-video encoder-decoder architecture of MAViL [50], our DiffMAViL architecture takes as input RGB video frames and audio spectrograms. The spectrogram and RGB frames are first randomly masked, and visible patches from each modality are encoded via their respective encoders. Masked patches are diffused and concatenated with the outputs of the audio-video fusion encoder, which are then fed through the audio and video decoders to obtain reconstructions of the input spectrogram and RGB frames.

5.1 Masked Autoencoders for Audio and Video

We begin by introducing the MAViL [50] framework, from which our model, DiffMAViL, is built upon.

5.1.1 Masked Audio Video Learners (MAViL)

Let (a, v) be an audio-video instance pair where a is an audio spectrogram and v is a tensor of RGB video frames. a and v are first patchified and tokenized, producing $\mathbf{a} = [a_1, \ldots, a_M]$ audio tokens and $\mathbf{v} = [v_1, \ldots, v_N]$ video tokens where $a_i, v_j \in \mathbb{R}^d$. In the encoding step, a fraction, $\rho \in (0, 1)$, of the audio and video tokens are then randomly masked,

yielding $\mathbf{a}'$ and $\mathbf{v}'$ containing $\lfloor (1-\rho)M \rceil$ and $\lfloor (1-\rho)N \rceil$ visible tokens, respectively, where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. These visible tokens are then embedded by audio and video ViT-based encoders, f_a, and f_v, producing the uni-modal audio and video representations $\mathbf{a}_{um} \triangleq f_a(\mathbf{a}')$ and $\mathbf{v}_{um} \triangleq f_v(\mathbf{v}')$. The uni-modal representations are then concatenated, forming $(\mathbf{a}_{um}, \mathbf{v}_{um})$, and passed through a ViT-based fusion encoder, g_{av}, producing multi-modal representations $(\mathbf{a}_{mm}, \mathbf{v}_{mm}) \triangleq g_{av}(\mathbf{a}_{um}, \mathbf{v}_{um})$. In the decoding step, the $\mathbf{a}_{mm}$ and $\mathbf{v}_{mm}$ are first projected onto the decoder space. Then, a learnable `[MASK]`token is appended to each of the multi-modal representations for each of the masked patches in the encoding step, yielding $\tilde{\mathbf{a}}_{mm}$ and $\tilde{\mathbf{v}}_{mm}$. These are then passed through ViT-based decoders for each modality, denoted f_a^{-1} and f_v^{-1}, followed by a linear projection head l_a and l_v. Therefore, the reconstructions of (patchified) a and v are given by $\hat{\mathbf{a}} \triangleq l_a(f_a^{-1}(\tilde{\mathbf{a}}_{mm}))$ and $\hat{\mathbf{v}} \triangleq l_v(f_v^{-1}(\tilde{\mathbf{v}}_{mm}))$. Letting $\mathbf{a}_i^{\text{raw}}$, $i = 1, \ldots, M$, and $\mathbf{v}_j^{\text{raw}}$, $j = 1, \ldots, N$ denote the patches of the original audio and video inputs, the mean-squared error (MSE) loss is given by $\mathcal{L}^{MSE} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}_{\{\mathbf{a}_i \text{ masked}\}} (\hat{\mathbf{a}}_i - \mathbf{a}_i)^2 + \frac{1}{N} \sum_{j=1}^{N} \mathbb{1}_{\{\mathbf{v}_j \text{ masked}\}} (\hat{\mathbf{v}}_j - \mathbf{v}_j)^2$ where the indicator functions ensure that the loss is only computed for masked patches.

In addition to minimizing the MSE loss, the first stage of MAViL also considers two contrastive losses. The first, the "inter-modal" loss, facilitates alignment across modalities by first averaging the audio and video uni-modal representations, $\mathbf{a}_{emb} \triangleq \text{Avg}(\mathbf{a}_{um})$, $\mathbf{v}_{emb} \triangleq \text{Avg}(\mathbf{v}_{um})$, where $\text{Avg}(\cdot)$ denotes averaging along the sequence length. These instance-level representations are then fed through the InfoNCE loss, where video and audio clips from the same video constitute positive pairs while all other pairs are negatives. The second loss, the "intra-modal" loss, promotes alignment within each modality. By applying a second random masking to the input audio and video clips, a second "view" of each modality can be obtained, $\bar{\mathbf{a}}_{emb}$ and $\bar{\mathbf{v}}_{emb}$, which are then also fed through the InfoNCE loss. In this case, the two views from the same instance are considered a positive pair and the negative pairs consist of the views from all other instances of the same modality. MAViL's first stage objective function is therefore a linear combination of the MSE loss and the two contrastive losses;

hence, this procedure consists of four forward passes through the encoders (one pass for each view through its respective modality's encoder).

5.1.2 Diffusion-Based Masked Audio Video Learners (DiffMAViL)

To encourage our model to learn representations that capture high frequency features, and motivated by DiffMAE [116], we augment the MAViL audio and video branches with diffusion [41]. Our approach is outlined in Fig. 5.1. In MAViL's audio and video decoders, learnable `[MASK]`tokens are used to represent masked spectrogram/RGB frame patches. In our DiffMAViL model, we replace the learnable `[MASK]`tokens with diffused patches. Let x_0^m represent a masked audio or video frame patch where m denotes a masked patch and the subscript denotes the diffusion time step. At each training iteration, we sample $t \sim \text{Unif}(\{1, 2, \ldots, T\})$, and diffuse x_0^m according to noise level t to obtain $x_t^m = \sqrt{1-\bar{\alpha}_t}\epsilon + \sqrt{\bar{\alpha}_t}x_0^m$ where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ is a standard normal sample with the same dimension as x_0^m. The multimodal embeddings output by the fusion encoder, along with x_t^m, are then projected onto the decoder's embedding space, and after restoring the original patch ordering, are fed through the corresponding decoder. The decoders are therefore tasked with reconstructing the original input from the visible patch emebeddings and diffused masked patches in a single step.

As in [116], when training with diffusion we use the "simple" objective function proposed in [41]. Namely, the objective is to minimize the reconstruction error between the masked input x_0^m, and the decoder's reconstruction given x_t^m and the visible latents. In other words, this reduces to the reconstruction MSE used by MAViL. The objective function optimized by DiffMAViL is therefore the same as MAViL.

5.2 Enhanced Training Efficiency

5.2.1 Leveraging Cross-Attention

We begin by replacing the Self-Attention modules in our video branch's decoder with Cross-Attention modules [111]. In Cross-Attention, masked patch embeddings only attend to visible patch embeddings, as illustrated in Fig. 5.2. Due to transformers' quadratic complexity in the sequence length, Cross-Attention is more efficient than Self-Attention which operates on the concatenated sequence of masked and visible patch embeddings. In particular, the complexity of standard Self-Attention is $O(L^2)$ where L is the sequence length, and the complexity for Cross-Attention is $O(\rho(1-\rho)L^2)$ where ρ is the masking ratio. We note that the use of Cross-Attention is facilitated by our use of diffusion; without diffused patches, Cross-Attention would apply attention between masked tokens—which contain no information about the masked patches—and visible tokens. Our decoder is similar to the "cross" decoder presented in [116], however, our Cross-Attention modules attend only to the visible latents of the final encoder block, rather than to all of them. For the audio decoder, we use the Swin-Transformer local attention [66] as this was shown to perform favorably in [51].

5.2.2 Curriculum-Based Masking Ratio

Masking Ratio Curriculum. Curriculum learning [10] aims to organize training samples in a way that facilitates learning. This notion has inspired several progressive learning methods [105, 77] (Chapter 4) that progressively increase the resolution of training samples throughout training. Inspired by this, we propose a dynamic masking ratio that progressively decays over the course of training. In MAViL, a fixed masking ratio, $\rho \in (0, 1)$, is used throughout training. As the transformer blocks for both the audio and video encoders in DiffMAViL operate only on visible patches, we can improve efficiency by processing fewer visible patches. The number of visible patches is a fraction, $1-\rho$, of the total number of patches. Hence, by using a larger value of ρ, we mask out a greater number of patches and consequently process

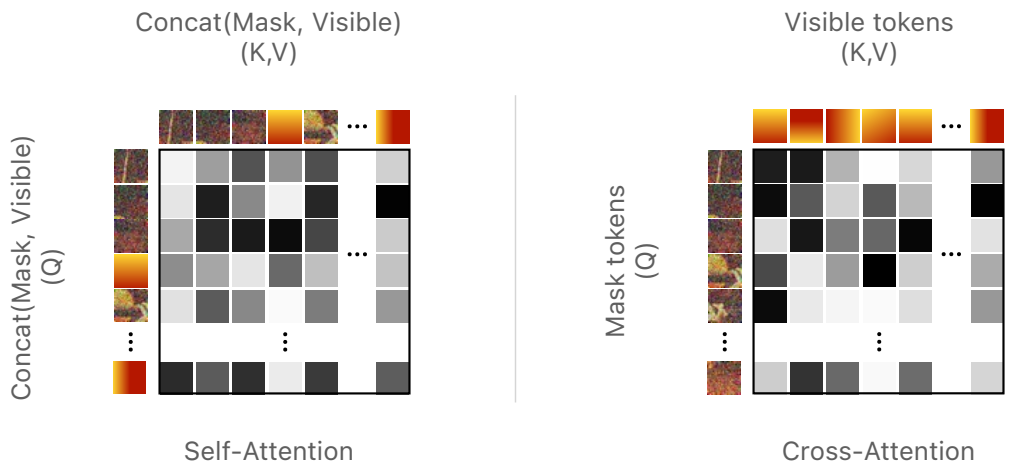


Figure 5.2: **DiffMAViL's video decoder leverages Cross-Attention for efficiency.** For a sequence with L tokens and masking ratio $\rho \in (0, 1)$, in standard Self-Attention (left), ρL masked patches + $(1-\rho)$L visible patches attend to ρL masked patches + $(1-\rho)$L visible patches, for a complexity of $O((\rho L + (1-\rho)L)^2) = O(L^2)$. In contrast, in Cross-Attention (right), ρL masked tokens attend to $(1-\rho)L$ visible tokens, for a complexity of $O(\rho(1-\rho)L^2)$.

fewer visible patches. We therefore propose having a dynamic masking ratio that begins at $\rho_1 \in (0, 1)$ and ends at $\rho_2 \in (0, 1)$ following a schedule. We consider a simple linear masking ratio schedule that varies from ρ_1 at the start of training to ρ_2 at the end of training.

In our experiments, we set $\rho_1 > \rho_2$, implying that training begins with a high masking ratio and gradually decays to a smaller masking ratio (ρ_1 in our experiments is the fixed masking ratio used in MAViL, so we mask patches more aggressively early in training). This curriculum encourages the model to initially learn global representations of its inputs by exposing it to only a few visible patches at the start of training, thereby limiting the context available to focus on local details. As training progresses and the number of visible patches increases, the model is able to focus on reconstructing local details, while building on the global representations learned earlier.

Adaptive Batch Size. In vision tasks, training with a lower sample resolution naturally entails the utilization of fewer computational resources, which may lead to underutilization of accelerators. As outlined in Chapter 4, this can be addressed by using an adaptive batch size, where larger batch sizes are used when training at a lower resolution and smaller batch sizes

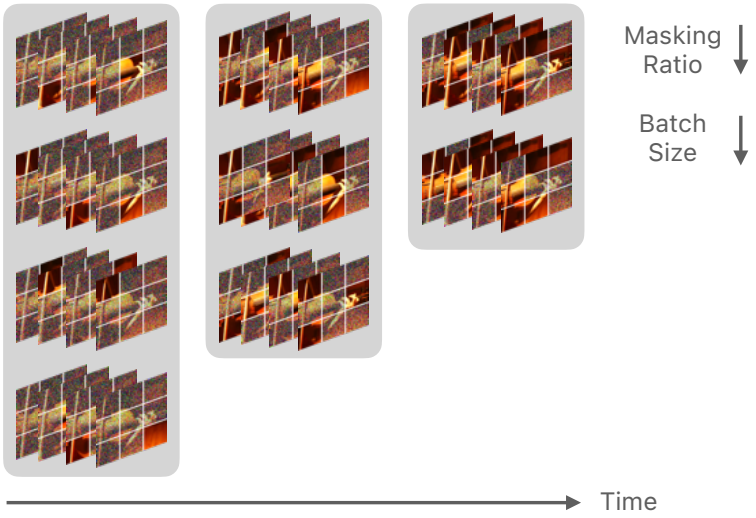


Figure 5.3: **DiffMAViL progressively decays the masking ratio while dynamically adjusting the batch size.** DiffMAViL begins training with a masking ratio ρ_1 which gradually decays to ρ_2 throughout training. As the masking ratio decays, DiffMAViL processes more visible patches. To maximize GPU utilization, DiffMAViL uses a large batch size when the masking ratio is large (fewer visible patches), and a small batch size when the masking ratio is small (more visible patches).

at higher resolution, resulting in faster training. We extend this method by making the batch size adaptive to the masking ratio, as illustrated in Fig. 5.3. For a base batch size B_0 (i.e., the batch size that will be used for the masking ratio $\min(\rho_1, \rho_2)$), the batch size at epoch e is given by $B_e = \frac{1-\min(\rho_1,\rho_2)}{1-\rho_e} B_0$, where ρ_e is the masking ratio at epoch e as determined by the masking ratio schedule.

5.3 Experiments

To pre-train our models, we use the union of the "balanced" and "unbalanced" splits of the AudioSet [26] dataset, denoted "AS-2M." We note that we were only able to acquire 85% of the total AudioSet dataset, as many videos are no longer available on YouTube. We pre-train all baselines on this dataset for fair comparison. We focus on fine-tuning on only the audio modality, i.e., we fine-tune only the audio encoder branch of our DiffMAViL and MAViL models. We fine-tune on the "balanced" AudioSet split (denoted "AS-20K") and report the

mean average precision (mAP). Additionally, we fine-tune on VGGSound [13], Environmental Sound Classification (ESC-50) [81], and Speech Commands v2 (SPC-v2) [115] where we use the split considered in [76]. We report the Top-1 (%) accuracy for VGGSound, ESC-50, and SPC-v2. For ESC-50, we report the mean accuracy under standard five-fold cross validation. For each experiment, we report the mean and standard deviation of three independent seeds. Additional training details are provided in Sec. 5.5.1.

In Tab. 5.1, we compare the performance of MAViL against our DiffMAViL model. We observe that the use of diffusion, coupled with our efficiency strategies outlined in Sec. 5.2, reduces pre-training FLOPS and wall-clock time without incurring a significant loss in performance.

Model	**AS-20K** (mAP ↑)	**VGGSound** (Top-1 ↑)	**ESC-50** (Top-1 ↑)	**SPC-v2** (Top-1 ↑)	**FLOPS**	**Avg. Epoch Time**
MAViL	35.9 ± 0.10	57.2 ± 0.12	93.7 ± 0.13	98.0 ± 0.08	$1\times$	$1\times$
DiffMAViL (ours)	35.8 ± 0.16	57.0 ± 0.17	93.1 ± 0.18	97.7 ± 0.04	$\mathbf{0.68}\times$	$\mathbf{0.82}\times$

Table 5.1: **DiffMAViL improves training efficiency while maintaining accuracy.** Our DiffMAViL model integrates diffusion into the MAViL [50] framework along with a Cross-Attention video decoder, linear masking ratio schedule, and a dynamic batch size to improve efficiency. *We present results for our own MAViL implementation as the public release is not available at the time of writing.

5.3.1 Benefits of Diffusion, Cross-Attention, Curriculum-Based Masking, and Adaptive Batch Sizes

In this section, we ablate over several design choices of DiffMAViL.

AudioMAE + Diffusion. Here, we investigate the benefits of using diffusion in the audio-only modality. In particular, we integrate diffusion into the AudioMAE [51] framework; this is equivalent to discarding our video modality branch and focusing only on audio. As described in Sec. 5.2, we simply replace the learnable `[MASK]`tokens in the decoder with

diffused spectrogram patches. In Tab. 5.2, we compare the downstream performance of our implementation of AudioMAE with AudioMAE + Diffusion. We observe that training with diffusion improves performance in downstream tasks, suggesting that diffusion may aid in the learning of richer audio representations.

Diffusion	AS-20K (mAP ↑)	VGGSound (Top-1 ↑)	ESC-50 (Top-1 ↑)	SPC-v2 (Top-1 ↑)
✗	34.2 ± 0.06	57.1 ± 0.16	92.6 ± 0.12	98.4 ± 0.03
✓	35.5 ± 0.07	57.9 ± 0.08	93.6 ± 0.02	98.4 ± 0.05

Table 5.2: **Diffusion improves the performance of AudioMAE.** We augment the AudioMAE [51] framework with diffusion and observe that diffusion facilitates the learning of richer audio representations in the absence of the video modality. Both models (with and without diffusion) were pre-trained on the AS-2M [26] dataset.

Replacing Self-Attention with Cross-Attention. To reduce pre-training compute, we replace the Self-Attention modules in the video branch's decoder with Cross-Attention [111]. In row R2 of Tab. 5.3, we observe that the use of Cross-Attention reduces pre-training FLOPS by 19% while preserving accuracy across multiple datasets.

Row #	Video attention	Masking ratio	Adaptive batch size	AS-20K (mAP ↑)	VGGSound (Top-1 ↑)	ESC-50 (Top-1 ↑)	SPC-v2 (Top-1 ↑)	FLOPS	Avg. epoch time
R1	Self	Fixed	✗	36.0 ± 0.08	57.5 ± 0.09	94.7 ± 0.10	97.9 ± 0.04	1×	1×
R2	Cross	Fixed	✗	36.3 ± 0.09	57.4 ± 0.03	94.2 ± 0.20	97.9 ± 0.08	0.81×	0.96×
R3	Cross	Linear	✗	36.0 ± 0.07	57.3 ± 0.19	93.3 ± 0.10	97.6 ± 0.05	0.68×	0.96×
R4	Cross	Linear	✓	35.8 ± 0.16	57.0 ± 0.17	93.1 ± 0.18	97.7 ± 0.04	0.68×	0.82×

Table 5.3: **DiffMAViL ablations.** Compared to our baseline DiffMAViL model (R1), replacing the video decoder's Self-Attention modules with Cross-Attention reduces pre-training FLOPS by 19% (R2). Replacing the fixed masking ratio of 0.8 with a linear schedule that decays from 0.9 to 0.8 reduces FLOPS by 32% (R3). Adding an adaptive batch size reduces pre-training wall-clock time by 18% (R4).

As mentioned in Sec. 5.2, the use of diffusion allows us to leverage Cross-Attention due to the information contained in the masked patches. To validate this, in Tab. 5.4 we compare how MAViL and DiffMAViL respond to using Cross-Attention versus Self-Attention. We

Model	Video Attention	AS-20K (mAP ↑)	ESC-50 (Top-1 ↑)	SPC-v2 (Top-1 ↑)
MAViL	cross	36.1 ± 0.04	93.6 ± 0.25	98.0 ± 0.12
DiffMAViL (ours)	cross	36.3 ± 0.09	94.2 ± 0.20	97.9 ± 0.08

Table 5.4: **DiffMAViL Is More Amenable to Cross-Attention.** Replacing Self-Attention in DiffMAViL's video decoder with Cross-Attention has a more positive effect on downstream performance compared to MAViL with Cross-Attention. Efficiency metrics are measured relative to the standard MAViL model in Tab. 5.1.

observe that DiffMAViL with Cross-Attention tends to have a stronger performance on downstream audio classification tasks compared to MAViL with Cross-Attention.

Curriculum-Based Masking. We further improve training efficiency by augmenting DiffMAViL with a curriculum for the masking ratio. In row R3 of Tab. 5.3, we show that a linear schedule that decays the masking ratio from 0.9 to 0.8 throughout training reduces pre-training FLOPS by 32%. In Sec. 5.3.2, we analyze the FLOPS reduction within each encoder and decoder module.

Adaptive Batch Size. While pre-training with a dynamic masking ratio reduces pre-training FLOPS, it does not have a significant decrease in the wall-clock training time as it leads to under-utilization of computational resources. To offset this, we augment DiffMAViL with an adaptive batch size in service of maintaining constant compute at each iteration (Sec. 5.2 for details). The dynamic balance between masking ratio and batch size allows us to utilize hardware more efficiently and maintain similar FLOPs to R3 in Tab. 5.3. Consequently, this reduces the number of optimization steps required per epoch, resulting in an 18% reduction in wall-clock pre-training time while maintaining accuracy.

5.3.2 FLOPS Analysis

In Tab. 5.5, we summarize the reduction in FLOPS on a per-module basis for each of our efficiency strategies. Efficiency metrics are measured relative to the standard MAViL model in Tab. 5.1. Row R1 is our DiffMAViL model with no efficiency strategies. Here we observe that the video encoder FLOPS are lower than MAViL's. This is because, in MAViL, the first step after patchifying the input is to project all the patches onto the encoder space and then mask them; however, in DiffMAViL, we first mask patches and subsequently project only the visible patches onto the encoder space. This is so that we can later diffuse the masked patches before projecting them onto the decoder space. Moreover, we observe that the decoder FLOPS in R1 are slightly higher than those of MAViL; we attribute this to the fact that, in DiffMAViL, the diffused masked patches must first be projected onto the decoder embedding space prior to being processed by the decoder. In contrast, the standard video decoder without diffusion only projects visible patch embeddings output by the encoder since the `[MASK]`tokens are already of the appropriate dimension. In row R2 we observe that the use of Cross-Attention reduces the video decoder FLOPS by about 47%. In row R3, we observe that the additional use of a linear masking ratio schedule reduces audio and video encoder FLOPS by about 26-28%. This is because a higher masking ratio yields fewer visible patches, and therefore fewer patches are processed by the encoders.

Row #	Masking Ratio	Video Attention	Audio Encoder	Audio Decoder	Video Encoder	Video Decoder	Fusion Encoder	Total
R1	Fixed	self	1.0	1.0	1.0	1.0	1.0	1.0×
R2	Fixed	cross	1.0	1.0	0.97	0.53	1.0	0.81×
R3	Linear	cross	0.74	1.0	0.72	0.54	0.74	0.68×

Table 5.5: **FLOPS reduction in audio/video encoders and decoders due to use of diffusion, Cross-Attention, and a masking ratio schedule.** The use of Cross-Attention instead of Self-Attention in the video decoder reduces total pre-training FLOPS by 19%. Adding a linear masking ratio curriculum further reduces the pre-training FLOPS by 32%. Efficiency metrics are reported relative to the standard MAViL model in Tab. 5.1.

5.4 Discussion

In this chapter, we introduced a diffusion-based masked autoencoder for audio and video, DiffMAViL. By augmenting our framework with diffusion, we were able to leverage Cross-Attention to enhance training efficiency. Combined with our use of a masking ratio curriculum, we were able to reduce training FLOPS and wall clock time without compromising accuracy.

Although we pre-trained our models on both audio and video data, we have only evaluated our model on downstream audio tasks due to computational constraints. Evaluating our method on downstream video tasks remains as future work. Moreover, while we have focused on the audio and video modalities, our implementation is suitable for other domains, such as images, as well. Extending our method to other domains is an exciting direction for future work.

5.5 Appendix

5.5.1 Training Details

Our audio encoder-decoder architecture follows that of AudioMAE [51], while our video encoder-decoder architecture follows that of SpatiotemporalMAE [23]. Namely, our audio and video encoders are both ViT-B models [21]. Both decoders have 8 transformer blocks, 16 attention heads, and an embedding dimension of 512. The audio decoder uses local attention Swin-Transformer [66] blocks. Both encoders and decoders use sinusoidal positional embeddings, and the video encoder and decoder use separable temporal and spatial positional embeddings. The fusion encoder consists of a two-layer Transformer. As a masking ratio of 0.8 was shown to perform well in [50], we also use a masking ratio of 0.8 as our default. Notably, we pre-train both our DiffMAViL and the standard MAViL models with the same hyperparameters. Moreover, as the the code for MAViL is not publicly available at the time of writing, our results for this model are from our own implementation.

We pre-train with both audio and video modalities, and fine-tune only on audio tasks. To construct audio spectrograms, we use the entirety of the data sample. For AudioSet and VGGSound, this corresponds to 10 second audio clips. ESC-50 and SPC-v2 correspond to 5 and 1 second clips, respectively. We use a 16K sampling rate and 128 Mel-frequency bands with a 25ms Hanning window shifting every 10ms. This yields spectrograms with shapes 1024×128, 1024×128, 512×128, and 128×128 for AudioSet, VGGSound, ESC-50, and SPC-v2, respectively. For video, we sample 4-second clips consisting of 16 frames. We use a spatial patch size of 16×16 for both audio and video, and a temporal patch size of 2.

In Tab. 5.6, we provide the hyperparameters used to train DiffMAViL and MAViL (note that we use the same hyperparameters for both models). For diffusion, we use a linear variance schedule, β_t, with $t \in \{1, 2, \ldots, 1000\}$. β_t increases linearly from 10^{-4} to 0.02. As was done in [116], we exponentiate the variances with hyperparameter $\phi = 0.8$ so that the noise variance is β_t^{ϕ}. This amplifies the noise used at lower diffusion steps t.

We note that we did not use a weighted sampling for neither pre-training nor fine-tuning on any dataset. All of our training was done on NVIDIA A100 GPUs.

Configuration	Pre-training	Fine-tuning			
	AS-2M	AS-20k	VGGSound	ESC-50	SPC-v2
Optimizer	AdamW [69]				
Optimizer momentum	$\beta_1 = 0.9, \beta_2 = 0.95$				
Weight Decay	1e-5	1e-4	1e-4	1e-4	1e-4
Learning rate	4e-4	2.5e-4	2e-4	2.5e-4	1e-3
Learning rate schedule	Cosine decay [68]				
Layer-wise learning rate decay [8]	None	0.75	0.75	0.75	0.75
Minimum learning rate	1e-6	1e-6	1e-6	1e-6	1e-6
Warm-up epochs	8	4	1	4	4
Epochs	60	60	60	100	60
Batch size*	2048	64	256	64	256
GPUs	256	1	4	1	1
Augmentation†	R	R	R+N	R	R
SpecAug [78] (time/freq)	None	192/48	192/48	96/24	48/48
Stochastic dropout [49]	0	0.1	0.1	0.1	0.1
Mixup [130]	None	0.5	0.5	0	0
Cutmix [127]	None	1.0	1.0	0	0
Multilabel	-	True	False	False	False
Loss function‡	MSE+Contrastive	BCE	BCE	CE	CE
Dataset mean	-4.268	-4.268	-5.189	-6.627	-6.702
Dataset std	4.569	4.569	3.260	5.359	5.448

Table 5.6: **Pre-training and fine-tuning hyperparameters.** We use the same hyperparameters for both diffusion and non-diffusion models. *: Batch size refers to effective batch size. †:"R" refers to sampling random starting points with cyclic rolling in time when loading waveforms. "N" refers to adding random noise to the spectrogram. ‡: "BCE" is binary cross entropy, and "CE" is cross entropy.

CHAPTER 6

Efficient Adaptation of Hybrid State Space Models to Long Sequences of Tokens[1]

In previous chapters, we enhanced training efficiency of models by modulating the model's capacity and data complexity. Moreover, we concentrated our efforts on vision tasks. In this chapter, we shift our attention to Large Language Models, particularly Hybrid State Space Models. We focus on enhancing pre-training efficiency; more specifically, we are interested in enabling the fine-tuning of pre-trained models on long context sequences. Similar to our approach in Chapter 5, we achieve this efficiency by replacing the Attention mechanism of the model with a more efficient variant. Within these layers, we show that modulating the size of the local attention span during training improves performance on downstream long context tasks.

While Transformer-based fine-tuning strategies for processing long sequences can also be applied to Hybrid models, they typically work under the assumption that information in the Attention layers is aggregated by recency and are thus more expensive as sequences get longer. There exist efficient LoRA-based methods [47] to bring this cost down, however, we show that Transformer-based methods tend to not work as well for Hybrid models as they do for Transformers. We show that this is mostly due to the fact that combining LoRA (on Attention layers) with SSM layers is not expressive enough to model long-range dependencies beyond the pre-training context size.

[1]Work completed during an internship at AWS.

To overcome such limitations, we propose a parameter-efficient fine-tuning method based on LoRA that is better suited to Hybrid models. Namely, we employ a fine-tuning strategy similar to LoRA+ [14] on Attention layers, while allowing the SSMs' recurrent parameters to be adapted as well. In particular, building on previous observations [128, 124], we also adapt the 1D convolutional layers, which we empirically validate provide the best results at a reduced computational cost.

6.1 Hybrid SSMs: Complementing Fading Memory with Eidetic Memory

Transformers [111] have dominated the fields of Natural Language Processing and Computer Vision. The core component driving Transformer models is the Attention mechanism, which scales quadratically in its input sequence length, precluding researchers from training Transformer models on long context tasks without access to vast computational resources. In an effort to mitigate the quadratic scaling of many works have proposed sub-quadratic approximations to Attention [58, 9, 114, 16, 75], though there remains a gap between the performance of these Attention mechanisms and standard full attention.

While a great effort has been made to improve the efficiency of Transformer models, a recent line of work has explored efficient alternative 'linear' architectures. In particular, State Space Models (SSMs) [31, 33, 32, 123, 100] have emerged as promising competitors to Transformer models due to their efficient scaling and strong empirical performance. SSMs are inspired by classical state space models [55]: they process the input sequence by maintaining a *fixed-size* state which acts as a compressed (lossy) representation of all the processed tokens. However, when implemented in hardware, the state must have finite precision, and therefore the information contained in the state about earlier tokens"fades" as more samples are processed. In contrast, Transformer models have a state determined by the number of tokens in their input sequence and are able to access information from all past tokens in their

context "eidetically." However, they do so at the cost of extra compute and memory.

Hybrid State Space Models have been recently introduced in an effort to complement the SSMs' fading state with Attention layers [18, 19, 63, 28, 128]. However, Attention layers only aggregate information by recency (i.e., they process the keys and values of the most recent tokens up to hardware limitations). Hence, any token that lies beyond the Attention's limited span can only be approximately recalled through the SSMs' state. This limits the effectiveness of Hybrid SSMs when applied to long sequences of tokens, especially when compute and memory resources are limited.

In this chapter, we modify Hybrid SSMs' Attention layers to allow their state to be allocated by *relevancy* rather than *recency*, allowing our models to retrieve information that would otherwise fall outside their Attention span. To do so, we propose Span-Expanded Attention (SE-Attn), a novel selection mechanism that complements the "fading" memory of SSMs with "eidetically"-retrieved tokens.

6.2 Span-Expanded Attention

Our goal is to enable pre-trained Hybrid SSMs to accurately process sequences with a larger number of tokens than were used for pre-training. For a sequence of L tokens with model dimension d_{model}, the computational complexity of standard Self-Attention is $O(d_{\text{model}}L^2)$. For very large L, computing Self-Attention can be prohibitively expensive. Training models with a large L is particularly challenging, as training has the additional cost of computing gradients, further limiting memory resources. To address this, we propose Span-Expanded Attention (SE-Attn), a drop-in replacement for standard Self-Attention in Hybrid SSMs. To train SE-Attn, we propose HyLoRA, a variant of LoRA+ [14] specifically tailored to Hybrid models. A schematic of our SE-Attn is provided in Fig. 6.1.

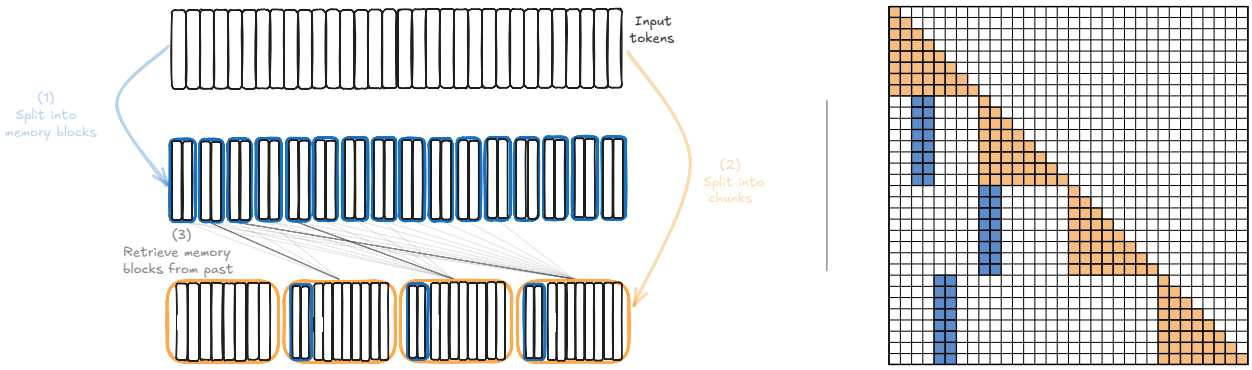


Figure 6.1: **Span-Expanded Attention (SE-Attn) overview.** SE-Attn is a sparse Attention mechanism used to expand the memory span of Hybrid SSMs. We do so by reserving a fraction of the Attention context for tokens retrieved arbitrarily far back in the past, and use summary tokens to efficiently look back with a reduced compute cost. We call this reserve the 'expansion span,' and we populate it with blocks of previous tokens (memory blocks). When new tokens arrive, a similarity-based search compares the queries with past memory blocks to decide which memory blocks are most relevant. Then, these retrieved memory blocks are jointly processed with the queries via Attention. While the final Attention mechanism always processes a fixed number of tokens, it can have a longer span since tokens from arbitrarily far back in the past can be retrieved.

6.2.1 Attention

At the heart of modern LLMs is the Attention mechanism, whose role is to construct a contextualized representation of their input. The input to Attention, $x \in \mathbb{R}^{L \times d}$, is a tensor of L tokens, each with dimension d. Attention is parameterized by $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_{\text{model}}}$ and $W_o \in \mathbb{R}^{d_{\text{model}} \times d}$, which are used to construct linear projections of the input: $Q = xW_Q$, $K = xW_K$, $V = xW_V \in \mathbb{R}^{L \times d_{\text{model}}}$. After adding positional information to the keys and queries, the output of the Attention layer is[2] $\mathit{Attention}(Q, K, V) = \left[\text{softmax}\left(\frac{QK^T}{\sqrt{d_{\text{model}}}}\right) V\right] W_o \in \mathbb{R}^{L \times d}$. The realization of the attention score matrix, $QK^T \in \mathbb{R}^{L \times L}$, grows quadratically in the sequence length, L. Moreover, the output of Attention is typically followed by a feed-forward layer (FFN), which expands and contracts the dimension of the input. While these FFNs are generally regarded as being the primary computational bottlenecks of Attention-based

[2]For simplicity, we consider Single-Head Attention in this exposition.

models, when the sequence length exceeds the expanded dimension, Attention becomes the primary bottleneck. Since LLMs consist of many layers of Attention, this computation is particularly expensive during training, where activations are cached. To make the Attention mechanism computationally amenable to longer contexts during training, we draw inspiration from RAG and Sparse Attention methods.

6.2.2 Amnesic Attention

The crux of our method is in the chunking of the L tokens in the input sequence x, into chunks of size M. Namely, we begin by computing projections $Q = xW_Q$, $K = xW_K$, $V = xW_V$ and adding positional information as in standard Attention. Next, each of the $Q, K, V \in \mathbb{R}^{L \times d_{\text{model}}}$ projections are split into $T = \frac{L}{M}$ chunks of M tokens in each along the sequence dimension, yielding $Q_i, K_i, V_i \in \mathbb{R}^{M \times d_{\text{model}}}$ for $i = 1, \ldots, T$. A naive way to reduce the quadratic cost of Attention is by applying it independently on each of these chunks, $A_i = \mathit{Attention}(Q_i, K_i, V_i)$ and then concatenate and project them to get the final output, Concatenate$(A_1, A_2, \ldots, A_T)$. However, since the chunks are processed independently, there is no information exchanged between them. Hence, this naive—though efficient—Attention mechanism, which we refer to as "SE-Attn-NoMem", cannot model time dependencies on contexts larger than the chunk size.

6.2.3 Eidetic Retrieval Attention

In this section, we improve upon SE-Attn-NoMem by allowing different chunks to exchange information while minimizing compute; we call this Attention mechanism SE-Attn. To this end, we augment the processing of each chunk with a mechanism to retrieve tokens from previous chunks. In particular, we allow chunk i to retrieve tokens from chunks $1, 2, \ldots, i-1$ and, when the most relevant chunks are selected, append their tokens to its context (its "expansion span"). SE-Attn takes as input a sequence, $x \in \mathbb{R}^{L \times d}$ and computes projections

$Q = xW_Q$, $K = xW_K$, $V = xW_V$ followed by the addition of RoPE [99] embeddings as in standard Attention. As described in Sec. 6.2.2, Q, K, V are split into T chunks with M tokens in each, yielding tuples (Q_i, K_i, V_i), $i = 1, \ldots, T$. Additionally, Q, K, V are split into a second set of U chunks with S tokens in each, yielding tuples $(Q_j^{\text{Mem}}, K_j^{\text{Mem}}, V_j^{\text{Mem}})$ where $Q_j^{\text{Mem}}, K_j^{\text{Mem}}, V_j^{\text{Mem}} \in \mathbb{R}^{S \times d_{\text{model}}}$ for $j = 1, \ldots, U$. We refer to these tuples as "memory blocks"; in SE-Attn, the query from each chunk, Q_i, attends not only to K_i, but also to a set of retrieved memory blocks which populate SE-Attn's expansion span. In particular, each Q_i retrieves k (top-k) key/value memory blocks from the past[3]: $\left(K^{\text{Mem}}_{\phi_i(U)_1}, V^{\text{Mem}}_{\phi_i(U)_1}\right), \ldots, \left(K^{\text{Mem}}_{\phi_i(U)_k}, V^{\text{Mem}}_{\phi_i(U)_k}\right)$ where $\phi_i(U)_j$ denotes the index of the j-th memory block selected by the i-th chunk. These retrieved blocks are then appended to the chunks' keys and values, and SE-Attn computes the Attention output for the ith chunk as in Eq. (6.3).

$$\tilde{K} = \text{Concatenate}(K^{\text{Mem}}_{\phi_i(U)_1}, \ldots, K^{\text{Mem}}_{\phi_i(U)_k}, K_i) \tag{6.1}$$

$$\tilde{V} = \text{Concatenate}(V^{\text{Mem}}_{\phi_i(U)_1}, \ldots, V^{\text{Mem}}_{\phi_i(U)_k}, V_i) \tag{6.2}$$

$$A_i^{\text{SE-Attn}} = \mathit{Attention}(Q_i, \tilde{K}_i, \tilde{V}_i) \tag{6.3}$$

Afterwards, each chunk's Attention outputs are concatenated and projected to obtain the SE-Attn layer output:

$$o^{\text{SE-Attn}} = \text{Concatenate}(A_1^{\text{SE-Attn}}, A_2^{\text{SE-Attn}}, \ldots, A_T^{\text{SE-Attn}}). \tag{6.4}$$

Note that Eq. (6.3) is a form of Cross-Attention since we are not concatenating memory query tokens to the chunk's query. This is to preserve causality, as the retrieved tokens cannot attend to the chunk's tokens. Next, we discuss how the memory tokens are retrieved.

Memory Retrieval. Each chunk x_i must judiciously select which memory blocks to retrieve

[3]By "the past," we mean tokens in the sequence that came before tokens in the chunk.

from the past. To do this efficiently, we associate a 1-dimensional tensor, $c_j \in \mathbb{R}^{d_{\text{model}}}$, to each of the $j = 1, \ldots, U$ memory blocks which act as a compressed representation of each memory block. To determine which memory blocks Q_i should attend to, we compute a "relevancy score" between Q_i and each c_j, which measures how relevant memory block j is to chunk i. This relevancy score is implemented with a Cross-Attention score between chunks and compressed memory block representations. More specifically, recalling that $Q_i \in \mathbb{R}^{M \times d_{\text{model}}}$, we compute relevancy score $R_{ij} \in \mathbb{R}$ between chunk $i \in \{1, 2, \ldots, M\}$ and memory block $j \in \{1, 2, \ldots, U\}$ as follows:

$$R_{ij} = \sum_{t=1}^{M} (Q_i c_j)_t. \tag{6.5}$$

$R_i \in \mathbb{R}^U$ represents the relevancy scores between chunk i and all memory blocks. However, since chunk i should only retrieve memory blocks that came before it temporally, we add a mask to R_i and then apply softmax to obtain the final scores, $\tilde{R}_i$, between chunk i and all memory blocks as follows:

$$\tilde{R}_i = \text{softmax}\left(\frac{1}{\sqrt{d_{\text{model}}}}(R_i + \mathcal{M}_i)\right) \in \mathbb{R}^U \tag{6.6}$$

where $\mathcal{M}_i$ is a mask of 0 and $-\infty$ constructed to set the scores of future memory blocks (relative to chunk i) to $-\infty$. Once all relevancy scores are computed, chunk i simply retrieves the top k memory blocks with the highest $\tilde{R}_{ij}$ scores and concatenates them with the keys and values as in Eqs. (6.1) and (6.2) before computing Attention as in Eq. (6.3).

Compressed Memory Blocks. Next, we discuss how we construct the compressed memory block representations, c_j. Recently, Landmark Attention [74] considered using "landmark" tokens to obtain compressed representations of memory blocks. We consider using landmark

tokens to construct c_j in Sec. 6.6.2. In SE-Attn, we consider a simpler approach which we found to work well. Namely, for each memory block, $(Q_j^{\text{Mem}}, K_j^{\text{Mem}}, V_j^{\text{Mem}})$, we perform standard non-causal Attention, $A_j^{\text{Mem}} = \text{softmax}\left(\frac{Q_j^{\text{Mem}}(K_j^{\text{Mem}})^T}{\sqrt{d_{\text{model}}}}\right) V_j^{\text{Mem}} \in \mathbb{R}^{S \times d_{\text{model}}}$; we consider non-causal Attention since we are interested in a global representation where all tokens within the memory block can attend to each other. Next, we simply compute the mean of these weighted memory tokens in order to compute the compressed representation of the memory block:

$$c_j = \frac{1}{S} \sum_{t=1}^{S} (A_j^{\text{Mem}})_t \in \mathbb{R}^{d_{\text{model}}} \tag{6.7}$$

where $(A_j^{\text{Mem}})_t$ denotes the tth row of A_j^{Mem}.

6.2.4 HyLoRA: Training SE-Attn with LoRA

In this chapter, we fine-tune models pre-trained with standard Attention; however, we fine-tune them with SE-Attn, which modifies the pre-trained Attention mechanism by introducing a retrieval mechanism. SE-Attn repurposes the model's Attention parameters (W_Q, W_K, W_V, W_o) to perform retrieval. In order to efficiently train the model to learn to use SE-Attn, we use a variant of LoRA. Recently, [14] introduced LoRA+, a variant of LoRA [47] designed to fine-tune Transformer models on long contexts. LoRA fine-tunes Attention parameters with low rank adapter matrices. LoRA+ differs from LoRA by also training embedding and normalization layers.

Recently, [24] found that pure SSMs can be fine-tuned by training the SSM projection layers with LoRA. Since we fine-tune on long contexts, we prioritize efficient training, and consequently apply LoRA only to the Attention layers of our hybrid models. However, it is common for SSM layers to include a 1D convolution layer after their initial projection in order to perform sequence mixing [128, 28, 63, 18, 19]. The 1D convolution parameters

constitute a very small portion of the model parameters (∼0.7% for Mamba-2-Hybrid 2.7B), but as discussed further in Sec. 6.3.5, we found that training these 1D convolution layers in conjunction with LoRA+ improved our models' performance on long-context tasks. We refer to this augmented LoRA+ variation as HyLoRA and use it to fine-tune all of our hybrid models.

6.3 Experiments

6.3.1 Experimental Setup

Models. We enhance the recall capabilities of pre-trained hybrid SSM models by fine-tuning them with different Attention layers on spans of tokens longer than the ones used for pre-training. We consider Mamba-2-Hybrid 2.7B[4] [112] as our representative SSM hybrid model. We also explore expanding the span of Transformer models, and consider Llama1 7B [109] as our representative model. Both pre-trained model checkpoints were obtained from HuggingFace and were pre-trained with a context size of 2048. Throughout our experiments, "Non-fine-tuned" refers to the pre-trained model with no fine-tuning.

Datasets. We fine-tune models using a common language dataset and provide additional results when fine-tuning on a mixture of language and code dataset in Sec. 6.3.2.1, and on PG-19 [84] in Sec. 6.3.2.2.

Baselines. For Mamba-2-Hybrid, we compare SE-Attn to three Attention variants: standard full Attention ("Full-Attn") [111], Sliding Window Attention ("SW-Attn") [9], and Shifted Sparse Attention (S^2-Attn) [14]. The Full-Attn baseline serves as the paragon, as the other methods aim to approximate it. For Llama1, due to computational constraints, we only compare against SW-Attn and S^2-Attn. All Attention implementations are based on FlashAttention-2 [17].

[4]We use the Mamba-2-Hybrid weights available on HuggingFace: https://huggingface.co/state-spaces/mamba2attn-2.7b.

Training Procedure. We adopt the training recipe used in [14] to fine-tune our models, with the exception of using HyLoRA for Mamba-2-Hybrid. Additional training details are provided in Sec. 6.6.1.

Evaluation Metrics. We assess the performance of our models across a range of common benchmarks. To assess the predictive capabilities of our fine-tuned models on long sequence lengths, we measure perplexity (PPL) on the PG-19 validation set. To assess performance on more real-world language tasks in the short and long-context settings, we evaluate our models on various LM Evaluation Harness tasks [25]. To measure our models' performance on long-context tasks, we evaluate on the challenging RULER [46] benchmark. We consider eleven RULER tasks, encompassing needle-in-a-haystack, variable-hopping, and aggregation tasks; we aggregate these metrics into five groups as explained in Sec. 6.6.3.4.

Evaluating with Full Attention Similar to [14], we utilize SE-Attn for efficient fine-tuning and revert to using Full-Attn during evaluation. Due to KV-caching, the complexity of Full-Attn scales linearly during evaluation, so an efficient Attention layer in this setting is not as crucial as in training. Nevertheless, for the sake of completeness, we begin by evaluating our models with the same Attention layer used during fine-tuning (i.e., we fine-tune and deploy our Hybrid model with SE-Attn). In Tab. 6.1, we experiment with models that use Full-Attn, SW-Attn, S^2-Attn, and SE-Attn; then, we evaluate perplexity on the PG-19 dataset. All models were fine-tuned with a context size of 8192. We observe that SW-Attn is best at preserving the perplexity on context sizes up to 32× larger than the one used for pre-training. SE-Attn is also able to maintain a lower perplexity across longer context sizes, while S^2-Attn deteriorates much more quickly.

6.3.2 Mamba-2-Hybrid

All of our Mamba-2-Hybrid models are fine-tuned with a context size of 8192. For SE-Attn, we use a block size of 32, and a top-k of 8. For SW-Attn, we use a window size of 4096. S^2-Attn uses the parameters from [14]. When fine-tuning Mamba-2-Hybrid using

Attention	Evaluation Context Size (PPL ↓)			
	8192	16384	32768	65536
Non-fine-tuned	14.99	19.35	26.37	34.51
Full-Attn	10.28	10.39	11.14	12.38
SW-Attn	10.33	10.22	10.16	10.16
S^2-Attn	10.73	10.76	11.85	13.72
SE-Attn	10.47	10.70	10.91	10.96

Table 6.1: **Mamba-2-Hybrid fine-tuned with SE-Attn or SW-Attn preserves perplexity up to** $32\times$ **the pre-training context size.** We fine-tune a Mamba-2-Hyrbid model (pre-trained on a context size of 2048) on a context size of 8192 using various Attention variants. Then, we deploy them on longer context sizes using the same Attention variant used during adaptation.

SE-Attn, we found that applying the same chunk sizes at each layer leads to suboptimal downstream performance, we therefore segment each sample into chunks of variable sizes (picked randomly from $\{2048, 4096\}$). We found this prevents the model from 'overfitting' the given segmentation of the past which is a hyper-parameter of our method fixed a-priori and not dependent on the actual data content. An ablation on SE-Attn with different chunk sizes is provided in Sec. 6.3.4.

Perplexity. We provide PG-19 PPL results for Mamba-2-Hybrid in Tab. 6.2. All fine-tuned models improve upon the non-fine-tuned model. SE-Attn yields the closest performance to the paragon model fine-tuned with Full-Attn, outperforming S^2-Attn and SW-Attn across all context sizes at or above the fine-tuning size.

LM Harness. Next, we evaluate Mamba-2-Hybrid models across short and long-context tasks in the LM Evaluation Harness suite. Our results are provided in Tab. 6.2, where we observe that all models perform similarly on short-context tasks—including the non-fine-tuned model—suggesting there is no performance regression when fine-tuning on larger contexts. Furthermore, we observe that fine-tuning with SE-Attn gives the closest performance to fine-tuning with Full-Attn.

RULER. Next, we assess the performance of our models on long-context tasks using the RULER benchmark. We plot the average accuracy across eleven RULER tasks in Fig. 6.2

Attention	Eval Context Size (PG-19 PPL ↓)				Short Context Tasks (↑)							Long Context Tasks (↑)			
	2048	8192	16384	32768	ARC-E	ARC-C	Hella.	LAMB.	PIQA	WG	Avg.	SWDE	SQA	SNQA	Avg.
Non-fine-tuned	10.72	14.99	19.35	26.37	69.91	37.97	67.62	69.84	76.06	65.04	64.41	85.60	15.18	3.65	34.81
Full-Attn	10.99	10.28	10.39	11.14	69.53	38.48	67.30	68.93	75.08	64.40	63.95	85.24	26.99	19.75	43.99
SW-Attn	10.98	10.80	11.82	13.45	69.82	38.23	67.35	69.18	75.30	63.85	63.95	84.61	24.85	15.41	41.63
S^2-Attn	10.87	12.89	14.67	16.37	70.12	38.05	67.39	69.84	75.95	64.56	64.32	86.41	17.44	8.53	37.46
SE-Attn	10.99	10.45	11.14	12.64	70.20	38.65	67.15	69.11	75.57	63.93	64.10	85.96	26.70	18.04	43.57

Table 6.2: **Fine-tuning Mamba-2-Hybrid with SE-Attn outperforms fine-tuning with S^2-Attn and SW-Attn on long-context natural language tasks.** We fine-tune Mamba-2-Hybrid with a context size of 8192 using various Attention variants. We evaluate PG-19 validation perplexity (PPL) and observe that fine-tuning with SE-Attn preserves performance at longer contexts better than S^2-Attn and SW-Attn. On short-context tasks from the LM Harness suite, all models perform similarly. On long context tasks from the LM Harness suite, SE-Attn outperforms S^2-Attn and SW-Attn. Abbreviations: Hella=HellaSwag, LAMB=LAMBADA, WG=WinoGrande, SQA=ScrollsQAsper, SNQA=ScorllsNarrativeQA. (see Sec. 6.6.3.4 for details on how we aggregate these metrics). We observe that fine-tuning with SE-Attn yields a performance similar to fine-tuning with Full-Attn, and outperforms S^2-Attn and SW-Attn. We also note a substantial improvement on the variable tracking (VT) task, which may be attributed to SE-Attn's retrieval during fine-tuning.

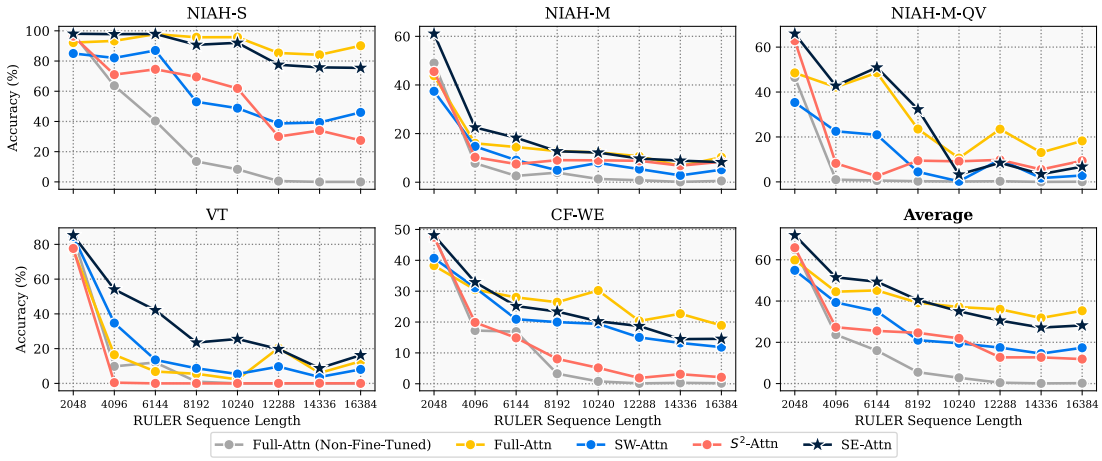


Figure 6.2: **Fine-tuning with SE-Attn outperforms SW-Attn and S^2-Attn on the RULER benchmark when applied to Mamba-2-Hybrid.** We fine-tune Mamba-2-Hybrid with a context size of 8192 using various Attention variants. We average over eleven RULER tasks, as explained in Sec. 6.6.3.4. Fine-tuning with SE-Attn consistently outperforms SW-Attn and S^2-Attn even when evaluating on context sizes beyond the fine-tuning size.

6.3.2.1 Fine-Tuning on Natural Language + Code

Next, we consider fine-tuning on a dataset that augments natural language with a code dataset. In particular, we construct a dataset consisting of 70% natural language, and 30% C++ code extracted from the Lots of Code [129] dataset. We provide PG-19 validation PPL results, as well as results on tasks from the LM Evaluation Harness suite in Tab. 6.3. Similar to fine-tuning on natural language only (as in Tab. 6.2), we observe that fine-tuning with SE-Attn yields the strongest performance in downstream tasks. We provide performance on RULER in Fig. 6.3. We again observe that fine-tuning with SE-Attn yields the strongest perfromance compared to other efficient Attention layers. Moreover, compared to fine-tuning only on natural language, here we observe a greater improvement on the variable tracking task. As explained in Sec. 6.6.3.4, this task requires the model to keep track of the values of variables that are defined and overridden throughout the input context, and must then return the variables equal to some value. This requires strong recall capabilities, which fine-tuning with SE-Attn enables.

Attention	Eval Context Size (PG-19 PPL ↓)				Short Context Tasks (Acc ↑)							Long Context Tasks (Acc ↑)			
	2048	8192	16384	32768	ARC-E	ARC-C	Hella.	LAMB.	PIQA	WG	Avg.	SWDE	SQA	SNQA	Avg.
Non-fine-tuned	10.72	14.99	19.35	26.37	69.91	37.97	67.62	69.84	76.06	65.04	64.41	85.60	15.18	3.65	34.81
Full-Attn	10.94	10.23	10.36	11.09	70.12	38.14	67.40	69.34	75.08	64.56	64.11	84.88	25.99	19.61	43.49
SW-Attn	10.94	10.76	11.81	13.43	69.70	38.82	67.51	69.38	75.41	64.88	64.28	84.52	24.44	15.30	41.42
S^2-Attn	10.86	12.89	14.67	16.36	69.95	37.88	67.45	69.94	76.01	64.72	64.32	86.50	16.65	8.61	37.25
SE-Attn	10.95	10.41	11.07	12.50	70.45	38.91	67.39	69.07	75.08	64.96	64.31	85.24	26.14	18.08	43.15

Table 6.3: **Fine-tuning Mamba-2-Hybrid with SE-Attn on a natural language + code dataset outperforms fine-tuning with S^2-Attn and SW-Attn on natural language tasks.** We fine-tune Mamba-2-Hybrid with a context size of 8192 using various Attention variants on a dataset that consists of 70% natural language and 30% code. We evaluate PG-19 validation perplexity (PPL) and observe that fine-tuning with SE-Attn yields better perplexity scores than S^2-Attn and SW-Attn. On short-context tasks from the LM Harness suite, all models perform similarly. On long context tasks from the LM Harness suite, SE-Attn outperforms S^2-Attn and SW-Attn.

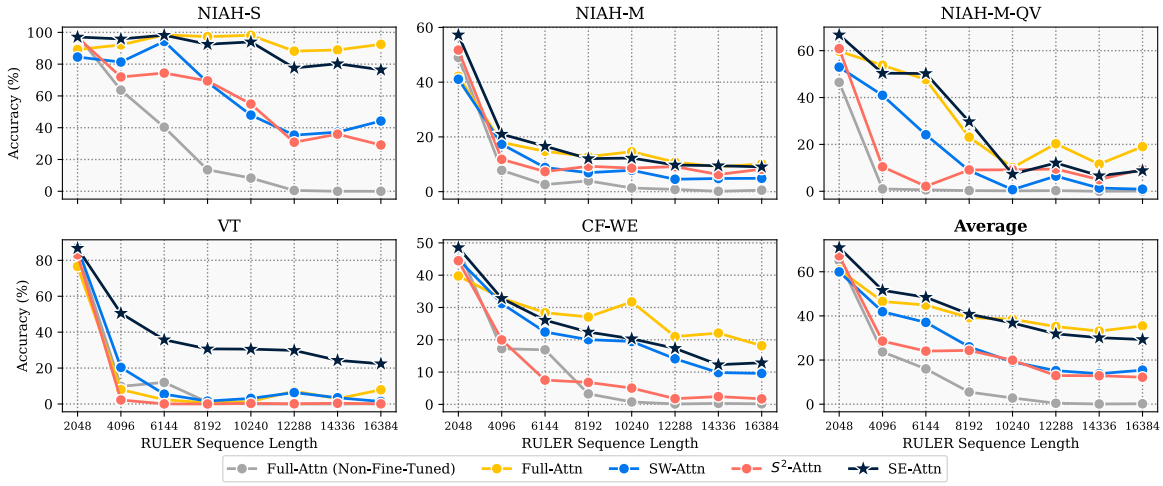


Figure 6.3: **Mamba-2-Hybrid RULER benchmark fine-tuned on natural language + code.** We fine-tune Mamba-2-Hybrid with different Attention layers on a dataset that consists of 70% natural language and 30% code and evaluate on RULER. Compared to fine-tuning on only natural language (as in Fig. 6.2), we see a substantial improvement on tasks like variable tracking (VT), and needle-in-a-haystack tasks (NIAH), both of which require strong recall capabilities enabled by fine-tuning with SE-Attn.

6.3.2.2 Fine-Tuning on PG-19

Next, we consider fine-tuning on the PG-19 [84] dataset. We provide perplexity results on the validation set, along with LM Harness task results in Tab. 6.4. Compare to fine-tuning on a different natural language dataset, and a natural language + code as in Tab. 6.2 and Tab. 6.3, here we obtain lower perplexity scores due to the lack of distribution shift. Interestingly, compared to fine-tuning on the previously mentioned datasets, we observe a slight degredation on LM Harness tasks. Moreover, in Fig. 6.4 we plot RULER performance when fine-tuning on PG-19, and here we also see a decay in performance compared to fine-tuning on the previous datasets. This suggests that the PG-19 dataset may be too far out of distribution for these long-context retrieval task. Nevertheless, we see that fine-tuning with SE-Attn yields the best performance across all of these tasks.

Attention	Eval Context Size (PG-19 PPL ↓)				Short Context Tasks (Acc ↑)							Long Context Tasks (Acc ↑)			
	2048	8192	16384	32768	ARC-E	ARC-C	Hella.	LAMB.	PIQA	WG	Avg.	SWDE	SQA	SNQA	Avg.
Non-fine-tuned	10.72	14.99	19.35	26.37	69.91	37.97	67.62	69.84	76.06	65.04	64.41	85.60	15.18	3.65	34.81
Full-Attn	10.73	10.04	10.14	10.91	67.34	37.12	66.80	68.62	74.32	62.67	62.81	84.88	25.35	18.46	42.90
SW-Attn	10.72	10.59	11.64	13.25	66.96	37.54	66.83	68.79	74.54	63.30	62.99	84.79	22.54	14.09	40.48
S^2-Attn	10.78	12.74	14.42	16.11	69.36	38.14	67.45	69.90	76.12	64.72	64.28	86.50	17.00	7.74	37.08
SE-Attn	10.73	10.22	10.84	12.28	67.42	37.88	66.48	69.22	73.88	61.88	62.80	85.15	23.06	16.46	41.55

Table 6.4: **Fine-tuning Mamba-2-Hybrid with SE-Attn on PG-19 outperforms fine-tuning with S^2-Attn and SW-Attn on long-context natural language tasks.** We fine-tune Mamba-2-Hybrid with a context size of 8192 using various Attention variants on PG-19. We evaluate PG-19 validation perplexity (PPL) and observe that fine-tuning with SE-Attn yields better perplexity scores than S^2-Attn and SW-Attn. On short-context tasks from the LM Harness suite, S^2-Attn has the strongest performance. On long context tasks from the LM Harness suite, SE-Attn outperforms S^2-Attn and SW-Attn.

6.3.3 Llama1 7B

In this section, we explore the efficacy of SE-Attn to Transformers. When fine-tuning Llama 7B models, we found that using a fixed chunk size of $M = 4096$ gave better downstream performance. All of our Llama models are fine-tuned with a context size of 16384. For SE-Attn, we use block size of 32, and a top-k of 8. For SW-Attn, we use a window size of 4096, and S^2-Attn uses the default parameters in [14]. All Attention variants use the same RoPE [99] scaling as in [14].

Perplexity. Fine-tuning Llama with SE-Attn leads to better generalization on context sizes smaller and larger than the one used for fine-tuning. In Tab. 6.5, we observe that fine-tuning with SE-Attn preserves the performance of the non-fine-tuned model at smaller context sizes, and offers greater generalization to larger context sizes than S^2-Attn and SW-Attn, as measured on PG-19 validation perplexity.

LM Harness. Fine-tuning Llama with SE-Attn yields a stronger performance on long-context tasks on the LM Evaluation Harness benchmark. As shown in Tab. 6.5, fine-tuning with SW-Attn, S^2-Attn, and SE-Attn all improve upon the non-fine-tuned model on shorter context tasks and perform similarly. However, SE-Attn gives a greater performance on long context tasks.

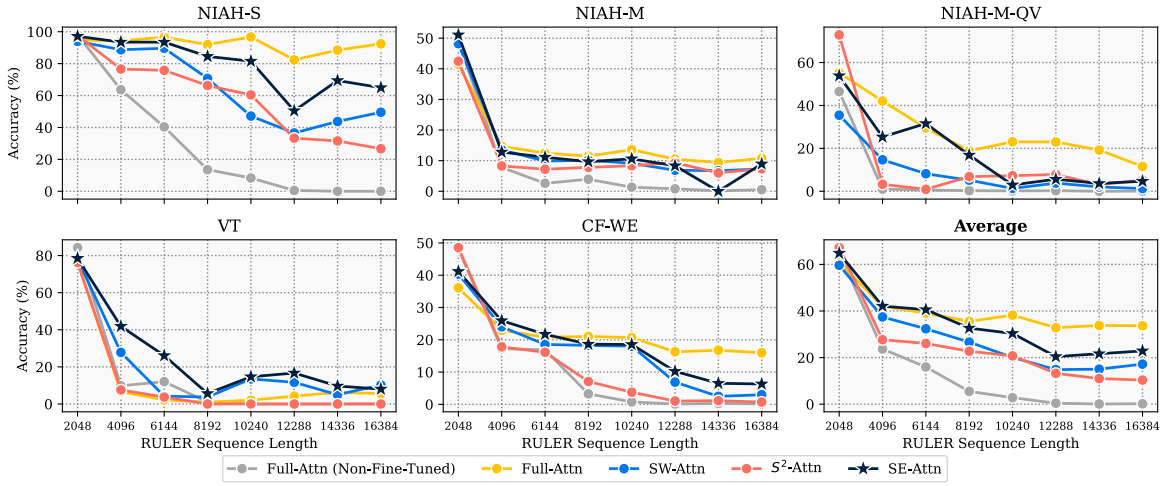


Figure 6.4: **Mamba-2-Hybrid RULER benchmark fine-tuned on PG-19.** We fine-tune Mamba-2-Hybrid with different Attention layers on the PG-19 [84] dataset and then evaluate on RULER. Compared to fine-tuning on other natural language datasets with a greater variety of text as in Fig. 6.2, here we observe a degradation in performance across all models, likely due to a distribution shift in the PG-19 data and the RULER tasks.

Attention	Eval Context Size (PG-19 PPL ↓)				Short Context Tasks (↑)							Long Context Tasks (↑)			
	2048	8192	16384	32768	ARC-E	ARC-C	Hella.	LAMB.	PIQA	WG	Avg.	SWDE	SQA	SNQA	Avg.
Non-fine-tuned	8.63	17.62	105.78	244.32	73.19	41.38	66.64	54.38	76.28	62.51	62.40	38.52	20.75	12.36	23.88
SW-Attn	8.80	8.17	8.35	10.32	74.20	43.09	75.00	71.96	77.42	67.48	68.19	82.81	24.41	21.06	42.76
S^2-Attn	9.32	8.64	8.58	10.42	74.20	42.58	73.84	69.94	77.80	67.72	67.68	80.56	23.36	19.65	41.19
SE-Attn	8.76	8.13	8.01	9.28	75.04	44.37	75.02	71.03	78.02	67.17	68.44	84.79	24.59	21.36	43.58

Table 6.5: **Fine-tuning Llama1 with SE-Attn outperforms fine-tuning with S^2-Attn and SW-Attn on natural language tasks.** We fine-tune Llama1 with a context size of 16384 using various Attention variants. Similar to applying SE-Attn to Mamba-2-Hybrid, here we again observe that fine-tuning Llama with SE-Attn improves upon SW-Attn and S^2-Attn .

RULER. Fine-tuning Llama with SE-Attn produces a model with stronger performance on RULER tasks than fine-tuning with SW-Attn and S^2-Attn, as illustrated in Fig. 6.5. On average, SW-Attn and S^2-Attn perform similarly, however, fine-tuning with SE-Attn improves performance by $\sim$5%. Interestingly, despite all models having a similar PPL up to a context size of 16k as shown in Tab. 6.5, we observe a substantial difference between the RULER performance of models fine-tuned with our SE-Attn and those fine-tuned with SW-Attn and S^2-Attn . We observe a similar PPL discrepancy for Hybrid models and expand on this

observation in Sec. 6.3.5.

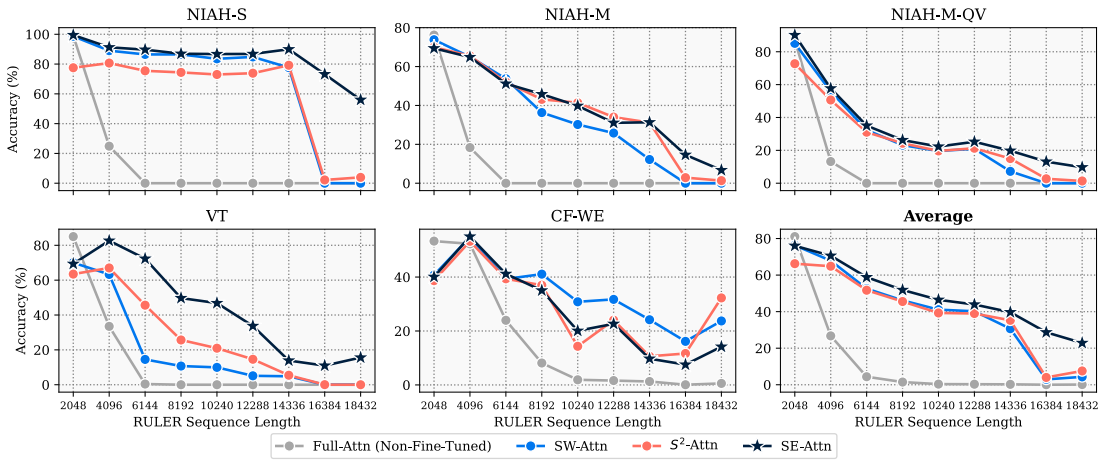


Figure 6.5: **Fine-tuning with SE-Attn outperforms SW-Attn and S^2-Attn on the RULER benchmark when applied to Llama1.** We fine-tune Llama1 with a context size of 16384 using various Attention variants. We average over eleven RULER tasks, as explained in Sec. 6.6.3.4. Fine-tuning with SE-Attn consistently outperforms SW-Attn and S^2-Attn even when evaluating on context sizes beyond the fine-tuning size.

6.3.4 Ablations on Mamba-2-Hybrid

In this section, we ablate over some of the design choices of SE-Attn on Mamba-2-Hybrid. For this analysis, we consider the RULER benchmark, as it is a strong indicator of performance on long-context tasks. As our metric, we use the average of the eleven RULER tasks defined in Sec. 6.6.3.4.

Does retrieval during training help? In Sec. 6.2.2, we introduced SE-Attn-NoMem, a variant of SE-Attn where we do not do any retrieval and process chunks independently. Naturally, we do not expect this to do well due to the lack of shared information across chunks. We confirm this in Fig. 6.6a, where we observe that SE-Attn-NoMem achieves a much lower performance than SE-Attn with retrieval. Furthermore, we also fine-tune with SE-Attn-Random, a variant of SE-Attn where we retrieve random memory blocks from the past. We observe that this improves upon SE-Attn-NoMem, indicating that retrieving some information from the past improves upon no information. However, SE-Attn-Random does

not do as well as SE-Attn, which decides which blocks to retrieve based on relevancy.

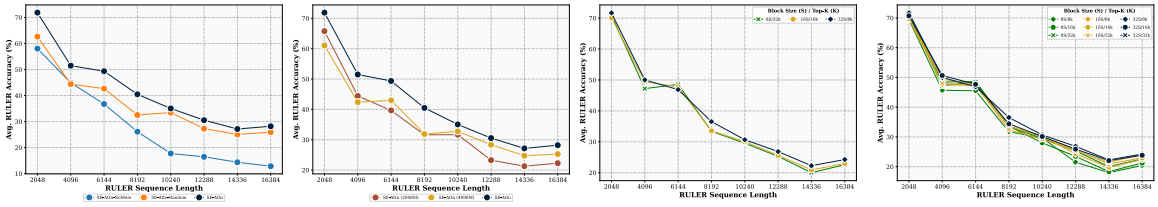
Chunk size. For Mamba-2-Hybrid, we found that using a random chunk size during each forward pass improved upon having a fixed chunk size. In Fig. 6.6b, we compare the performance of SE-Attn, which chooses a random chunk size in $\{2048, 4096\}$ during each forward call, to SE-Attn (2048M) and SE-Attn (4096M), which use a fixed chunk size of 2048 and 4096, respectively. We observe that SE-Attn with a fixed chunk size of 4096 improves upon SE-Attn with a fixed chunk size of 2048, but SE-Attn outperforms both. This suggests that training with random chunk sizes may have a regularizing effect, making the model more robust to changes in the input context size.

Block size and top-k. For a fixed expansion span size, we might expect that retrieving memory blocks at a finer granularity should perform better than retrieving a smaller number of large blocks; this is because retrieving a larger number of small blocks is as flexible as retrieving a small number of large blocks. However, as illustrated in Fig. 6.6c, we found that fine-tuning SE-Attn with larger block sizes and a smaller top-k gave the best performance. This is possibly due to the increased complexity of the retrieval task, which becomes more difficult as the number of possible blocks to retrieve increases. As we are fine-tuning with LoRA, we hypothesize that the model's capacity may not be sufficient to learn to retrieve effectively in this setting. As illustrated in Fig. 6.6d, we found that an expansion span populated with 256 retrieved tokens (32 memory blocks with 8 tokens in each) works best.

6.3.5 HyLoRA for Hybrid Models and the Subtle Pitfalls of Perplexity

We fine-tune our hybrid models using HyLoRA, which builds upon LoRA+ [14] by also training 1D convolution layers. As illustrated in Fig. 6.7, when evaluated on long-context RULER tasks, fine-tuning models with either Full-Attn or SE-Attn using HyLoRA gives the strongest performance.

Effect of LoRA rank. We next explore how the rank used for LoRA fine-tuning affects



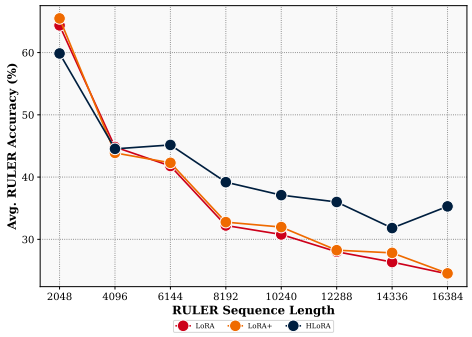
(a) (b) (c) (d)

Figure 6.6: **SE-Attn ablations on Mamba-2-Hybrid.** We plot the average of eleven RULER tasks as described in Sec. 6.6.3.4. (a): SE-Attn-NoMem processes chunks of tokens without retrieval, while SE-Attn-Random populates its expansion span by retrieving random memory blocks for each chunk. We observe that our Attention-based memory retrieval (SE-Attn) gives the strongest performance. (b): Using SE-Attn with a chunk size chosen randomly from $\{2048, 4096\}$ acts as a regularizer and outperforms SE-Attn with fixed chunk sizes of 2048 and 4096. (c): SE-Attn with larger memory blocks (i.e., more tokens per memory block) with a smaller top-k tends to do better than smaller block sizes with a larger top-k. (d): An expansion span consisting of 256 total tokens (8 memory blocks with 32 tokens in each) gives the strongest performance. 32S/32k is omitted due to memory constraints.

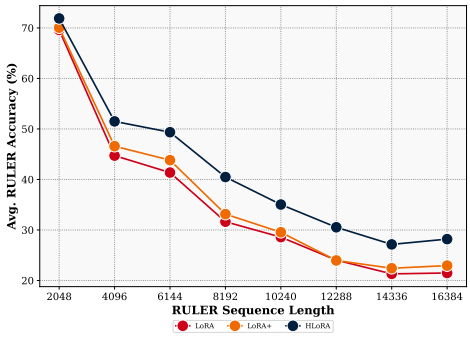
downstream performance on the RULER task. Given its stronger performance, we focus on our HyLoRA. In our previous Mamba-2-Hybrid experiments, we used a LoRA rank (r) of 32 and an α of 64 (for our Transformer experiments, we used $r = 8$ and $\alpha = 16$ as in [14]). In Fig. 6.8, we plot the average RULER results for Mamba-2-Hybrid models fine-tuned with HyLoRA using different ranks (we scale α to maintain the ratio $\alpha/r = 2$). Here, we observe that training with a larger rank improves downstream performance, and we start to see some saturation around $r = 64$.

In Tab. 6.6, we provide PG-19 perplexity results for Mamba-2-Hybrid trained with different LoRA variants. All models are fine-tuned with a context size of 8192, and evaluated with multiple context sizes. For a given context size, we do not see a substantial difference in the perplexity. This is similar to the observation in [14]. However, as discussed above, the rank can have a significant effect on downstream long-context tasks that require strong recall capabilities, as in RULER. Hence, while perplexity results may be promising, they are not necessarily indicative of performance on more complex long-context tasks.

Based on the analyses in this section, we conclude the following:



(a) Full-Attn



(b) SE-Attn

Figure 6.7: **HyLoRA outperforms LoRA and LoRA+ on Hybrid models.** We fine-tune Mamba-2-Hybrid with Full-Attn (a) and SE-Attn (b) using LoRA, LoRA+, and HyLoRA. We find that LoRA and LoRA+ perform sub-optimally. HyLoRA augments LoRA+ by additionally training the 1D convolution layers and yields strong performance regardless of which Attention mechanism is used during fine-tuning.

- Fine-tuning the 1D convolution layers, as we do in our HyLoRA, significantly improves performance on downstream long-context tasks that require retrieval, such as RULER.
- Fine-tuning with larger LoRA ranks improves performance up to a certain point—64 for our experiments.
- Perplexity may not be the most faithful metric for assessing performance on long-context downstream tasks. Instead, researchers should consider evaluating on more complex tasks, such as those in the RULER benchmark.

6.4 Runtime Analysis

Empirical Analysis. SE-Attn is much faster than Full-Attn and S^2-Attn on large contexts. In Fig. 6.9, we profile different Attention layers on various context sizes and see that SE-Attn is much faster than S^2-Attn and Full-Attn. Moreover, we see that the runtime of SE-Attn is similar to that of SW-Attn, indicating that our retrieval overhead is minimal despite the much longer Attention span.

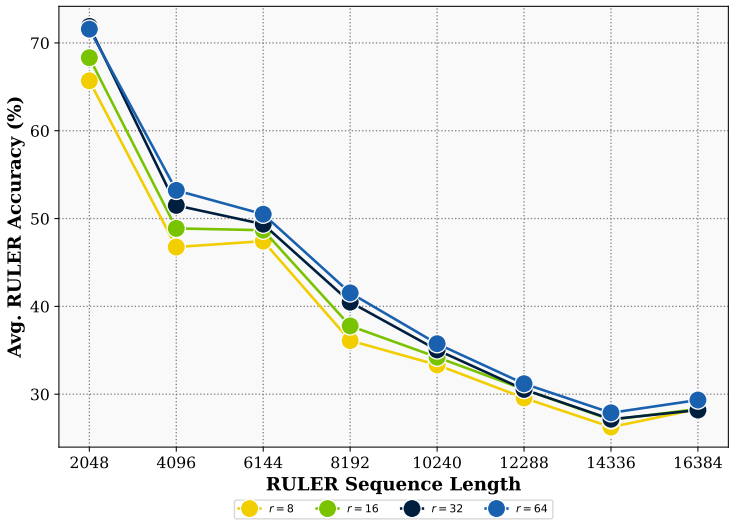


Figure 6.8: **Fine-tuning with a larger LoRA rank using HyLoRA improves performance on the RULER benchmark.** We fine-tune Mamba-2-Hybrid using HyLoRA with different LoRA ranks (we maintain a LoRA rank to alpha ratio of 2). We observe that fine-tuning with a larger rank produces stronger downstream results on RULER, with some saturation with a rank of 64.

Sequence Length	LoRA Method (Rank=32)			LoRA Rank (Method=HyLoRA)			
	LoRA	LoRA+	HyLoRA	8	16	32	64
2048	10.77	10.98	10.99	11.00	11.00	10.99	10.99
8192	10.29	10.49	10.45	10.45	10.46	10.45	10.44
16384	10.91	11.18	11.14	11.03	11.10	11.14	11.14
32768	12.34	12.66	12.64	12.37	12.53	12.64	12.64
65536	13.99	14.36	14.26	13.84	14.08	14.26	14.28

Table 6.6: **Mamba-2-Hybrid LoRA Ablations.** We fine-tune Mamba-2-Hybrid with a context size of 8192 with SE-Attn using different LoRA variants. We consider LoRA, LoRA+, and HyLoRA (ours) and evaluate perplexity on the PG-19 dataset. We observe that all LoRA variants yield similar perplexity results. However, as depicted in Fig. 6.7 and Fig. 6.8, different LoRA variants yield substantially different performances on more complex tasks.

Analytical Analysis. The runtime of SE-Attn is lower than Full-Attn and S^2-Attn on large contexts. For an input with sequence length L, SE-Attn first constructs $U = \frac{L}{S}$ memory blocks of size S. Attention is applied on each. Thus, the complexity of this operation is

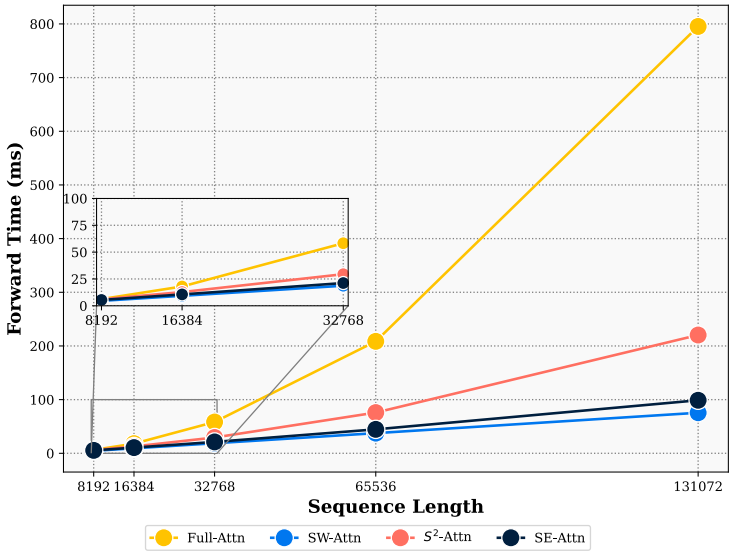


Figure 6.9: **SE-Attn is faster than S^2-Attn and Full-Attn, especially on long contexts.** The retrieval overhead of SE-Attn is minimal, with runtime similar to SW-Attn (which has a limited pre-determined Attention span).

$O(d_{\text{model}}US^2) = O(d_{\text{model}}LS)$. Next, the input sequence is split into $T = \frac{L}{M}$ chunks of size M. Cross-Attention is then applied between each chunk's query and each memory block's compressed representation; the cost of this is $O(d_{\text{model}}MTU) = O(d_{\text{model}}\frac{L^2}{S})$. Cross-Attention is then applied between each chunk's query tokens and the concatenations of the chunk's key tokens and memory tokens. Since SE-Attn retrieves K blocks with S tokens in each, the cost of this Cross-Attention for each chunk is $O(d_{\text{model}}M(SK + M)) = O(d_{\text{model}}M^2)$ since $SK < M$. The cost for all chunks is therefore $O(d_{\text{model}}TM^2) = O(d_{\text{model}}LM)$. The total cost of SE-Attn is therefore $O(d_{\text{model}}LS + d_{\text{model}}LM + d_{\text{model}}\frac{L^2}{S})$.

6.5 Discussion

In this chapter, we introduced SE-Attn , an efficient retrieval-based Attention mechanism used to expand the span of Hybrid SSM models, enabling them to operate over longer sequences. To adapt Hybrid SSM models to leverage this retrieval mechanism, we introduced HyLoRA,

a variant of LoRA designed for Hybrid SSMs. We demonstrated improved performance compared to existing baselines on a comprehensive set of long context tasks requiring strong recall capabilities.

We close with the limitations of our method. Although we have conducted experiments at a relatively large model scale (2.7B for Hybrid models and 7B for Transformers) and long contexts (up to 32k), testing our method for efficiently adapting larger models on longer contexts is paramount. Furthermore, while we utilize datasets that require modeling of long-range dependencies, we found that perplexity-based tasks do not faithfully measure models' capabilities to handle long contexts, and instead tasks like RULER provide better signals of long-context capabilities. However, RULER is mostly a synthetic dataset and does not cover more nuanced tasks that require reasoning over long documents. Validating our method on more complex long-range benchmarks is a promising area for future work.

6.6 Appendix

6.6.1 Training Details

Our fine-tuning recipe largely follows [14], with the exception of using a larger learning rate for SE-Attn, SW-Attn, and Full-Attn models (2×10^{-4} vs. the default 2×10^{-5} for S^2). We found that using a larger learning rate for S^2 did not improve its performance, as shown in Fig. 6.10, so we used the default 2×10^{-5} learning rate for all S^2 fine-tuning experiments.

All of our experiments are conducted on a single 8xA100 node. We fine-tune for 1000 steps with a total of 0.5B tokens. We fine-tune Mamba-2-H with 8192 tokens with 8 accumulation steps and a per-device batch size of 1. We fine-tune Llama1 with 16384 tokens with 4 accumulation steps and a per-device batch size of 1. We use FlashAttention-2 [17] and DeepSpeed Stage 2 [86].

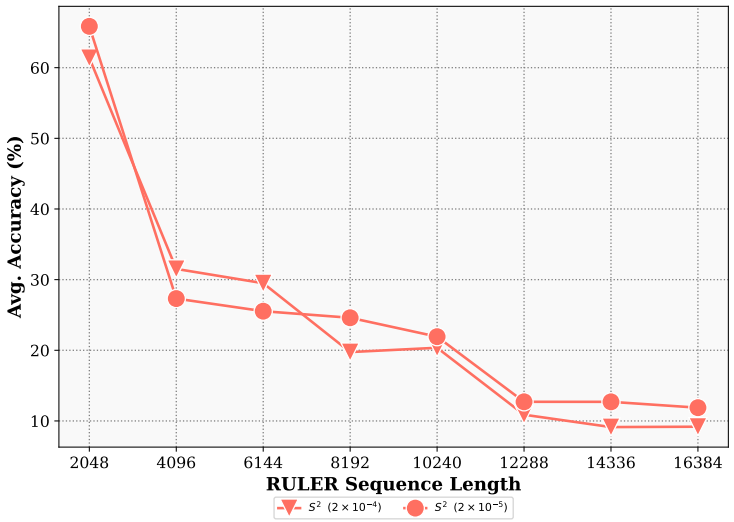


Figure 6.10: **A larger learning rate does not improve the performance of S^2.** Here we fine-tune a Mamba-2-Hybrid model using S^2-Attn with two different learning rates: 2×10^{-4} and 2×10^{-5}. The learning rate used in [14] is 2×10^{-5}, which we found to work well. For all other Attention layers, we found 2×10^{-4} to offer a slight improvement over 2×10^{-5}.

6.6.2 Retrieval with Landmark Tokens

Landmark Attention [74] was recently introduced as way for Transformer models to process long sequences by inserting 'landmark' tokens into the sequence whose representations would then be used as summaries of the blocks of tokens that came before them. At a high level, our approach in SE-Attn is similar. However, we simplify the process of compressing blocks of tokens by forgoing the use of landmark tokens and instead using Attention to summarize them, as described in Sec. 6.2.3. Moreover, to learn which blocks to retrieve, we do not rely on a complex grouped softmax function, and instead use a simple Cross-Attention score to ascertain relevance. In this way, we implement retrieval natively into the model's architecture.

We consider a variant of SE-Attn, which we refer to as SE-Attn-LM, that is inspired by Landmark Attention. Namely, instead of using our Attention-based compression to construct summaries of memory blocks, we insert a non-learnable 'landmark' token into each

memory block, and use the cross-attention between this token and the memory tokens as the summary of the memory block. We compare the performance of this variant to our summaries computing using average pooling of Attention scores (see Eq. (6.7)) in Fig. 6.11. Here, we see that SE-Attn yields better performance. We suspect this is because using a non-learnable landmark token to summarize memory blocks is too challenging of a task to accomplish using standard LoRA. While full fine-tuning (without LoRA) may improve the performance of SE-Attn-LM, is is beyond the scope of our work as we prioritize efficiency.

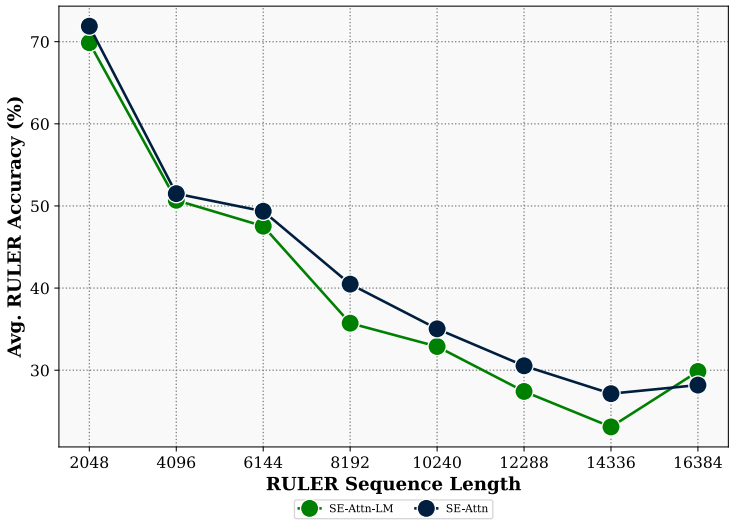


Figure 6.11: **Summarizing memory blocks via average pooling of attention yields stronger performance than summarizing them using 'landmark' tokens.** We fine-tune Mamba-2-Hybrid using our SE-Attn, and SE-Attn-LM, which summarizes memory blocks with a landmark token (similar to [74]). We find that our simpler SE-Attn produces a stronger model, likely due to the easier training task which does not require adapting the model to leverage landmark tokens for comression.

6.6.3 RULER Task Definitions

The RULER benchmark [46] consists of four different task categories: retrieval, multi-hop tracing, aggregation, and question answering. In this chapter, we focus only on the retrieval,

multi-hop tracing, and aggregation tasks (we provide question answering results on the LM Evaluation Harness benchmark). These three categories span eleven different tasks as explained below.

6.6.3.1 Needle-in-a-Haystack (NIAH) Tasks

RULER consists of 8 different NIAH tasks. These tasks embed "needles" in a string of noise. These needles are typically key-value pairs, and the goal is to return the value of a key. These tasks are characterized by six parameters:

- type_haystack (TH): This specifies the type of noise to embed the key in. The choices are "repeat" which constructs noise as in [74], "essay" which will use sentences from the Paul Graham essays [56], or "needle" in which case each sentence will define a new key-value pair.
- type_needle_k (TK): This specifies the type of the needle's key. The options are "words", in which case the key is a word (in the form of adjective-noun, e.g., spiritual-oven), or "uuids" in which case the key is a UUID.
- type_needle_v (TV): This specifies the type of the needle's value. It can either be "numbers" in which case the value is a 7-digit number, or it can be "uuids" in which case the value is a UUID.
- num_needle_k (NK): This specifies the number of key-value pairs to embed in the haystack.
- num_needle_v (NV): This specifies how many different values a key is assigned. If greater than 1, the goal is output all the values of they key.
- num_needle_q (NQ): This specifies the number of different keys the model must return the value for.

NIAH Task	TH	TK	TV	NK	NV	NQ
Single 1	repeat	words	numbers	1	1	1
Single 2	essay	words	numbers	1	1	1
Single 3	essay	words	uuids	1	1	1
Multikey 1	essay	words	numbers	4	1	1
Multikey 2	needle	words	numbers	1	1	1
Multikey 3	needle	uuids	uuids	1	1	1
Multivalue	essay	words	numbers	1	4	1
Multiquery	essay	words	numbers	1	1	4

Table 6.7: **RULER NIAH definitions.** The 'Needle-in-a-Haystack' (NIAH) tasks in the RULER benchmarks are defined by 6 parameters which modulate the difficulty of the tasks. We consider 8 different NIAH tasks as defined above (these are the default NIAH tasks in the RULER library).

6.6.3.2 Multi-hop Tracing Tasks

RULER considers a "variable tracking" task that is a form of coreference resolutions. In this task, a sequence of variables are defined throughout noisy text as in [74]. New variables are defined as previous ones, and a final value is assigned to a particular variable. The goal is to be able to trace back which variables have also been assigned the final value. We use the default num_chains=1 and num_hops=4 parameters

6.6.3.3 Aggregation Tasks

RULER considers two aggregation tasks, common words extraction (CWE), and frequent words extraction (FWE). In CWE, the context consists of list of words, and the goal is to return the most common words. We use the default parameters freq_cw=30, freq_ucw=3, and num_cw=10. In FWE, the context consists of random word strings, and the goal is to return the ones that appear the most frequently. We use the default alpha=2 parameter for this.

6.6.3.4 Aggregating RULER Tasks

We aggregate the eleven RULER tasks above into six groups:

1. *NIAH-S*: NIAH Single 1, NIAH Single 2, NIAH Single 3.
2. *NIAH-M*: NIAH Multikey 1, NIAH Multikey 2, NIAH Multikey 3.
3. *NIAH-M-QV*: NIAH Multivalue, NIAH Multiquery.
4. *VT*: Variable Tracking.
5. *CF-WE*: Common Words Extraction (CWE) and Frequent Words Extraction (FWE).
6. *Average*: The average of all eleven tasks above.

CHAPTER 7

Conclusion and Future Work

This thesis has demonstrated that dynamic modulation of model capacity, driven by progressive curricula, can reduce training costs while maintaining or improving model accuracy. We demonstrate the efficacy of such modulation across a broad spectrum of vision tasks and architectures. In several instances, our training techniques not only reduced training compute, but also yielded efficient models for deployment—typically in the form of compressed models. We hope that this exploration will encourage researchers to consider dynamic model modulation during training as a means for both reducing training costs and developing more robust and accurate models.

Although our exploration spans multiple areas of deep learning, our work is not exhaustive and there are numerous avenues for further research. First, except for our work on Early Model Pruning (Chapter 2), our curricula strategies are generally agnostic to the model's training performance. Future work could explore dynamically adjusting the curricula according to the model's performance on the training data (or a held-out dataset) during training. Unlike our current framework, which pre-defines the curriculum structure, this approach could yield even more accurate models. Second, our methods can be combined with other forms of model compression techniques. For instance, our work on Compound Scaling (Chapter 4), could be extended by exploring quantization as a means of compression rather than solely focusing on model pruning. Third, except for our work on Hybrid State Space Models (Chapter 6), our training frameworks were developed and tested for vision applications. Establishing whether Large Language Models exhibit Early Pruning Periods, for instance, represents an exciting

avenue for future research with significant potential to substantially reduce the financial costs of training.

The field of machine learning is evolving rapidly. Many of the methods that we developed in this thesis have been applied to models that are becoming increasingly irrelevant. However, the principles underlying these methods, such as the importance of efficiency and adaptability, remain timeless. As models become larger and more complex, the need for innovative strategies to optimize their training and deployment will only become more pressing. We hope that the insights and approaches presented in this work can help lay the foundation for more efficient deep learning methods that can help shape the future of accessible and sustainable deep learning.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 62

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 2

[3] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep neural networks. *arXiv preprint arXiv:1711.08856*, 2017. 4, 9, 11, 14, 28

[4] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2000. 14

[5] Anthropic. Model card: Claude 3, 2024. Accessed: 2024-11-11. 2

[6] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016. 47

[7] Dzmitry Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 1

[8] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021. 99

[9] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. 101, 108

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. 3, 64, 90

[11] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 2

[12] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021. 2

[13] Honglie Chen, Weidi Xie, Andrea Vedaldi, and Andrew Zisserman. Vggsound: A large-scale audio-visual dataset. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 721–725. IEEE, 2020. 93

[14] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023. xxv, 101, 102, 107, 108, 109, 114, 117, 118, 122, 123

[15] Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, and David Owen. The rising costs of training frontier ai models. *arXiv preprint arXiv:2405.21015*, 2024. 2

[16] Zihang Dai. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019. 101

[17] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023. 108, 122

[18] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024. 6, 102, 107

[19] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024. 6, 102, 107

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 1, 45, 66

[21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. xxviii, 2, 73, 74, 84, 97

[22] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 2

[23] Christoph Feichtenhofer, Yanghao Li, Kaiming He, et al. Masked autoencoders as spatiotemporal learners. *Advances in neural information processing systems*, 35:35946–35958, 2022. 97

[24] Kevin Galim, Wonjun Kang, Yuchen Zeng, Hyung Il Koo, and Kangwook Lee. Parameter-efficient fine-tuning of state space models. *arXiv preprint arXiv:2410.09016*, 2024. 107

[25] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. 109

[26] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 776–780. IEEE, 2017. xxx, 92, 94

[27] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2918–2928, 2021. 76

[28] Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*, 2024. 6, 102, 107

[29] Ian Goodfellow, Honglak Lee, Quoc Le, Andrew Saxe, and Andrew Ng. Measuring invariances in deep networks. *Advances in neural information processing systems*, 22, 2009. 68

[30] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1586–1595, 2018. 62

[31] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. 6, 101

[32] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. 6, 101

[33] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34:572–585, 2021. 101

[34] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. 4, 8, 9, 12

[35] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992. 8

[36] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022. 86

[37] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017. xxix, 74, 75, 84

[38] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 12, 42, 43, 45, 46, 66, 72, 77, 84

[39] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. *ICCV*, 2021. 68

[40] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *CoRR*, abs/1907.07174, 2019. 67

[41] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 89

[42] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997. 1

[43] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019. 34, 46, 73, 74, 77

[44] Andrew Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 34, 73, 74, 77

[45] Jeremy Howard. Training imagenet in 3 hours for usd 25; and cifar10 for usd 0.26, Apr 2018. 64

[46] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024. 109, 124

[47] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 100, 107

[48] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018. 72, 73

[49] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 646–661. Springer, 2016. 71, 99

[50] Po-Yao Huang, Vasu Sharma, Hu Xu, Chaitanya Ryali, Haoqi Fan, Yanghao Li, Shang-Wen Li, Gargi Ghosh, Jitendra Malik, and Christoph Feichtenhofer. Mavil: Masked audio-video learners. *arXiv preprint arXiv:2212.08071*, 2022. xxi, xxx, 5, 86, 87, 93, 97

[51] Po-Yao Huang, Hu Xu, Juncheng Li, Alexei Baevski, Michael Auli, Wojciech Galuba, Florian Metze, and Christoph Feichtenhofer. Masked autoencoders that listen. *Advances in Neural Information Processing Systems*, 35:28708–28720, 2022. xxx, 90, 93, 94, 97

[52] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015. 34, 43

[53] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 42, 45

[54] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26, 2013. 68

[55] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960. 101

[56] Gregory Kamradt. Needle in a haystack - pressure testing llms., 2023. 125

[57] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 2

[58] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020. 101

[59] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 12, 42, 43, 45

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 1

[61] Ananya Kumar, Percy S Liang, and Tengyu Ma. Verified uncertainty calibration. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 68

[62] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989. 8

[63] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024. 6, 102, 107

[64] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755. Springer International Publishing, 2014. 74

[65] Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. Fq-vit: Fully quantized vision transformer without retraining. *ArXiv*, abs/2111.13824, 2021. 51

[66] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. xxviii, 2, 73, 74, 84, 90, 97

[67] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763, 2017. xxvi, 37, 45, 57

[68] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 99

[69] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017. 99

[70] Sachin Mehta, Farzad Abdolhosseini, and Mohammad Rastegari. Cvnets: High performance library for computer vision. In *Proceedings of the 30th ACM International Conference on Multimedia*, MM '22, 2022. xxviii, 61, 63, 66, 74, 83

[71] Sachin Mehta, Saeid Naderiparizi, Fartash Faghri, Maxwell Horton, Lailin Chen, Ali Farhadi, Oncel Tuzel, and Mohammad Rastegari. Rangeaugment: Efficient online augmentation with range learning. *arXiv preprint arXiv:2212.10553*, 2022. 66

[72] Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. *CoRR*, abs/2110.02178, 2021. 63, 70

[73] Matthias Minderer, Josip Djolonga, Rob Romijnders, Frances Hubis, Xiaohua Zhai, Neil Houlsby, Dustin Tran, and Mario Lucic. Revisiting the calibration of modern neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15682–15694. Curran Associates, Inc., 2021. 68

[74] Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers. *arXiv preprint arXiv:2305.16300*, 2023. xxv, 106, 123, 124, 125, 126

[75] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024. 101

[76] Dianwen Ng, Yunqi Chen, Biao Tian, Qiang Fu, and Eng Siong Chng. Convmixer: Feature interactive convolution with curriculum learning for small footprint and noisy far-field keyword spotting. *arXiv preprint arXiv:2201.05863*, 2022. 93

[77] Elvis Nunez, Thomas Merth, Anish Prabhu, Mehrdad Farajtabar, Mohammad Rastegari, Sachin Mehta, and Maxwell Horton. On the efficacy of multi-scale data samplers for vision applications. *arXiv preprint arXiv:2309.04502*, 2023. 90

[78] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019. 99

[79] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani,

Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith 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. 45

[80] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 62

[81] Karol J Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1015–1018, 2015. 93

[82] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 82

[83] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10425–10433, 2020. 72, 73, 83, 84

[84] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019. xxiii, 108, 113, 115

[85] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021. 2

[86] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020. 122

[87] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? *CoRR*, abs/1902.10811, 2019. xxviii, 68, 70

[88] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. 63

[89] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the*

IEEE/CVF conference on computer vision and pattern recognition, pages 10684–10695, 2022. 2

[90] Nicolas Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. *Advances in neural information processing systems*, 25, 2012. 68

[91] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 1

[92] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 34, 46, 73, 74, 77

[93] Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019. 24

[94] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013. 24

[95] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pages 2217–2225. PMLR, 2016. 68

[96] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 12, 46

[97] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. 70

[98] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13693–13696, 2020. 2

[99] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. 105, 114

[100] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023. 101

[101] Ilya Sutskever. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014. 1

[102] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 1

[103] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 46

[104] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 72, 73

[105] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021. 64, 90

[106] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *Advances in neural information processing systems*, 35:10078–10093, 2022. 86

[107] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 18–24 Jul 2021. 2, 46

[108] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 32–42, 2021. 46

[109] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 108

[110] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 44, 49

[111] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 90, 94, 101, 108

[112] Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, et al. An empirical study of mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024. 108

[113] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. Videomae v2: Scaling video masked autoencoders with dual masking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14549–14560, 2023. 86

[114] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. 101

[115] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018. 93

[116] Chen Wei, Karttikeya Mangalam, Po-Yao Huang, Yanghao Li, Haoqi Fan, Hu Xu, Huiyu Wang, Cihang Xie, Alan Yuille, and Christoph Feichtenhofer. Diffusion models as masked autoencoders. *arXiv preprint arXiv:2304.03283*, 2023. 89, 90, 98

[117] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *International conference on machine learning*, pages 564–572. PMLR, 2016. 62

[118] Wei Wen, Feng Yan, Yiran Chen, and Hai Li. Autogrow: Automatic layer growing in deep convolutional networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 833–841, 2020. 78

[119] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 1

[120] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. 62

[121] Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11217–11227. PMLR, 18–24 Jul 2021. 5, 33, 35, 46

[122] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 42, 44, 49, 52

[123] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023. 101

[124] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024. 101

[125] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1803–1811, 2019. xvi, xxvi, 34, 37, 38, 41, 42, 43

[126] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2018. xxvi, 34, 37, 42, 43

[127] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon 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. 99

[128] Luca Zancato, Arjun Seshadri, Yonatan Dukler, Aditya Golatkar, Yantao Shen, Benjamin Bowman, Matthew Trager, Alessandro Achille, and Stefano Soatto. B'mojo: Hybrid state space realizations of foundation models with eidetic and fading memory. *arXiv preprint arXiv:2407.06324*, 2024. 101, 102, 107

[129] Vladislav Zavadskyy. Lots of code, 2017. Accessed: 2024-10-27. 112

[130] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 99

[131] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. 39