REDUCING TRAINING COST AND IMPROVING INFERENCE SPEED THROUGH NEURAL NETWORK COMPRESSION

by

Cody Blakeney, B.S.

A dissertation submitted to the Graduate College of
Texas State University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
with a Major in Computer Science
May 2023

Committee Members:

Ziliang Zong, Chair

Yan Yan

Tanzima Islam

Vangelis Metsis

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

This dissertation is dedicated to my mother Sharon Frances Chapman Blakeney. Here's to making the world a brighter place. Sláinte!

ACKNOWLEDGEMENTS

First, I would like to acknowledge my wife, Bridget Blakeney, who not only stuck by me over the past five years of this crazy journey but even decided to become Mrs. Blakeney during this time.

I would also like to express my sincere gratitude to my advisor, Dr. Ziliang Zong, for his guidance, support, and encouragement throughout the course of this research. His expertise and valuable insights have been instrumental in shaping this thesis.

I also extend my thanks to the members of my thesis committee, Dr. Tanzima Islam, Dr. Yan Yan, and Dr. Vangelis Metsis for their valuable feedback and constructive criticism.

And finally, I want to acknowledge my good friends and colleagues Gentry Atkinson and Keshav Bhandari for always lending me their ears over beers on matters academic and personal. More than one of my publications during this time was born from the seeds of these conversations. I certainly wouldn't be here without their patience to indulge my hair-brained ideas.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

Figure **LIST OF FIGURES** **Page**

ABSTRACT

As AI models have become integral to many software applications used in everyday life, the need for ways to run these computationally intensive applications on mobile and edge devices has grown. To help solve these problems, a new research area of neural network compression has emerged. Techniques like quantization, pruning, and model distillation have become standard. However, these methods have several drawbacks. Many of these techniques require specialized hardware for inference, reduce robustness to adversarial examples as well as amplify existing model biases, and require significant retraining done in a time-consuming iterative process.

This dissertation explores several shortcomings in model compression, how to address them, and ultimately provides a simple, repeatable recipe for creating high-quality neural network models for inference. It shows that model pruning is not a true compression process, and in fact, pruning causes model representations to change such that they are as different as a new model trained at random. It explores how pruning can cause unwanted effects of pruning and how knowledge distillation can be used to mitigate these effects. It demonstrates how model compression for more accurate fidelity to the original can be achieved while also deconstructing it into a highly efficient and parallelized process by replacing sections of the model in a block-wise fashion. Finally, it examines how knowledge distillation can be used during the training process such that it both improves training efficiency, amortizes the cost of hyper-parameter searchers, and can provide state-of-the-art compression results.

I. INTRODUCTION

Deep neural networks (DNNs) have demonstrated remarkable success in various challenging tasks in natural language processing, speech recognition, and computer vision. These networks have been widely adopted to support a range of powerful applications, including machine translation of social media messages into multiple languages [2, 3], enhancing image previews [4], providing virtual green screen backgrounds for video conferencing, and facilitating the capture of professional-quality photographs by amateurs [5, 6].

Recent advances in generative models, such as chatGPT [7, 8] and Text-to-Image generation with diffusion models [9, 10], suggest that we are on the cusp of another transformative revolution in this field. However, DNNs of the past decade, and those of the future, are known to be computationally intensive, memory-intensive, and power-hungry, making them difficult to deploy on edge devices such as mobile phones and IoT equipment. Furthermore, training DNNs requires significant resources. Therefore, it is critical to developing compression techniques that can significantly reduce the computation, size, and power consumption of DNNs, to facilitate their deployment on resource-constrained systems and reduce the cost of compression training.

Currently, the state-of-the-art compression techniques for DNNs can be broadly classified into three categories: (1) Quantization [11, 12, 13, 14], by far the most widespread, which reduces the memory footprint and computation demand of DNNs by representing each weight using a smaller number of bits. (2) Network pruning, which includes unstructured pruning [15] and structured pruning [16]. Both aim to eliminate unimportant weights in a DNN to reduce its size and computation demand. The difference is that unstructured pruning sets unimportant weights to zero arbitrarily, while structured pruning can remove entire kernels or filters when

necessary. (3) Knowledge distillation [17], which typically distills the knowledge learned from a large, complex teacher model (or an ensemble of models) to a simpler student model with less computation, smaller size, and similar accuracy.
However, current compression techniques have several drawbacks:

1. **Training time**: training time for compressing models is significant.
   - **Pruning** generally requires retraining done in an iterative process, which is both time-consuming and energy-intensive. This can account for 20-40% of the entire training budget [18].
   - **Knowledge Distillation**, on the other hand, requires that after first having trained a larger model (or models), you must now train from scratch a new smaller model (or student). Loading of the model or ensemble of the models can slow the training time anywhere from 30-200% as these models generally take more time to complete the forward pass than the student model.

   Since many machine learning models are trained often to respond to shifts in data distribution, the running time and energy consumption of these compression methods are also critical to consider.

2. **Specialized Hardware**: Many techniques, like unstructured pruning and more extreme quantization, require specialized hardware to run or realize speed up.

3. **Quantization**: Arguably, the most successful compression methodology is quickly approaching the limits of what is possible. Most models are already trained in 16-bit, 8-bit training is soon becoming standard [19], and the literature points to 4-bits as the extreme limit of what is possible without a significant reduction in quality [20, 21]. It is unlikely that this trend will continue to provide significant savings in the future.

4. **Abundant Compression Choices**: The design space for selecting an appropriate model and compressing it to a level suitable for specific deployment hardware is extensive. The field of neural architecture search has emerged to address the systematic design of efficient neural networks. However, the computational demands of these techniques are significant, limiting their implementation to only a select few organizations with substantial computational resources.

Dissertation Organization

The chapters of this dissertation consist of four comprehensive studies that explore and address the challenges outlined above. These studies were conducted in a way that builds on the findings of the previous studies, allowing for a deeper understanding of model compression and the potential solutions that may be effective in addressing them.

- Chapter I summarizes the motivation and major contributions of this dissertation.

- Chapter II shows that model pruning is not a true compression processes [22]. This study investigates the effectiveness and limitations of unstructured neural network pruning to reduce large neural networks' size and computational demands. We study three unstructured pruning methods and use the Singular Vector Canonical Correlation Analysis (SVCCA) to analyze the layer representations of pruned and original models. We show that model representations change significantly during the pruning and retraining process, and pruning should not be viewed as a neutral compression process.

- Chapter III explores how pruning impacts the implicit bias of models, how to measure the impact, and how knowledge distillation can be used to mitigate

these effects [23, 24]. After establishing that pruning is more than a neutral compression process, this study examines the bias introduced into neural networks when performing model pruning and how to mitigate it. It demonstrates that knowledge distillation can effectively mitigate induced bias in these networks, even when working with unbalanced datasets. The study also finds that model similarity has strong correlations with bias in pruned neural networks, which can be used to explain the occurrence of bias in these networks.

- Chapter IV examines an alternative to structured pruning and traditional knowledge distillation via block-wise distillation for replacing individual components of a neural network [25]. It then demonstrates this process can be deconstructed into highly parallelizable tasks [26]. This approach has the advantage of loading only a portion of the network graph into memory, making the knowledge distillation significantly more energy and resource-efficient. It also presents scheduling and system adjustments that can be made such that these individual jobs scale well at both the single-node and distributed system levels.

- Chapter V investigates how distillation can not only compress models, but also accelerate the training of deep neural networks. It explores the distillation of same-sized models and how doing so can amortize the training cost of hyper-parameter sweeps, which allows for more efficient training of models of any size while also achieving SOTA performance on "compressed" models. The effectiveness of this methodology has been evaluated using both vision and language tasks.

- Chapter VI summarizes this dissertation and discusses future work for efficient deep learning.

Contributions

Understanding Pruning and Model Representations

1. This study shows that pruned neural networks evolve to substantially different representations while still maintaining similar accuracy, suggesting that pruning is not a passive compression process without learning new knowledge.

2. It demonstrates that initialized sparse models can achieve good accuracy, overturning the conventional wisdom that models must be first trained in full capacity before they undergo pruning.

3. It finds that sparsity structures discovered by pruning are not inherently important or useful, suggesting that new methodologies can be developed to speed up the pruning/training process by training models with some amount of sparsity from the beginning.

Mitigating Bias in Pruned Networks With Knowledge Distillation

1. This study demonstrates that knowledge distillation can mitigate bias and improve accuracy in pruned neural networks, especially when working with unbalanced datasets.

2. It introduces the Combined Error Variance (CEV) and Symmetric Distance Error (SDE) metrics for evaluating bias in compressed neural networks.

3. It provides evidence that model representation similarity (as measured by SVCCA) is correlated with bias in pruned neural networks, providing motivation for future knowledge distillation research.

Block-wise Distillation and Its Efficient Parallelization

1. This study develops a parallel blockwise distillation algorithm that performs independent blockwise distillation, using depthwise separable layers as an efficient replacement block architecture and addressing limiting factors that affect parallelism.

2. It achieves 3x speedup and 19% energy savings on VGG distillation, and 3.5x speedup and 29% energy savings on ResNet distillation without compromising accuracy.

3. It further improves the speedup of ResNet distillation to 3.87 when implemented on a distributed cluster with four RTX6000 GPUs.

4. It creats a more efficient and generalizable student model through the use of independently trained replacement blocks and an optimized replacement block architecture.

Efficient Training and Compression with Knowledge Distillation

1. This study demonstrates that training with distillation is almost always more efficient than training without distillation, even when using the poorest-quality model as a teacher, in both ResNet-50 and BERT.

2. It acheives state-of-the-art (SOTA) distillation performance for ResNet-18 and ResNet-34.

3. It achieves a training speedup of 1.96x in ResNet-50 and 1.42x on BERT when evaluated on GLUE.

II. IS PRUNING COMPRESSION?

Introduction

Nowadays, deep artificial neural networks have undoubtedly become the most promising method in solving many challenging computer vision problems [27, 28]. However, the model size and parameter space of successful deep neural networks are typically massive, which prevents them from being deployed on edge-devices (*e.g.,* mobile phones) with limited resources.

To address this problem, pruning has been studied extensively in the literature [29, 30, 31, 32, 33, 34] as an effective technique that can significantly reduce theoretical model size, computation demand and energy consumption of large neural networks without compromising accuracy. The key idea of pruning is to eliminate or mask non-essential components (*e.g.,* less important neurons or negligible weight values) of a deep neural network. Exemplary pruning methods include the early work presented by [35] and a more recent work by Han *et al.* [30]. Since then, a variety of pruning methods, such as parameter pruning and sharing [30, 31, 29], low-rank factorization [36, 37, 38], and compact convolutional filters [39, 40], have been published (ref. Related Work for details). Despite all this progress, our fundamental understanding about pruning is still in its infancy. For example, existing pruning theories and techniques tend to agree with the following hypotheses:

- Hypothesis 1: Pruning is an iterative compressing process. It compresses the original model to a subnet and the representation of the original network remains similar, which is why a pruned network can achieve similar accuracy as the original network.
- Hypothesis 2: A complex model needs to be trained first before it can be pruned. Models with sparsity at initialization are unlikely to succeed.

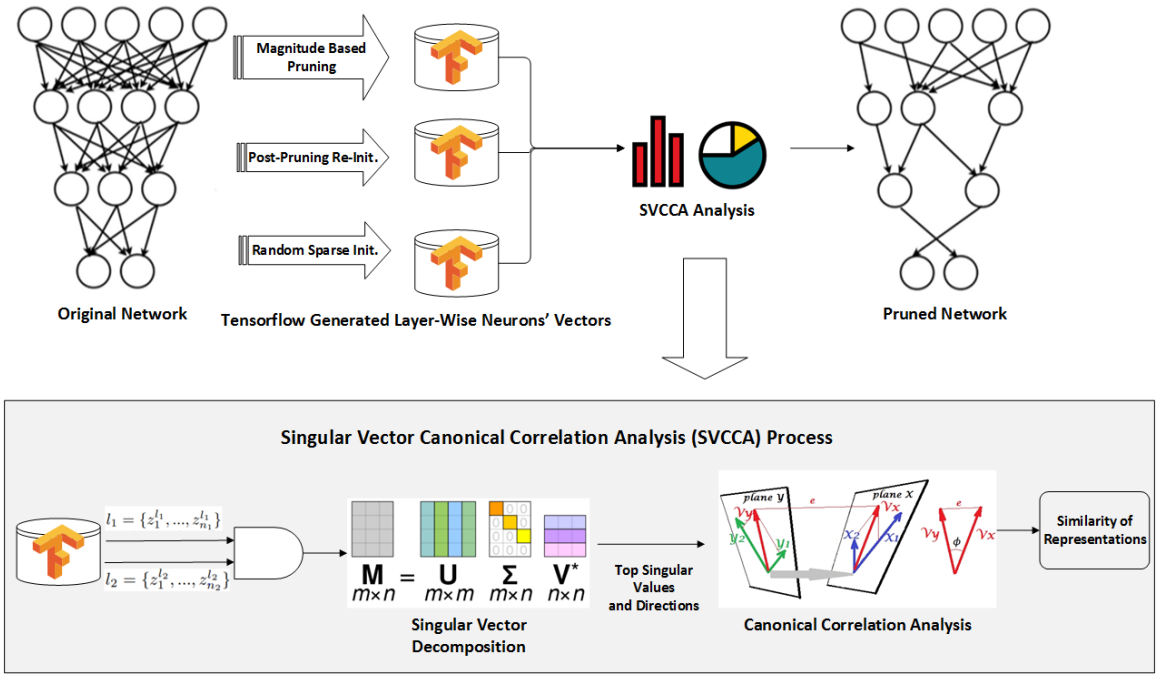


Figure 2.1: Overview of proposed approach.

- Hypothesis 3: Once a deep neural network is pruned and recovered to good accuracy (after retraining), its pruned model structure and weights carry important information and should help improve models trained from scratch.

As more pruning methods being developed, it is time to rethink if these hypotheses that are derived from previous research and practices still hold true and ask the following fundamental questions about pruning. Do the pruned neural networks contain the same representations as the original network? Is pruning truly a compression process? Does pruning only work on trained neural networks? Do we really need sophisticated pruning strategies? What happens if sparsity is chosen randomly?

This chapter strives to answer these questions. Specifically, we analyze three fine-grained pruning methods (magnitude based pruning, post-pruning re-initialization, and random sparse initialization). In our experiments, the Singular Vector Canonical Correlation Analysis (SVCCA) tool [41] is utilized to study and

contrast layer representations of pruned and original ResNet [28], VGG, and ConvNet [42] models. We find that: 1) Pruning is not a passive compression process without learning new knowledge. Rather, the pruned model is capable of evolving proactively to survive in a dramatically changed environment, which is done by learning and transforming to more effective representations when aggressive pruning is occurring. 2) Models initialized with sparsity structures can achieve reasonably good accuracy compared to well-engineered pruning methods. 3) Sparsity structures discovered by performing unstructured pruning on models are not inherently important or useful.

Figure 2.1 illustrates the overview of our proposed approach. We first take an original network (ResNet, VGG, or ConvNet) and prune it using magnitude based pruning, post-pruning re-initialization, or random sparse initialization (ref. section 4 for details) at different sparsities respectively. The neurons' vectors at each layer, which are generated by Tensorflow during the training and pruning process, are then stored and processed by the SVCCA analysis tool created by Google [41] (ref. section 3 for details). Lastly, the accuracy of different pruning methods and the similarity of different representations are analyzed and presented in section 5.

Related Work

Deep neural networks have become extremely popular and been successfully used in different applications recently. However, most designed neural networks in machine learning and computer vision field [27, 28] focused on accuracy rather than efficiency. There has been some work on reducing the storage and computation cost by model compression. For example, Lecun Yann had done early work about pruning network which has been investigated in the optimal brain damage work [35]. The basic idea is that different neurons contribute differently in the network. The low ranking neurons can be removed, which results in a smaller and faster network. Recently, Mariet et al. [43] proposed to identify a subset of diverse neurons that do not require retraining to

reduce redundancy of the network. In this section, we first review the model compression [44] from three aspects, *i.e.*, parameter pruning and sharing, low-rank factorization, and compact convolutional filters. Afterwards, we briefly discuss the Lottery Ticket Hypotheses and the existing work to explore the nature of network pruning.

Parameter Pruning and Sharing reduces redundant parameters which are not sensitive to the performance. It can be used in both convolutional layers and fully connected layers. Han *et al.* [30] proposed to reduce the total number of parameters and operations in the entire neural network. Chen *et al.* [31] introduced a HashedNet model that used a low-cost hash function to group weights into hash buckets for parameter sharing. Lebedev *et al.* [29] imposed group sparsity constraint on the convolutional filters to achieve structured brain damage. Zhou *et al.* [32] proposed a group-sparse regularizer on neurons during the training stage to learn compact CNNs with reduced filters. Magnitude-based weight pruning methods are computationally efficient and scalable to large networks and datasets, which makes it become a popular approach for network pruning. See *et al.* [33] showed that weight pruning with retraining was a highly effective method of compression and regularization on a state-of-the-art NMT system, compressing the model to 20% of its size with no loss of performance. Narang *et al.* [34] proposed a technique to reduce the parameters of a network by pruning weights during the initial training of the network. At the end of training, the parameters of the network were sparse while accuracy was still close to the original dense neural network. The network size was reduced by 8x and the time required to train the model remained constant. Anwar *et al.* [45] introduced a three-level pruning of the weights and locate the pruning candidates using particle filtering, which selected the best combination from a number of random generated masks. Polyak *et al.* [46] detected the less frequently activated feature maps with sample input data for face detection applications.

Low-rank Factorization uses matrix or tensor decomposition to estimate the informative parameters. It can be used in both convolutional layers and fully connected layers. Rigamonti *et al.* [36] introduced learning separable 1D filter following the idea of dictionary learning. Jaderberg *et al.* [37] proposed using different tensor decomposition schemes to achieve double speed for a single convolutional layer with 1% drop in classification accuracy in text recognition. Tai *et al.* [38] proposed a new algorithm for computing the low-rank tensor decomposition for training low-rank constrained CNNs from scratch.

Compact Convolutional Filters is to design special structural convolutional filters to save parameters. The approaches can be only used for convolutional layers. Cohen *et al.* introduced Group equivariant Convolutional Neural Networks (G-CNNs), a natural generalization of convolutional neural networks that can reduce sample complexity by exploiting symmetries. G-CNNs use G-convolutions, a new type of layer that enjoys a substantially higher degree of weight sharing than regular convolution layers. Zhai *et al.* [39] proposed doubly convolutional neural networks (DCNNs), which significantly improved the performance of CNNs by further exploring this idea. Instead of allocating a set of convolutional filters that were independently learned, a DCNN maintained groups of filters where filters within each group were translated versions of each other. Shang *et al.* [40] integrated CRelu into several state-of-the-art CNN architectures and demonstrated improvement in their recognition performance on the CIFAR-10/100 and ImageNet datasets with fewer trainable parameters.

Lottery Ticket Hypotheses and The Nature of Network Pruning. After the renewed interest in pruning proposed by [30], many researchers have begun studying what is happening during the process of pruning and how it works. A notable recent work which focuses on unstructured iterative pruning [47] proposes the Lottery Ticket Hypothesis which states "Dense, randomly-initialized, feed-forward networks contain sub-networks (winning tickets) that - when trained in isolation – reach test accuracy

comparable to the original network in a similar number of iterations." These subnets as the paper articulates have particularly lucky weights and connections such that they are able to converge through gradient descent towards accuracies as better than the model as a whole. The paper suggests that the effectiveness of deep learning is due in part to the high number of weights and layers which increases the odds of initializing some subnets within the model that can find some good local minima. While the authors claim that it is a combination of both weights and connections that make these "winning tickets", it is unclear to what role, the weights and connections respectively, play. Our work tries to address more thoroughly on how network topology affects accuracy and learned representations.

Another work [48] trying to explain the nature of pruning focuses primarily on structured pruning. They find the resulting model architectures from the structured pruning process are well suited for training from scratch. That is, the resulting architectures (not the found weights) are useful. Their work suggests that structured pruning can be considered as one type of neural network architecture search. The contradictions between their findings and that from [47] show that there is a fundamental difference in what structured and unstructured pruning algorithms do with the original model.

Nevertheless, none of the prior work has studied the fundamental questions that we asked previously. The lack of understanding about the nature of pruning, the learned representations during the pruning and retraining process, and the trend of developing more sophisticated pruning algorithms raises concerns to us. By exploring these questions, this study is distinct from all previous work on neural network pruning.

Singular Vector Canonical Correlation Analysis (SVCCA)

Most previous work has been focused on improving different pruning methods without deep understanding about how network pruning really works. In order to take a deep dive into the learning process, it is essential to understand what representations are learned at every stage of training and pruning. Additionally, it is critical to have a tool that can quantitatively compare two representations and evaluate how similar or different they are. In this study, we leverage the Singular Vector Canonical Correlation Analysis (SVCCA) tool created by [41] to compare and analyze the learned representations of different layers. This section summarizes the SVCCA process and briefly discusses how it functions.

In SVCCA, a series of activations of a neuron is treated as a vector and the model's layers are treated as subspaces that those vectors will span. SVCCA is able to compare the learned representations of any two layers and tell whether or not they have learned the similar or different representations. It does this by combining Singular Vector Decomposition (SVD), which reduces the dimensions of layer subspaces, and Canonical Correlation Analysis (CCA), which maximizes a projection between the two layers that results in the highest correlation. This process allows layers with different configurations of weights, neurons, biases and activation functions to be analyzed. For a given dataset $X = \{x_1, ..., x_m\}$ and a neuron i on layer l the output of that neuron on the entire dataset is defined as the vector z_i^l. SVCCA takes input from two layers $l_1 = \{z_1^{l_1}, ..., z_{n_1}^{l_1}\}$ and $l_2 = \{z_1^{l_2}, ..., z_{n_2}^{l_2}\}$ where n_1 and n_2 are the number of neurons in their respective layers. SVD is performed on each layer to get new subspaces l_1' and l_2' where $l_1' \subset l_1$, $l_2' \subset l_2$. Next, l_1' and l_2' are linearly transformed to be as aligned as possible and correlation coefficients are calculated. SVCCA outputs aligned directions $(\tilde{z}_i^{l_1}, \tilde{z}_i^{l_2})$ and how well they correlate. The higher the correlation value ρ_i is, the more similar the two aligned directions are.

In this study, we primarily focus on the SVCCA similarity $\bar{\rho}$. $\bar{\rho}$ is the mean of the ρ_i values from the top CCA directions and essentially describes how similar the representations of two layers are with each other. The $\bar{\rho}$ values can be used to observe how the learned representations change overtime if similarties of layers are calculated at different time steps. In other words, we use layer similarities as the metric to evaluate the changes in learned representations as models undergo the pruning and retraining process.

Pruning Methods

Despite various pruning methods published in the literature, we only evaluate three unstructured pruning methods (magnitude based pruning, post-pruning re-initialization, and random sparse re-initialization) in this study. They are carefully selected to focus on answering the key questions about pruning: 1) do the representations remain identical during the pruning and retraining process? 2) How important is it to carefully design pruning methods? 3) How important is the structure learned from the pruning process

Magnitude Based Pruning

The magnitude based pruning presented by [30] serves as the baseline method, which first trains an original model to convergence, then prunes each layer by removing the weights with the smallest absolute value until the desired sparsity is reached, and finally retrains the pruned model to recover to a similar accuracy as the original model. The important distinction between our method and Han's approach [30] is that pruning is done layer by layer (not globally on the whole model). We use both an iterative and single shot pruning method, which aims to identify what knowledge is carried over from the original model. We also omit pruning the final fully connected layer as it represents only a tiny portion of the overall weights of a

model. We speculate that magnitude based pruning would result in the most similar representations to the original model across layers.

Post-Pruning Re-initialization

The post-pruning re-initialization method makes copies of each magnitude based pruned model, reinitializes the all variables, but keeps the mask from the magnitude based method. In this way the topology of the pruned network is preserved and the network is trained from scratch for the same number of epochs as the baseline model. The post-pruning re-initialization method is specifically designed to find out: 1) how much capacity is needed for training; 2) if restricting weights to a learned topology will result in more similar representations to the original; and 3) if this topology uncovered by pruning encodes important information about the original model and helps improve models trained from scratch.

Random Sparse Initialization

To measure the efficacy of the post-pruning re-initialization method, we compare it with a model initialized to the same sparsity where the pruned weights are chosen at random. For each layer in the original model, we randomly prune an index from its weights until the desired sparsity is reached. We then allow those models to train for the same number of epochs as the baseline model. Random sparse initialization sheds lights on how models can be taught when their capacity is reduced in perhaps the least advantageous way possible. The random selection of weights ensures that it cannot take advantage of any architectural structure advantage, and the process may potentially remove important components.

Experimental Results

This section presents a series of experiments that are specifically designed to answer the following key questions: 1) Is pruning truly a compression process? 2) Does pruning work for untrained neural networks? 3) Do learned sparse structures carry important information? For each question, we restate the hypothesis, explain the detailed experiments, discuss the results, and finally draw our conclusions.

Is Unstructured Pruning Compression?

The conventional wisdom believes that pruning is merely a compression process, in which the redundant information is removed and the key network structure is preserved. In fact, almost all existing literature refer pruning as a compress technique for neural networks. The recently published "Lottery Ticket Hypothesis" [47] claimed that successfully trained large networks contain wining tickets from the beginning. The winning tickets refer to the sub-networks that have connections and initial weights that make training particularly effective. Therefore, the pruning process is just a lucky draw that helps to find the wining ticket. If pruning is merely a compression process, a neural network should learn none or minimal new knowledge while being pruned. As a result, the learned representations among all layers and models should have very high similarities (e.g. close to 100%). However, as the authors themselves noted in [47], the discovered lottery ticket winning sub-nets have weights that change the most during the retraining process.

To verify if pruning is truly compression or not, we design and conduct an experiment as follows. We train the original ResNet and VGG models using CIFAR-10 and CIFAR-100 datasets to serve as the baseline. The representations of each layer are recorded and analyzed using the SVCCA tool. Once the baseline model has been trained, we prune it in two ways: 1) iteratively using the magnitude based

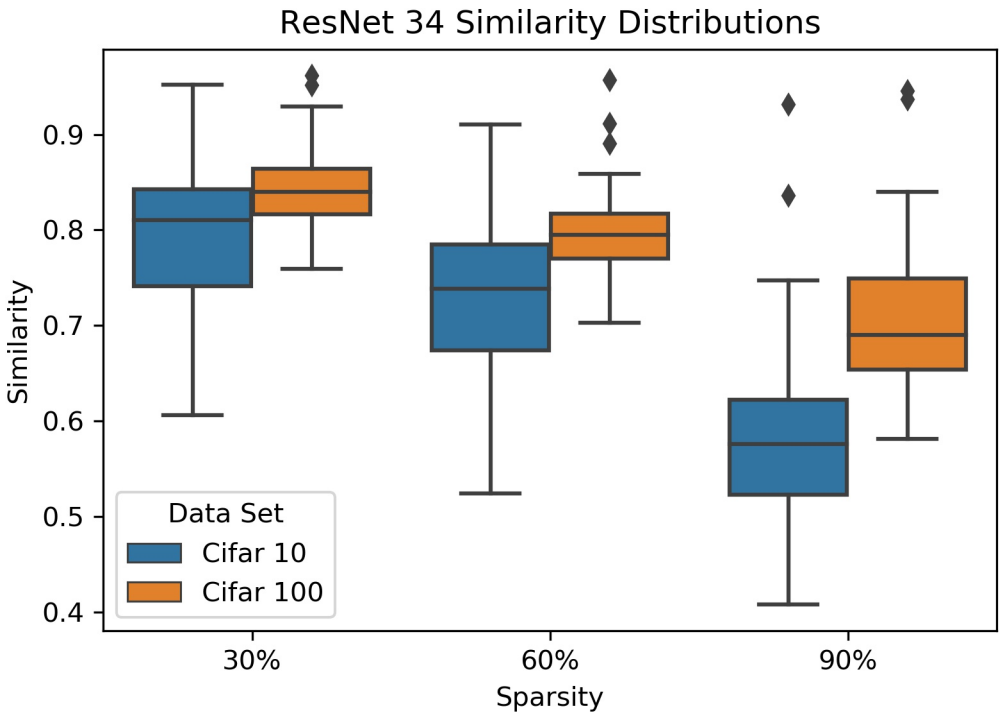


Figure 2.2: Similarity distributions for Pruned Magnitude ResNet-34 as compared to the original model.

pruning method, and 2) at incremental sparsity levels of 30%, 45%, 60%, 75% and 90% using magnitude based pruning, post-pruning re-initialization, and random sparse re-initialization. This allows us to observe how models react from moderate pruning towards intensive pruning where they could no longer maintain the desired accuracy. When all pruned models have finished training, we evaluate their accuracy, record their representations, and analyze the similarities using the SVCCA tool. We construct the input function in a way that guarantees each model to see the exact same images in the same order, which is critical for preserving the consistency of learned neuron vectors. Lastly, we calculate the SVCCA similarity value $\bar{\rho}$ for each layer and use it as the metric to fairly compare the similarities of different representations.

Figure 2.2 shows the SVCCA similarity value distributions for iteratively pruned

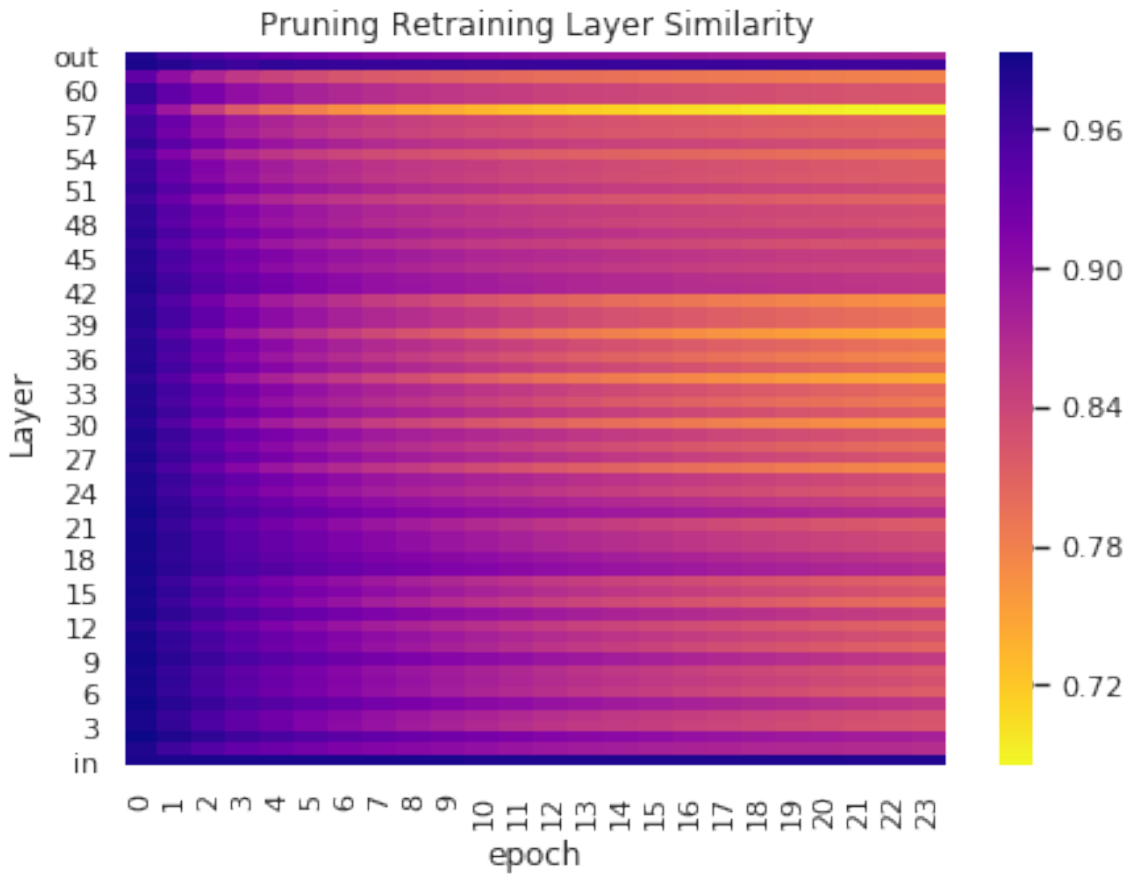


Figure 2.3: Similarity of 90% sparse ResNet during retraining. Similarities are calculated with respect to the initial state of the model before retraining (note: additional parts of the computation graph like batch normalization and skip layer additions are also calculated resulting in a layer count greater than 50).

magnitude ResNet-34 as compared to their original models. We can observe that the learned representations of moderate pruning (*e.g.* 30% of sparsity) remain relatively similar with the original model. However, the more aggressively (*i.e.* the higher the sparsity) a model is pruned, the less similar its learned representations are to the original model it is derived from.

We also investigate in what ways the models are changing while they undergo pruning. Figure 2.3 shows the similarities using a heatmap for the 90% sparsity model's layers in different epochs while retraining ResNet. This allows us to obtain visual insights about the evolutionary process the model undergoes to regain its

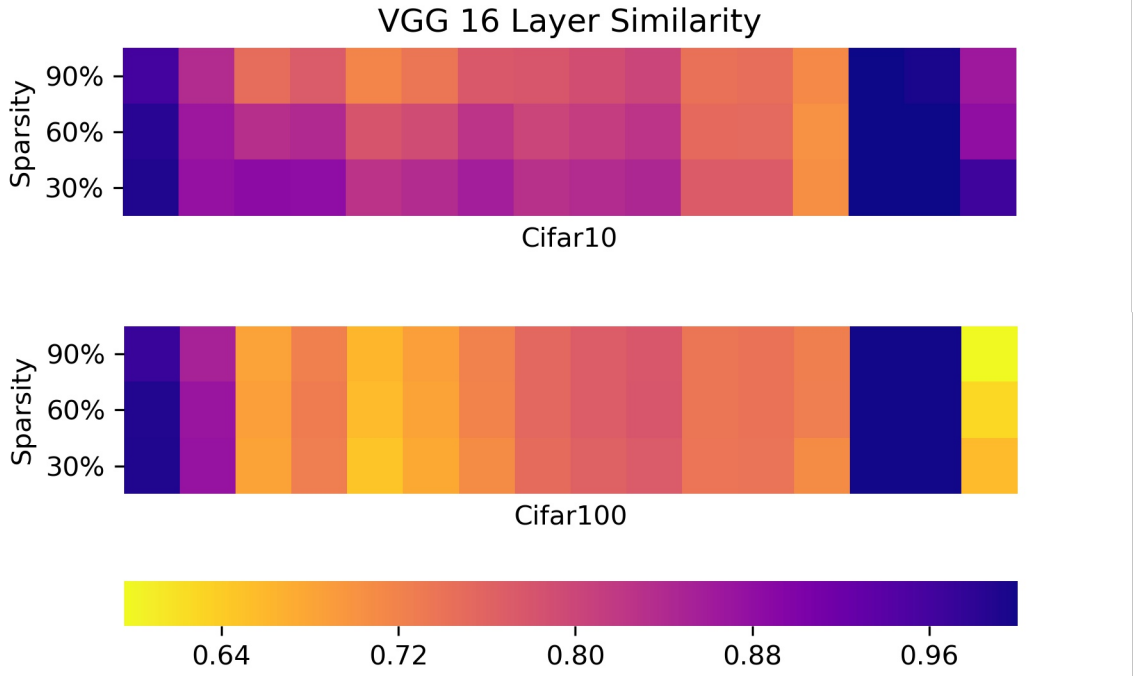


Figure 2.4: Similarities of VGG layer by layer at different sparsities resulting from iterative pruning. Layers go from left to right with the leftmost squares being the first convolutional layer and the last three being the fully connected and softmax layers.

accuracy. We observe that the filters that are closer to the output, where presumably high-level class features are located, change at the fastest pace. The high-level layers reach a steady state in the earliest time and propagate backward to the earlier layers in the model. This is different from the observations reported in [41] for models that are training from scratch, where the lower level layers converge to their final representations first. We believe our results complement [41] well with Google's results by showing how pruned models must repair their high-level representations first during the retraining process. In addition, an interesting stripping pattern is observed in the similarities, which is the result from the residual block of ResNet. While further investigation is needed to understand the phenomena, we speculate it is easier for the model to discard the information between the skip connections than to repair the

damage to the layers caused by pruning. Figure 2.4 contains a similar visualization for layer similarities of VGG at different sparsities compared to its original model. Figure 2.4 shows that the first two convolution layers maintain high similarities in both data sets similar to ResNet's first few layers. The CIFAR-10 VGG model has a pattern of propagating change similar to the ResNet CIFAR-10 model, while the CIFAR-100 model has noticeably more disruption in the middle layers at all sparsity values.

These results clearly demonstrate that pruning does not always result in the same representation. Actually, the aggressive pruning seems to force the network to adapt to the dramatically changed environment. As a result, the learned representations evolve to a new form, probably a more effective one than the originally learned representation. These observations are controversial to the traditional compression theory which treats pruning as a passive process without much of new learning.

Does Pruning Work for Untrained Neural Networks?

It is the common belief that a complex model should be trained with full capacity first before it can be pruned for better efficiency and initialized sparsity in an untrained neural network is unlikely to succeed. To test this hypothesis, we conduct an experiment with sparsly intialized models to see if pruning before training is a viable option, and if so how much model capacity is needed for training. In this experiment, we use the post-pruning re-initialization method. First a baseline mode is trained using TensorFlow and either the CIFAR-10, or CIFAR-100 dataset. Once the baseline model is finished training, the model is pruned to incremental sparsity levels of 30%, 45%, 60%, 75% and 90%. We take those pruned models and reinitialize all of their values, but leave their sparse structures, and let them train for the same number of epochs as the baseline model. Table 2.1 and 2.2 show the results of the post-pruning re-initialization accuracy as it compares to the original model, from which we can observe that full capacity is not always needed to train a model from

Table 2.1: Accuracy of single pass Pruned Models on CIFAR-10

Accuracy							
Sparsity		Ref (0%)	30%	45%	60%	75%	90%
ResNet-34	Mag.	92.6%	92.4%	92.5%	91.9%	91.2%	87.2%
	Re-Init.	92.6%	92.1%	92.0%	90.8%	89.9%	88.4%
	Rand.	92.6%	92.4%	91.8%	91.1%	90.2%	87.2%
VGG-16	Mag.	88.7%	88.7%	88.6%	88.2%	87.4%	10.0%
	Re-Init.	88.7%	87.3%	86.9%	10.0%	10.0%	10.0%
	Rand.	88.7%	87.0%	87.0%	10.0%	10.0%	10.0%
ConvNet	Mag.	86.3%	86.1%	85.8	86.0%	86.0%	83.6%
	Re-Init.	86.3%	85.7%	85.6%	84.6%	84.3%	81.0%
	Rand.	86.3%	86.0%	86.2%	85.2%	84.1%	80.1%

Table 2.2: Accuracy of single pass Pruned Models on CIFAR-100

Accuracy							
Sparsity		Ref (0%)	30%	45%	60%	75%	90%
ResNet-34	Mag.	68.8%	68.2%	68.5%	67.6%	65.8%	54.4%
	Re-Init.	68.8%	68.3%	67.5%	66.1%	65.4%	60.8%
	Rand.	68.8%	68.2%	67.2%	66.4%	64.6%	60.5%
VGG-16	Mag.	56.7%	57.5%	57.25%	56.4%	50.1%	1.1%
	Re-Init.	56.7%	53.1%	56.6%	1.0%	1.0%	1.0%
	Rand.	56.7%	57.4%	56.2%	1.0%	1.0%	1.0%
ConvNet	Magn.	58.5%	58.5%	58.3	57.6%	58.1%	53.4%
	Re-Init.	58.5%	57.9%	57.9%	58.0%	58.4%	58.4%
	Rand.	58.5%	58.3%	58.0%	58.8%	58.0%	58.%

scratch. For all but the most extreme level of pruning (e.g. 90% sparsity), the models can be trained to very similar accuracy as the baseline model. Furthermore, as demonstrated in Table 2.1, the sparsely initialized models perform equally well and sometimes even better than the original magnitude based pruning strategy for ResNet and ConvNet models.

VGG, however, is an exception. The accuracy of VGG (with initialized sparsity pruning) drops significantly when the sparsity goes beyond 45% (e.g. 10% for CIFAR-10 and 1% for CIFAR-100) and even the magnitude based method drops to 1.1% at 90% sparsity (see Tables 2.1 and 2.2). This indicates that sparsities over 45% prevents the VGG model from learning. The recorded test accuracies become only as good as random guessing for both datasets. We tried to adjust the learning rates and reduce or remove the drop out layers, but none of these changes helped the model to learn anything meaningful.

This is interesting because the ConvNet is similar in design philosophy to the VGG model. It has 2 convolutional layers, 2 fully connected layers, and a softmax layer. If redundancy of a model is measured only in number of weights or layers, then the VGG model should be able to survive from more aggressive pruning and still learn. These results agree with the observation by [48] that redundency in VGG is not evenly distributed. We believe ResNet is able to survive these catastrophic collapses during training with initialized sparsity because of its skip connections in the residual blocks. If a single convolution layer is broken or problematic, its deep network is capable of bypassing the layer.

Comparing the accuracies of the iterative pruning methods in Table 2.3 to that of the single pass magnitude pruning and pre-initialized sparsity methods in Tables 2.1 and 2.2, we observe that the real advantage iterative methods have is increasingly better initialization for the model weights and substantially larger training budgets. The original VGG paper [49] explicitly explained how sensitive VGG is to initialization, going as far as to train a smaller model to warm start the larger VGG-16 and VGG-19. This may also explain why iterative and single pass magnitude pruning are able to succeed at higher sparsities while the initialized sparsity methods can not.

These results overturn the conventional wisdom that models must be first trained

in full capacity before they undergo pruning. When attempting to design efficient models, new methodologies thus can be developed to speed up the pruning/training process by training models with some amount of sparsity from the beginning.

Table 2.3: Accuracy of Iteratively Pruned Models

Accuracy						
Sparsity		Ref (0%)	60%	70%	80%	90%
CIFAR-10	ResNet-34	92.6%	92.9%	93.0%	92.6%	91.7%
	VGG-16	88.7%	89.2%	89.9%	88.5%	87.9%
CIFAR-100	ResNet-34	68.8%	69.3%	69.3%	68.5%	60.7%
	VGG-16	56.7%	63.0%	63.4%	63.7%	63.4%

Do Learned Sparse Structures Carry Important Information?

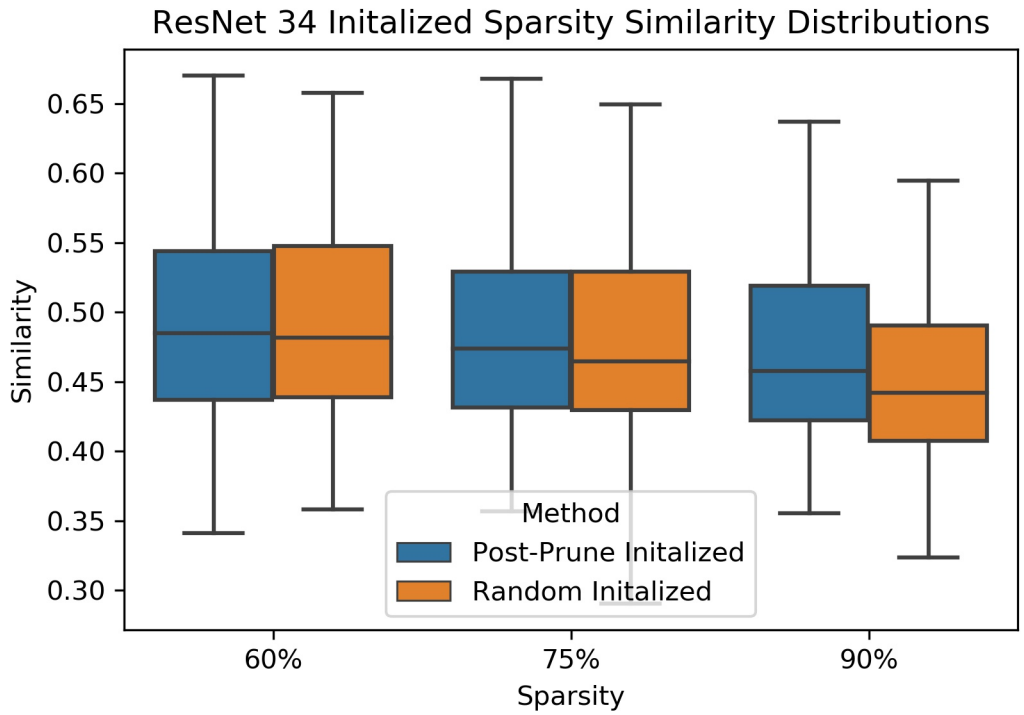


Figure 2.5: Similarity distributions for Post-Pruning Re-initialization and Random Sparse Initial ResNet-34 as compared to the original model.

Sparse structures resulting from structured pruning as in [48] has been demonstrated to have significant value. It is less clear what if any value network topology that is a result of unstructured pruning has. We design an experiment in this subsection to verify if the learned structure of the sparsity is important and if random sparsity can yield good accuracy.

In this experiment, we take the baseline model and randomly select weights in each layer to prune until the desired sparsity is reached. We prune to the same sparsities described in the previous experiment, and allow the model to train for the same number of epochs as our baseline model. Tables 2.1 and 2.2 compares the results of the learned pre-pruning method to the random pre-pruning method. In contrast to [48]'s findings for structured sparsity, there is virtually no distinction in performance between our random sparse initalized and post-pruning re-initialization methods. Our random initialized sparsity method does nearly as well or better than the post-pruning initialized method at all sparsities. ResNet-34 model preforms only 1% less in accuracy compared with the time-consuming magnitude based pruning method at the different sparsity levels. It also performs as well as magnitude based pruning methods at the 90% sparsity level for CIFAR-10 and outperforms magnitude based pruning for CIFAR-100. We also show that models that have prior knowledge of good sparse structures do not necessarily preform better. The only requirement is not to be over-aggressive for capacity (e.g. use 90% or higher sparsity). As long as moderate amounts of weights are available, the model will find a way to adapt and overcome its lower capacity and still learn. This indicates that not only can pre-pruning be effective, but it can be easily implemented using randomness.

We have already demonstrated that post-pruned re-initialized models do not have any significant difference in performance to randomly intialized sparse models. Purhaps as [47] suggest the data intensive process of unstructured pruning will create a sparsity structure that contains a certain bias. To further investigate if the learned

sparsity structures carry valuable information, we analyze the SVCCA similarity value distributions for the post-pruning initialized and random initialized ResNet-34 models compared with the original ResNet-34 model (see Figure 2.5). It can be observed that the median similarity values of the representations are less than 50% at all sparsities and the distinction between the randomly initialized and post-prune initialized methods is negligible. We also observe that as the level of sparsity increases, the similarities of post-prune initialization actually decreased (it should increase if the sparse structures really carry important information), which indicates that the learned sparse structures do not carry valuable information.

Conclusion

As an effective technique to reduce the size of modern neural networks, pruning has been extensively studied in recent years. However, the current understanding about pruning is still in its early stage with numerous misconceptions and inappropriate hypotheses. This chapter explores several fundamental questions about pruning and strives to find out 1) if pruning is truly compression; 2) if pruning can work on untrained neural networks; and 3) if sparsity structures from unstructured pruning provide valuable information. The following conclusions can be drawn from our experiments and observations.

First, after analyzing the similarities of various learned representations using the SVCCA tool, we find that pruning is not a passive compression process without learning new knowledge. Rather, the pruned model is capable of evolving proactively to survive in a dramatically changed environment, which is done by learning and transforming to more effective representations when aggressive pruning is occurring.

Second, our results indicate that initialized sparsity can work for untrained neural networks with certain architectures or capacities. Taking advanced knowledge from one model (e.g. which weights are important) and using it to train another

model will not result in any significant performance gains when training.

Third, we observe that the sparsity structure from the post-pruning re-initialization is not inherently useful or meaningful. Random initialized sparse models perform equally well compared to post-pruning re-initialized sparse models. The similarity of their layer representations share almost no connection or valuable information to the original models where they derive from. This is in contrast to structured pruning where the discovered structures can be used to train efficient models from scratch.

III. BIAS MITIGATION WITH DISTILLATION

Introduction

Deep neural networks (DNNs) are widely deployed across many domains but they are incredibly compute- and memory-intensive. To solve this problem, compression techniques such as pruning and quantization have been widely used to reduce the model size, computation demand, and energy consumption of complex DNNs without compromising top-line metrics (e.g. top-1 and top-5 accuracy). Nevertheless, recent literature has exposed the prevalence of undesirable biases in compressed neural networks[50, 51]. For example, Hooker et al. found that pruned networks often sacrifice the accuracy of a subset of the classes to preserve overall accuracy in classification tasks [50]. This exacerbates existing issues of algorithmic bias already observed in DNNs like CV models working better for people with lighter skin [52]. Unfortunately, these algorithmic biases generated by AI are more abstract and unintuitive, which makes it harder to explain, evaluate, and mitigate. Despite the rising concerns in bias and fairness of AI, research on how to evaluate and mitigate bias is still in its infancy.

In this paper, we strive to tackle the challenging issues of evaluating, mitigating, and explaining bias caused by pruning neural networks. Specifically, we make the following three contributions.

- **Bias evaluation**: We propose two simple yet effective metrics, Combined Error Variance (CEV) and Symmetric Distance Error (SDE), to quantitatively evaluate the bias prevention quality of pruned neural networks. Compared to existing metric such as Pruning Identified Exemplars (PIEs), CEV and SDE have clear advantages. First, they do not require training large populations of models. Two models can be compared directly. Second, SDE and CEV can

evaluate any compression methods (like PIEs or CIEs) but they are more objective than PIEs or CIEs by providing clear measurement of model quality in bias prevention.

- **Bias mitigation**: We demonstrate that knowledge distillation can effectively mitigate bias in pruned neural networks, even with unbalanced datasets. We apply state-of-the-art knowledge distillation algorithms on network pruning and use our proposed CEV and SDE metrics to evaluate their model bias. Our results clearly show that knowledge distillation provides a viable solution for mitigating bias in pruned neural networks. We also investigate the impact of unbalanced dataset on model bias. Although unbalanced datasets can amplify bias issues in pruned networks, our experiments confirm that knowledge distillation remain effective in mitigating bias in pruned models with unbalanced datasets.

- **Bias explanation**: We reveal that model similarity has strong correlations with compression induced bias, which provides a powerful method to explain why bias occurs in pruned neural networks. We utilize Singular Vector Canonical Correlation Analysis (SVCCA) [41] to compare similarity of layer representations between the original (non-pruned) and pruned models. Our experimental results unanimously show that pruned models that yield high similarity to original models induce less bias.

Evaluation Metrics for Model Bias

To evaluate the severity of bias in different AI models, it is essential to define fair and effective metrics. Recently, Google has conducted important work by exposing the prevalence of undesirable biases in compressed neural networks [51] and proposed Pruning Identified Exemplars (PIEs) [50] as a metric for measuring compression

induced bias in pruned neural networks. In this section, we discuss PIEs and its limitations. Meanwhile, we propose two new metrics, Combined Error Variance (CEV) and Symmetric Distance Error (SDE), and demonstrate their effectiveness by comparing to PIEs.

Pruning Identified Exemplars

(a) Shared PIE

(b) PIE without distillation

Figure 3.1: PIEs may represent noisy or confusing data (e.g. image (a) is hard for humans to classify).

PIEs are defined as the following equation, which represents images in a dataset for which pruning explicitly changes the behavior of the answer. In [50], they trained a population of 30 models for each compression method and sparsity level. They then defined an image i as an exemplar if modal label $Y^M_{i,t}$, or class most predicted by the t-compressed model population disagrees with the label produced from the original unpruned networks.

$$PIE_{i,t} = \begin{cases} 1, & \text{if } y^M_{i,0} \neq y^M_{i,t} \\ 0, & \text{otherwise} \end{cases}$$

While PIEs help reveal the bias issues in pruned models, not every image reported as a PIE represents a problem. A good portion of PIEs represent images that are equally hard for human to classify and may simply be a case where the

uncompressed networks overfit to learn the example. Figure 3.1(a) is a PIE image that is found both in pruned models with and without distillation. This image doesn't have a clear subject and is hard for human to label as well. It is probably not proper to say that miss labeling this example is a poor model. However, Figure 3.1(b) is a very different case. The subject is in the center of the image, is not obstructed, and contains no other objects that might be labeled. Missing labeling it indicates low quality of pruned models.

While PIEs are an important step towards identifying and detecting bias in pruned models, they have clear limitations. First, finding PIEs requires training a population of models for which to compare a modal answer. Therefore, it is not practical to directly compare the quality of a pruned model to the original model. Second, as PIEs only compare examples labeled by two populations (or original model), they cannot provide insights about distribution variance or shift, which are critical to evaluate model bias.

To address the weaknesses of PIEs, we propose two new metrics, Combined Error Variance (CEV) and Symmetric Distance Error (SDE). Pruning and quantization have been observed to sacrifice accuracy on a subset of class in classification tasks in order to retain overall top-k accuracy [50]. To catch and reflect this bias, our metrics are designed to quantify both the *spread* of the change in classification error as well as changes in *how* the model is making mistakes. As a result, both of our proposed metrics consider the distribution of change in false positive and false negative rates (FPR, FNR) for all classes.

Combined Error Variance

The principle of the Combined Error Variance (CEV) metric is to discourage the bias of sacrificing one class for the benefit of another class. Therefore, CEV calculates the variance of the FNR/FPR changes and penalizes changes in FNR/FPR away from

the model's average change. Mathematically, CEV is defined and calculated as follows.

Let X_i be a tuple of the FPR and FNR for class i of the compressed model and $\hat{X}_i$ be the original models FPR/FNR tuple. We first find a the normalized change in FPR/FNR δX_i and the mean normalized change δX_μ.

$$\delta X_\mu = \frac{1}{n}\sum_{i=0}^{n}(\delta X_i) \qquad \text{(III.2)}$$

$$\delta X_i = \frac{X_i - \hat{X}_i}{\hat{X}_i} \times 100 \qquad \text{(III.1)}$$

The variance of the change in FPR and FNR is calculated for all n classes as follows.

$$cev = \frac{1}{n}\sum_{i=0}^{n}(\delta X_\mu - \delta X_i)^2 \qquad \text{(III.3)}$$

Symmetric Distance Error

The principle of the Symmetric Distance Error (SDE) metric is to discourage another undesirable bias behavior in pruned models. That is, a class with more training examples or that has similar features to another class is more frequently to be chosen by the model as its capacity degrades. Conversely, classes with fewer training examples are more likely to be cannibalized by the model and guessed less frequently. To reflect this bias behavior, SDE calculates "how far away" from balanced is the change in FPR/FNR for a single class error. More intuitively, if we make a scatter plot with changes in FPR and FNR as X and Y values, the diagonal line in that plot would be a perfectly balanced change in FPR/FNR. Therefore, the SDE can be calculated as the symmetric distance of each change to that balance line.

Mathematically, it can be proven that for a line given by the equations $ax + by + c = 0$, the distance d from any point (x_0, y_0) can be derived from the

following equation:

$$d = \frac{|a(x_0) + b(y_0)|}{\sqrt{a^2 + b^2}} \tag{III.4}$$

In our specific context, the diagonal of the Cartesian plane (i.e. the balance line) is $x = y$ or $x - y = 0$. Therefore, given any change in the form of FNR (i.e. x) and FPR (i.e. y) coordinate, the symmetric distance of that change to the balance line can be calculated as:

$$d = \frac{|(1)(x_0) + (-1)(y_0)|}{\sqrt{(1)^2 + (-1)^2}} = \frac{|x_0 - y_0|}{\sqrt{2}} \tag{III.5}$$

Once the symmetric distance of each change is calculated, SDE of a pruned model can be calculated as the mean magnitude of normalized FP/FN rate difference divided by $\sqrt{2}$.

$$sde = \frac{1}{n} \sum_{i=0}^{n} \frac{|\delta FNR_i - \delta FPR_i|}{\sqrt{2}} \tag{III.6}$$

Comparison between CEV/SDE and PIEs

To demonstrate the effectiveness of CEV and SDE for evaluating model bias in pruned models, we outline the following experiments and compare with the PIEs metric.

Table 3.1: Knowledge distillation methods and corresponding abbreviations

Distillation Method	Proposed By	Abbreviation
Knowledge Distillation	Hinton et al. [17]	KD
Attention Transfer	Zagorukyo et al. [53]	AT
Similarity-Preserving Knowledge Distillation	Tung et al. [54]	SP
Flow of Solution Procedure	Yim et al. [55]	FSP
Probablistic Knowlege Transfer	Passalis et al. [56]	PKT
Contrastive Representation Distillation	Tian et al. [57]	CRD

Pruning Details

We train a CNN ResNet [58] (ResNet32x4) model on CIFAR100 [59] and prune it to various sparsities using Filter-wise Structured Pruning. We use pruning as the baseline model and the rest of models are pruned jointly with one of the state-of-the-art knowledge distillation methods shown in Table 3.1. We utilize Tian et al's [57] implementation for all distillation methods tested. For all experiments the models are pruned initially to 10% sparsity then gradually pruned every 5 epochs until the desired sparsity is reached according to the AGP [60] schedule. All pruning is completed at the halfway point of training and allowed to continue to finetune for another 120 epochs. The chosen sparsity levels are 30%, 45%, 60%, 75%. Each layer in the model is pruned to the same sparsity. We do not perform any layer sensitivity analysis or prune layers at different ratios. Although that may have resulted in higher accuracy, our goal is not to reach state-of-the-art compression ratios but study the effect of pruning and how to mitigate bias.

Results

Figure 3.2 plots the experimental results on accuracy and model bias of each pruned model evaluated using the PIE metric and our CEV/SDE metrics. We can observe from Figure 3.2 (a) that several methods reach nearly identical accuracy (e.g. KD, SP, and FSP at 45% sparsity). Figure 3.2 (b) shows the evaluation results of model bias using the PIEs metric, from which we can see that the ranking of three models with similary accuracy would be SP $>$ KD $>$ FSP (i.e. less PIEs indicates better models). However, the ranking will change dramatically to KD $>$ FSP $>$ SP (i.e. small CEV/SDE indicates better models) when evaluating using CEV (ref. Figure 3.2 (c)) and SDE (ref. Figure 3.2 (d)) respectively. To find out which metrics better reflect the bias issues in pruned models, we plot the actual distribution of

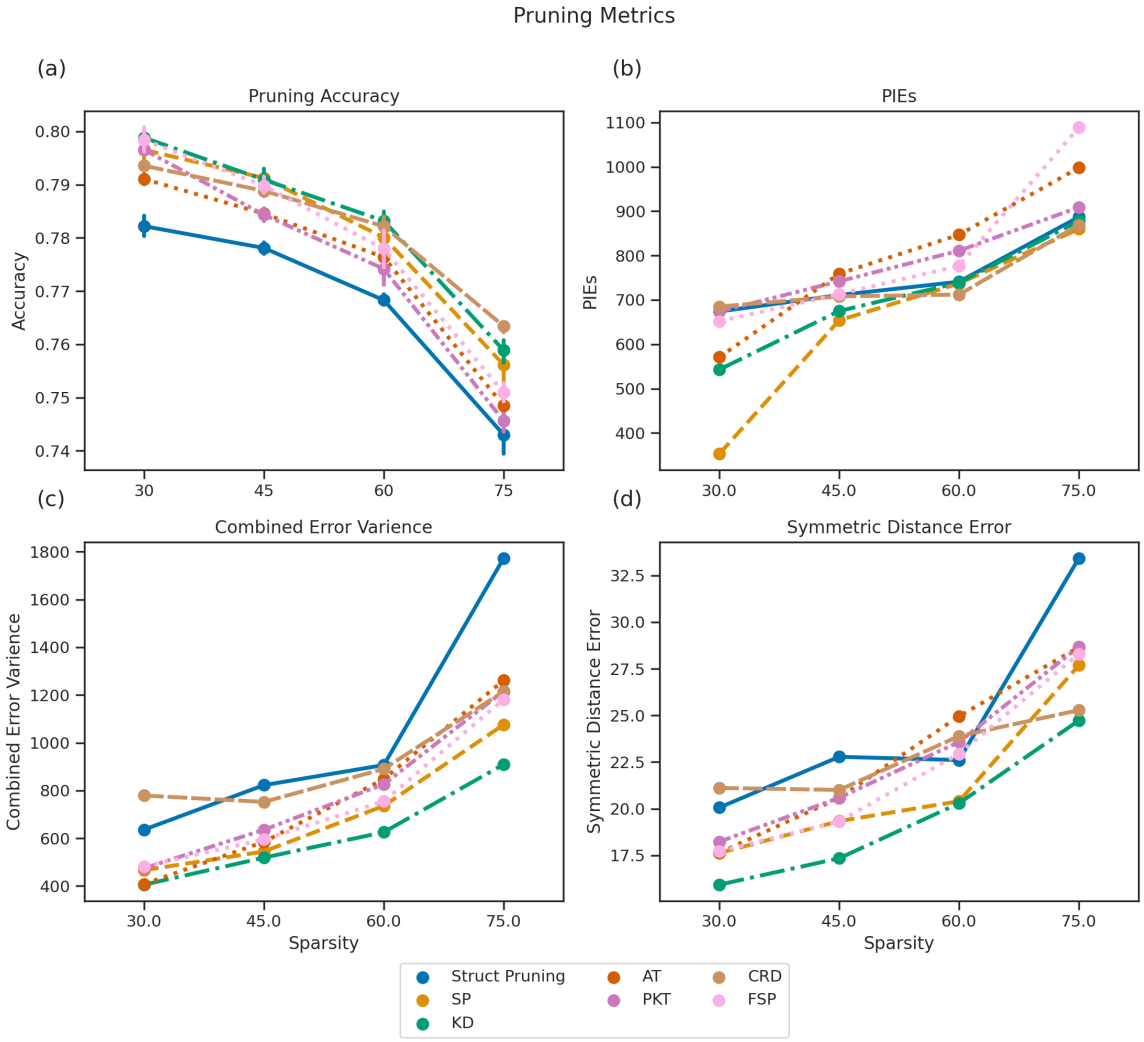
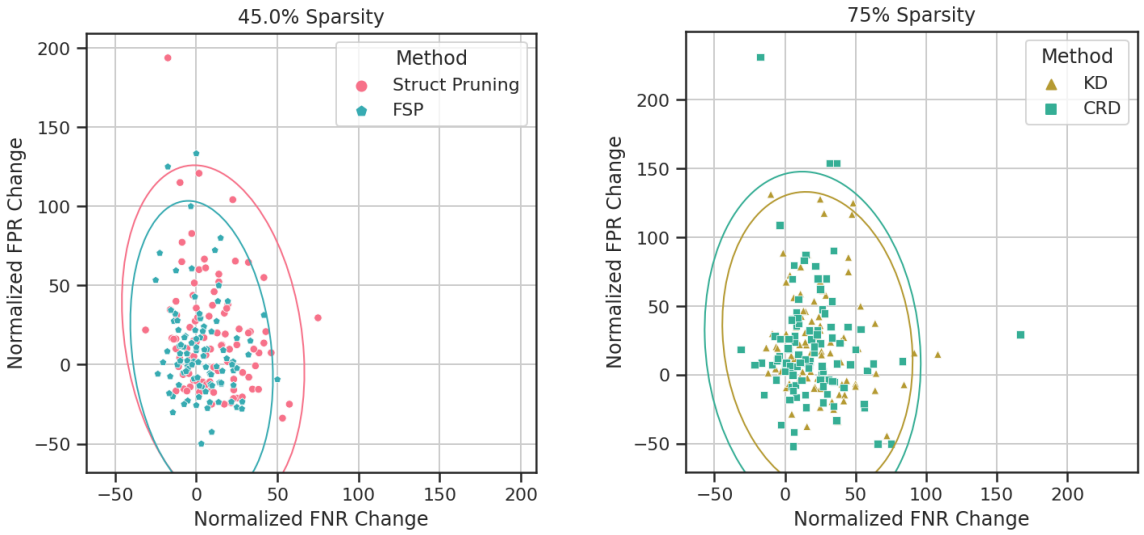


Figure 3.2: Accuracy, PIEs, CEV, and SDE results and comparison

FPR/FNR change of the baseline pruned model and FSP in 3.3 (a). We can immediately see that the ellipse of FSP is much smaller than the ellipse of SP. Since ellipses contain 95% of data points, it clearly shows the error distribution of FSP is certainly better than SP (i.e. less biased), although FSP has a higher number of PIEs.

An even more exaggerated case is the comparison of the CRD and KD models. CRD has a higher accuracy by nearly half a percentage point at 75% sparsity (ref. Figure 3.2 (a)) and the number of PIEs for the two models at this sparsity is nearly identical (ref. Figure 3.2 (b)). CRD seems to be a better model. However, when



(a) Pruning vs. FSP

(b) KD vs. CRD

Figure 3.3: Scatter plot of normalized false positive and false negatives rate (FPR/FNR) change for different distillation methods on CIFAR100. Each point represents a single class's normalized change. Ellipses contain 95% of data points.

plotting the error distribution of KD and CRD at 75% sparsity (ref. Figure 3.3 (b)), we can clearly observe that KD has better error distribution for the majority of classes and its outliers are much less pronounced than the CRD distilled models. These important characteristics are missed by PIEs but correctly reflected by our CEV/SDE metrics (ref. Figures 3.2 (c) and (d)). These two examples provide strong evidence that our metrics are more descriptive than the PIEs metric in evaluating model bias.

Mitigating Bias with Knowledge Distillation

From previous experiments, we notice that all pruned models with distillation achieves higher accuracy at every sparsity than the model pruned without distillation (ref. Figure 3.2 (a)). Surprisingly, we also notice that knowledge distillation can effectively mitigate pruning induced bias by tightening the distribution between the normalized change in FPR and FNR (ref. Figure 3.3 (a) as an example). It appears that distilled pruned models have tighter distribution of classes overall and their

extreme outliers are less pronounced.

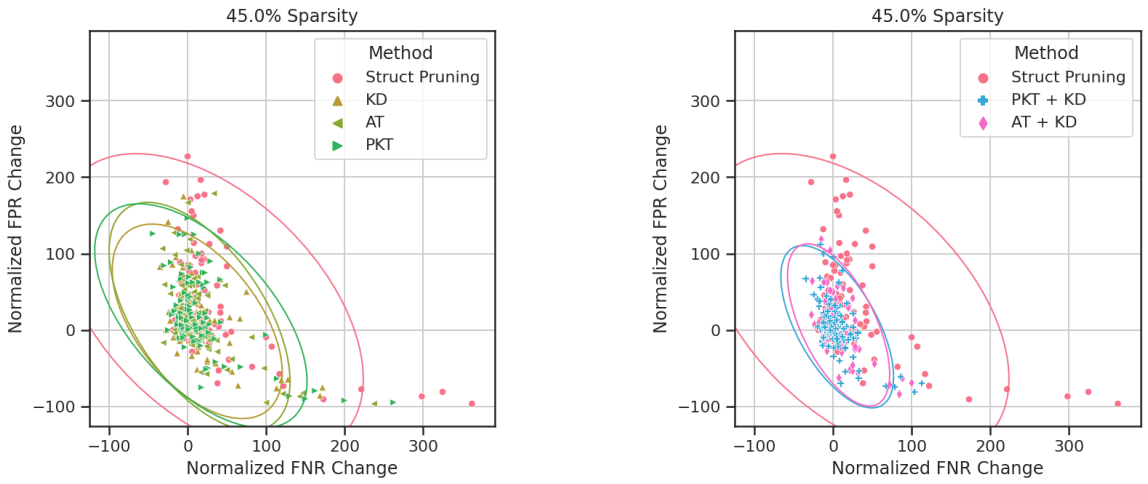
Data Induced Bias

Table 3.2: CIFAR100 class sample rate and mean class top-1 accuracy for original dataset (Org) and down-sampled dataset (Bias) pruned at 45% sparsity.

	Method	Struct Pruning		KD		FSP		FSP + KD		SP	
		Org	Bias	Org	Bias	Org	Bias	Org	Bias	Org	Bias
Class Name	Percentage Remain										
apple	10.00	88.74	66.99	91.24	79.24	90.74	72.99	91.49	78.74	91.49	75.49
bicycle	10.00	84.24	33.24	88.74	62.24	85.74	37.74	85.74	58.24	85.99	42.74
crocodile	10.00	92.49	63.99	92.49	79.49	92.74	71.99	92.24	79.49	92.49	71.24
maple tree	10.00	88.74	55.99	87.24	77.99	89.24	63.49	89.99	77.24	91.24	63.74
shrew	10.00	76.49	29.99	80.24	45.49	78.74	34.99	74.99	48.99	75.74	34.99
butterfly	20.00	83.24	59.74	84.24	71.24	82.99	68.49	85.24	69.49	83.99	65.74
can	20.00	89.74	80.99	88.49	81.24	89.74	80.49	89.24	82.74	90.24	81.24
oak tree	20.00	77.74	63.74	78.49	70.74	77.49	65.49	79.49	68.74	77.24	68.49
palm tree	20.00	88.24	71.99	90.24	79.49	90.49	77.99	90.49	80.24	90.24	77.49
train	20.00	95.49	84.99	94.74	90.24	94.24	86.49	95.99	87.24	94.99	86.24
dinosaur	50.00	92.74	89.74	92.74	90.49	92.74	88.99	91.74	89.99	92.99	90.24
elephant	50.00	92.24	86.99	89.74	89.74	93.49	90.74	91.99	89.49	91.74	89.74
flatfish	50.00	82.74	74.49	86.49	80.99	86.49	76.24	86.49	**82.24**	85.99	76.49
orange	50.00	81.49	70.49	82.49	75.99	78.24	72.74	81.74	75.99	82.49	74.24
raccoon	50.00	75.49	62.74	77.99	72.49	77.99	68.24	78.99	72.24	79.49	70.74

	Method	SP + KD		PKT		PKT + KD		AT		AT + KD	
		Org	Bias	Org	Bias	Org	Bias	Org	Bias	Org	Bias
Class Name	Percentage Remain										
apple	10.00	92.24	79.24	90.74	79.99	90.49	83.99	89.74	81.24	89.74	**84.99**
bicycle	10.00	88.24	60.74	85.74	49.99	88.24	**69.49**	87.24	59.74	86.99	68.99
crocodile	10.00	92.99	81.24	90.99	71.99	92.49	84.74	90.99	73.99	92.99	**86.24**
maple tree	10.00	91.24	75.74	88.49	65.99	88.74	**82.49**	87.49	72.99	88.99	82.49
shrew	10.00	76.99	48.99	79.49	41.24	80.74	54.74	73.99	43.49	81.24	**56.24**
butterfly	20.00	84.24	73.49	86.99	72.99	87.24	75.74	88.74	72.49	87.49	**77.24**
can	20.00	88.74	83.49	87.24	81.74	86.99	85.99	89.49	83.24	89.74	**87.24**
oak tree	20.00	78.24	70.74	77.24	69.24	78.24	71.49	75.74	69.49	77.24	**71.99**
palm tree	20.00	90.24	80.99	89.99	77.49	89.99	83.99	90.74	81.74	88.24	**84.99**
train	20.00	95.49	90.24	93.99	84.99	95.49	90.49	96.24	86.49	93.74	**90.74**
dinosaur	50.00	92.99	91.49	93.99	88.49	92.74	**92.74**	92.74	90.74	93.74	91.74
elephant	50.00	89.99	89.74	91.99	90.49	92.99	**91.99**	92.24	89.49	90.99	90.99
flatfish	50.00	85.99	79.24	84.49	76.99	84.24	80.49	81.49	76.99	82.49	76.74
orange	50.00	81.74	74.49	85.49	76.24	84.24	**77.99**	84.24	76.24	84.24	76.74
raccoon	50.00	77.74	74.24	79.24	72.49	78.24	**74.74**	75.24	68.24	78.99	73.74

While the results we have presented so far are encouraging, they are merely based on our experiments on CIFAR100 (a well balanced dataset), which is not



(a) Pruning vs Distillation

(b) AT/PKT + KD

Figure 3.4: Normalized FP/FN Rate Change for distillation methods on biased CIFAR100 dataset

necessarily reflective of datasets in the wild. Real world data is often unbalanced and could amplify the bias induced by pruning. Hooker et al. reported that the negative effects of compression are most observed on underrepresented groups [51]. Therefore, we design a series of experiments to verify if knowledge distillation can mitigate data induced bias in pruned networks. We resample the CIFAR100 dataset to purposely under represent certain classes in the training set. We select 15 classes and reduce the remaining training samples to 50%, 20%, and 10% of their original numbers. Table 3.2 shows the selected classes and their original Top-1 accuracy. These classes are selected according to which classes from all our previous pruning experiments have either the smallest change in FPR/FNR or have an increase in accuracy. Our hypothesis is that these biased classes with less data samples will more likely be picked as "victims" by the pruned models. We also test if combining our feature map based distillation methods with KD (e.g. AT + KD or PKT + KD) can achieve additional benefits.

From Table 3.2, we can see that pruning alone performs significantly worse than knowledge distillation on nearly all classes with removed samples (marked in red).

Noteably the "shrew" class went from 76.49% top-1 accuracy when pruned on the entire dataset to just 29.99% when reduced to just 10% of of its original training examples. While AT + KD reached 54.24% top-1 accuracy, an 88.09% improvement. In addition, the best outcomes (bolded) for down sampled classes all came from combining distillation methods that use internal feature maps with KD. Table 3.3 shows that Attention and PKT + KD reduced SDE in half at 45% sparsity and CEV to less than 25% of its value with pruning alone. Looking at the distribution of FPR/FNR change in Figure 3.4, we can see that the improvements are not limited to the 15 classes we intentionally down sampled. The models pruned normally have seen a substantial number of classes with significant shifts in both the false positive (FP) and false negatives (FN) direction. The pruned models over guess classes not down sampled and under guess those down sampled (i.e. bias is amplified). It is notable that the distributions of error change of PKT/AT + KD in Figure 3.4 (b) are clearly more desirable than that of pruning alone pruning, even though they have nearly identical top-1 accuracy. This reinforces the usefulness of CEV/SDE in evaluating model bias. We conclude that knowledge distillation methods, especially those that align both feature maps as well as the output layer, are incredibly effective at mitigating pruning induced bias, even with unbalanced datasets.

Explaining Model Bias Using Model Similarity

Previous studies [22, 61] have shown that pruned neural networks evolve to substantially different representations while striving to preserve overall accuracy. In Section 3, we have demonstrated that knowledge distillation can effectively mitigate both pruning and data induced bias in compressed networks. Inspired by both work, we will be able to explain the occurrence of bias in pruned networks if strong correlations between model similarity and model bias can be confirmed.

We utilize the Singular Vector Canonical Correlation Analysis (SVCCA) [41] to

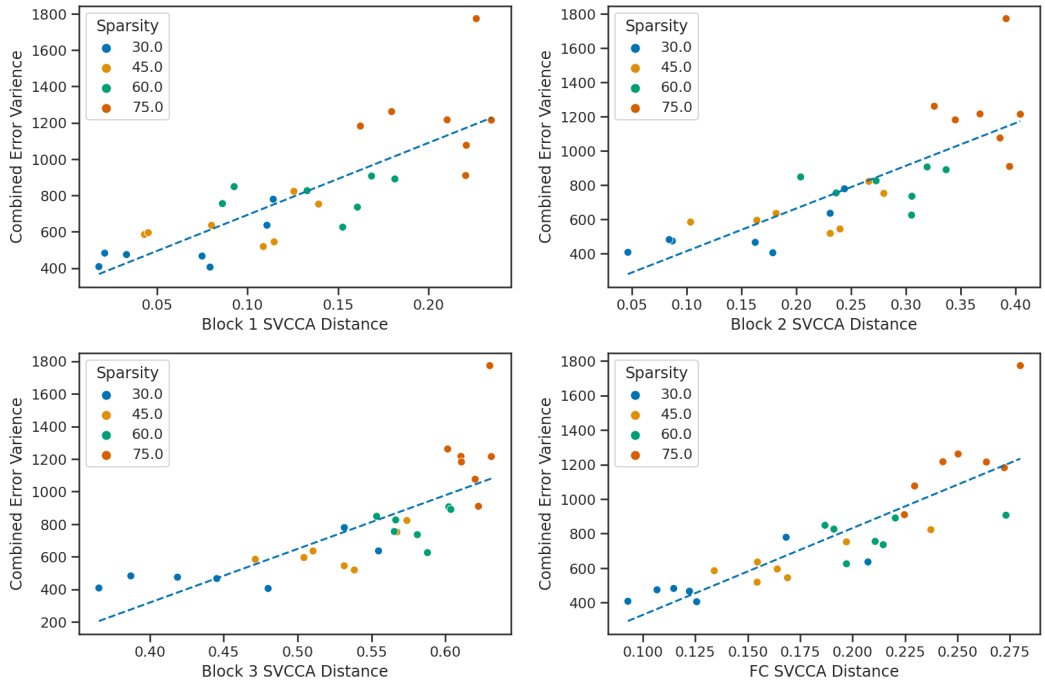


Figure 3.5: Scatter Plot of Combined Error Variance Plotted against SVCCA Distance at each layer output with regression line overlaid. Values are colored by network sparsity

Table 3.3: Average SVCCA distances at each block and CEV/SDE results on biased CIFAR100 dataset

# PIEs	Method	Block 1	Block 2	Block 3	FC	CEV	SDE	Accuracy
742	AT + KD	0.051	0.121	0.480	0.122	851.694	25.246	77.100
748	PKT + KD	0.071	0.164	0.495	0.128	903.329	25.103	77.335
768	SP + KD	0.107	0.231	0.538	0.161	1506.103	30.974	76.927
742	FSP + KD	0.063	0.187	0.514	0.156	1518.462	30.668	75.285
770	KD	0.106	0.230	0.540	0.160	1536.566	30.789	78.142
909	AT	0.044	0.104	0.475	0.142	1955.926	37.000	78.097
881	PKT	0.075	0.167	0.506	0.154	2190.690	38.121	78.963
838	SP	0.113	0.240	0.535	0.187	2654.847	41.378	78.520
877	FSP	0.045	0.162	0.508	0.171	2901.940	43.169	78.413
887	Struct Pruning	0.127	0.268	0.581	0.261	4237.391	51.146	77.242

study and contrast layer representations of pruned and original models. It combines Singular Vector Decomposition (SVD) to reduce the dimensionality of layer activations, and Canonical Correlation Analysis (CCA), to maximize a projection between two layers that results in the highest correlation. For a given dataset

$X = \{x_1, ..., x_m\}$ and a neuron i on layer l, the output of that neuron on the entire dataset is defined as the vector z_i^l. SVCCA takes input from two layers $l_1 = \{z_1^{l_1}, ..., z_{n_1}^{l_1}\}$ and $l_2 = \{z_1^{l_2}, ..., z_{n_2}^{l_2}\}$ where n_1 and n_2 are the number of neurons in their respective layers. SVD is performed on each layer to get new subspaces l_1' and l_2' where $l_1' \subset l_1$, $l_2' \subset l_2$. Next, l_1' and l_2' are linearly transformed to be as aligned as possible and correlation coefficients are calculated. SVCCA outputs aligned directions $(\tilde{z}_i^{l_1}, \tilde{z}_i^{l_2})$ and how well they correlate. The higher the correlation value ρ_i is, the more similar the two aligned directions are.

To calculate the similarity, we compare the feature maps and activations of the baseline model in our compressed populations. We use the same sets of features targeted by our various distillation methods. These activation output blocks in ResNet are the feature maps in the model where their input shapes change. In CIFAR100, the output shapes include 32x32, 16x16, 8x8. We also look at the similarity of the fully connected output layer before softmax, which is a tensor of size 100. In our experiments, we calculate the SVCCA distance $1 - \bar{\rho}$ where $\bar{\rho}$ is the mean of the ρ_i values from the top SVCCA similarity. A larger SVCCA distance indicates a lower similarity between two layers (or block of layers).

The calculated SVCCA distance and corresponding CEV values at different sparsities are shown in Figure 3.5, which are strongly correlated as evidenced by the regression line. In other words, a pruned model that yields higher similarity to the original model will have better CEV values (i.e. less induced bias). High similarity from the FC layer leading to better outcomes may intuitively be less surprising. After all, if two models with different configurations both reach the same answer, a small SVCCA distance is expected. However, we observe the strong correlations between similarity and CEV not only in the FC layer but ***all*** levels of model features.

If knowledge distillation shows *how* to reduce desperate impact of underrepresented classes when pruning, we believe network feature similarity explains

why it works. Regardless of sparsity level or pruning methods, we find the best predictor of CEV in our models is SVCCA layer distance. This holds true for both our experiments with blanced CIFAR100 and biased CIFAR100, as shown in Table 3.3. Additionally, we find that top-1 accuracy is not a great indicator of CEV. In fact, our models with the highest top-1 accuracy on the biased dataset are ranked near the bottom of CEV/SDE scores. We believe the strong correlation between similarity and CEV provides an important direction for future research in developing better pruning and distillation methods with mitigated bias.

Related Work

With the ubiquitous usage of AI, a substantial literature has recently emerged concerning algorithmic bias, discrimination, and fairness. Here we only discuss relevant studies that focused on addressing bias induced by data and algorithms. Mehrabi et al. conducted a survey on bias and fairness in machine learning [62]. Mitchell et al. explored how model cards can be used to provide details on model performance across cultural, racial, and intersectional groups and to inform when their usage is not well suited [63]. Gebru el al. proposed to use datasheets as a standard process for documenting datasets [64]. Amini,et al. proposed to mitigate algorithmic bias through resampling datasets by learning latent features of images [65]. Wang et al. designed a visual recognition benchmark for studying bias mitigation in visual recognition. [66]. Literature on network pruning has been historically focused on accuracy [67, 68] with recently work on robustness [69, 70]. Movva and Zhao [61] investigated the impact of pruning on layer similarities of NLP models using LinearCKA [71]. Ansuini et al. and Blakeney et al. also investigated how pruning can change representations using similarity based measures [72, 22]. Unfortunately, very few work have studied how pruning can induce bias and how to evaluate and mitigate the pruning induced bias. To the best of knowledge, our work is the first one to tackle

the evaluation, mitigation and explanation of pruning induced bias altogether.

Broader Impact

Compressed DNNs are widely used and our research can help alleviate increasing concerns in society regarding algorithmic bias in AI. Specifically, (1) we help the community better understand why bias occur in pruned models. (2) we present a viable solution to mitigate algorithmic bias induced by pruning. (3) CEV and SDE provide new metrics to evaluate the bias-prevention quality of pruned models. However, our research is currently limited to study and mitigate bias induced by pruning only. We believe there are many other types of biases exist in various AI models, which may not be addressed by this dissertation. Our metrics are not a replacement for carefully studying the behavior of models and ensuring that they meet other fairness standards. Therefore, we do not encourage policy makers or stakeholders use our metrics to establish a scoring system for ranking general AI models.

Limitations

Our work has three primary limitations. First, we only show extensive experimental results on a single model (ResNet) and dataset(CIFAR100). However, we do include limited eexperiments using VGG16 on CIFAR10, Imagenette, and Imagewoof in the appendix. Second, our model population size for each method/sparsity is much lower than the populations size (30 each) of Google's PIE paper [50]. While more results could increase our confidence, we believe our experiments are sufficient to draw conclusions from. Third, there is no human involvement in our method. The combination of automatic bias prevention techniques (like ours) and human-in-the-loop process (e.g. auditing) may further reduce biases in pruned neural networks.

Conclusion

In this chapter, we propose two new metrics, Combined Error Variance (CEV) and Symmetric Distance Error (SDE), to evaluate the pruning induced bias in compressed neural networks. We also demonstrate that knowledge distillation can effectively mitigate bias in pruned models, even with unbalanced datasets. Moreover, we reveal that model similarity has strong correlations with pruning induced bias, which can be used to explain why bias occurs in pruned neural networks.

IV. CRAFT DISTILLATION AND PARALLEL BLOCKWISE DISTILLATION

Introduction

This chapter discusses work from both [25], which describes how one can design a replacement for individual layers of blocks in neural networks and replace them after a model is trained to convergence (Craft Distillation), and from [26], which extends this and demonstrates how this can be broken into embarrassingly parallel jobs (Parallel Blockwise Distillation). Notably, this technique uses a feature distillation technique similar to AT [53] from the last chapter, which was found to preserve best the similarity and original biases' of the model. This allows for an alternative to structured pruning, which is also well-suited for compressing models without specialized hardware.

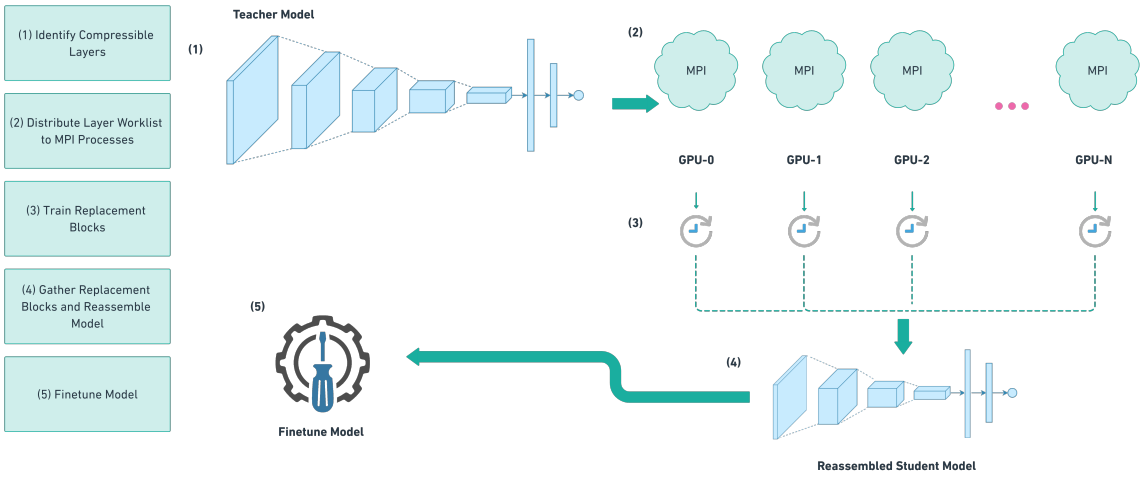


Figure 4.1: System level overview of parallel blockwise distillation

The two parts of this chapter jointly address two challenges with knowledge distillation. First, typically in knowledge distillation, one must select both a student

and a teacher [17]. However, this is not a trivial choice as it has been well documented that differences in architecture and "capacity gap" (loosely defined as differences in the number of parameters of flops) can cause inferior performance [73, 74]. This can result in significant wasted compute experimenting with training models to find an acceptable accuracy size trade-off curve. Craft Distillation solves this problem by transforming *any* teacher architecture by replacing computationally expensive or memory-intensive layers with more efficient ones. It can be seen as an extreme form of Progressive Block-wise Knowledge Distillation [75], allowing for more granular control over the compression process while not being constrained to a specific model architecture.

The second problem, which applies to knowledge distillation and other memory-intensive techniques in deep learning, is that scaling the method efficiently is challenging. Most models are trained using only data parallelism and require synchronization after the backwards pass. This scales well with single GPU and even single-node training but often scales poorly across multiple nodes. Parallel Blockwise Distillation provides the solution to the second issue by allowing distillation to scale to an arbitrary number of GPUS distributed to any number of nodes. It also reduces the total memory consumption of each training job by only loading the required portion of the model, reducing the training time of each job and energy consumption.

The combined steps are illustrated in Figure 4.1. First, it identifies all compressible layers in the teacher model and creates independent tasks for replacing them (one task for each layer). All tasks will be distributed to multiple GPUs via a group of TensorFlow instances and MPI processes following a specific scheduling algorithm such as round robin, bin packing, or work stealing. Each process trains its replacement blocks independently on the activations of the layers to be replaced. Once all layers have been trained, the main MPI process gathers the weights from all processes and reassembles the compressed DNN. Last but not least, the reassembled

student model will be fine-tuned to minimize the accuracy loss from the teacher model.

Our algorithm has the following advantages: (1) It requires relatively few epochs for each training layer and runs once (not iteratively), which reduces training time over traditional compression methods. (2) It utilizes task parallelism to increase the simultaneous execution of different tasks on multiple GPUs with very little communication, which minimizes synchronization overhead. (3) It transparently ensures elasticity and scalability because users do not need to adjust their hyperparameters when adding more GPUs. (4) It significantly reduces training time and total energy consumption, which is crucial for DNNs used in production software, where they are often deployed, evaluated, retrained, and redeployed in repeated development cycles.

Craft Distillation

In traditional knowledge distillation [17], a large “teacher model" is used to train a smaller “student model." In this phase, we present a different process of distilling a model in a layer-wise fashion which we call “craft distillation". Figure 4.2 illustrates the proposed craft distillation process, which selects a single layer of the original model at a time and teaches a block consisting of one or more depthwise separable layers (or other layer types that can reduce complexity). After selecting a convolution layer to replace, the input feature maps for the layer are used as the training data and act as the teacher. The output feature maps become the labels. The training is treated as a simple regression problem using L2 loss. Let us consider the case of replacing layer l in our model. Let f_l be the function that maps an input image to the activations at layer l. Let g be the replacement block for layer l. Our loss function can be represented as $L = \frac{1}{N}\sum_{i=1}^{N} ||f_l(x_i) - g \circ f_{l-1}(x_i)||^2$. When the replacement block has reached convergence, it is inserted into the original model in place of the selected

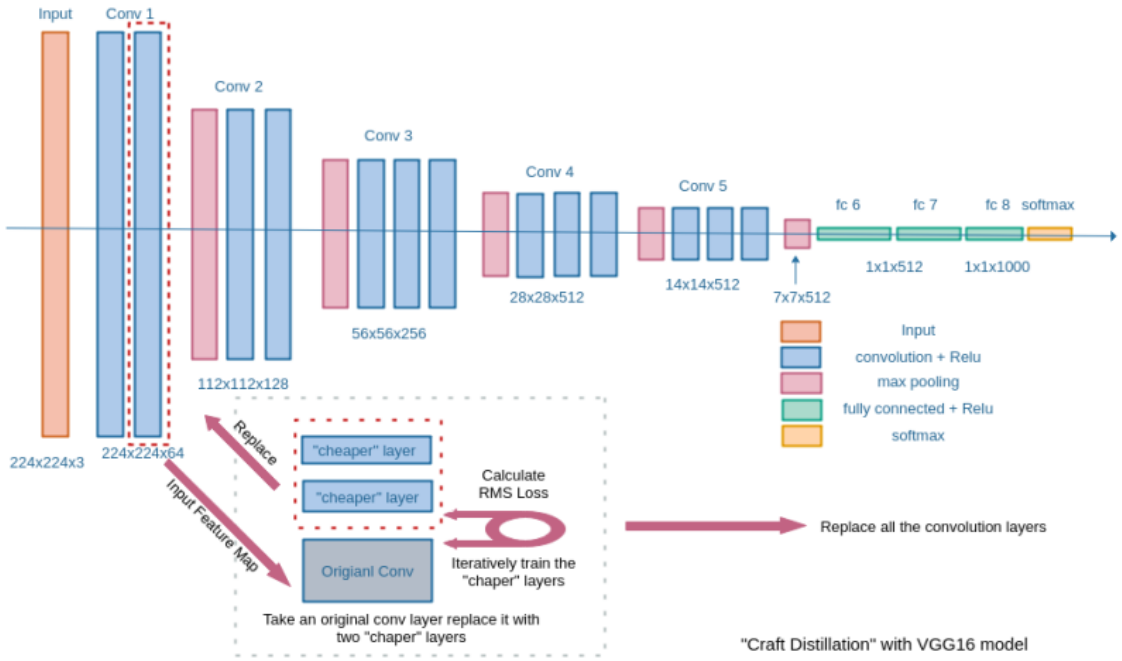


Figure 4.2: Overview of the craft distillation process. A convolutional layer of the dense model is selected as the “Teacher" to train cheaper depthwise separable convolution layers. The trained cheaper layers are inserted back to replace the original convolutional layer. Such a layer-wise distillation process is repeated in a top-down manner to replace other convolutional layers in the dense model.

convolutional layer. This process is referred to as layer-wise distillation. Craft distillation performs layer-wise distillation repeatedly to replace several or all convolutional layers in the “teacher model". The large “teacher model" is transformed into a small model when craft distillation is completed.

In order to effectively carry out the craft distillation process described above, several key research questions need to be answered.

1. How to select the replacement blocks?

2. In what order should layers be replaced?

3. How does craft distillation perform compared to other structure-altering compression techniques?

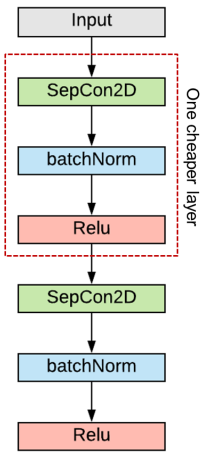
In the following subsections, we describe the designed experiments to answer

each of these questions. Each experiment is carefully designed to determine the best strategy when conducting craft distillation as well as evaluate the effectiveness of craft distillation compared to other model compression techniques. Subsequently, after demonstrating that this method works sequentially, we also demonstrate that independent training of the blocks is possible and just as effective and demonstrate how this can be exploited to speed up the compression of the model by leveraging parallelism.

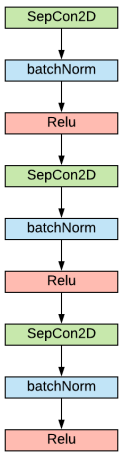
Architecture Search

Finding an efficient replacement block for the costly convolution layers is essential to the effectiveness of craft distillation. We first attempt to train a single depthwise separable layer as a replacement for a convolution layer but end up with inferior results. The next natural thought will be stacking several depthwise separable layers, which still use fewer parameters and flops than a traditional convolution layer, thanks to the properties of depthwise separable layers. We conduct a series of architecture searches to evaluate the best way to stack depthwise separable layers. The ideal architecture for replacement blocks should keep the added number of parameters as low as possible and be able to minimize the MSE loss.

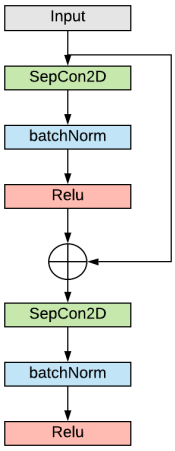
Figure 4.3 depicts the four candidate architectures explored in our experiments, which include the two depthwise separable layers (a), three depthwise separable layers (b), two depthwise separable layers with skip connections similar to residual blocks (c), and three depthwise separable layers with skip connections (d). We design experiments to evaluate the effectiveness of four candidates as follows. First, we train a VGG16 model on CIFAR-10 with a baseline accuracy of 86.45%. Each candidate architecture aims to replace the second convolution layer in the VGG model. We train each block for 20 epochs on the teacher model's intermediate activations. After that, the layers are inserted into the original model to replace layer 2, and the model's



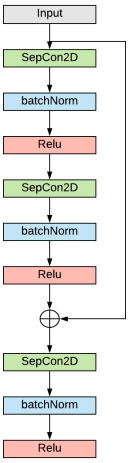
(a) Two Layer Replacement



(b) Three Layer Replacement



(c) Two Layers w/ Skip



(d) Three Layers w/ Skip

Figure 4.3: Candidate Replacement Layer Architectures. For the purpose of this work we refer to the combination of depth-wise separable, batch normalization, and relu activation as one layer.

accuracy is recorded. The weights of the 2nd layer are then fine-tuned using traditional cross-entropy loss on the labeled CIFAR-10 image data. All model weights other than the replacement block are frozen during fine-tuning.

Table 4.1 summarizes the results of the four candidate architectures. While the layers with skip connections do reasonably well, they do not outperform the simple two-layer variation. We hypothesize that this might be because these blocks are not

Table 4.1: RMS Loss, Top-1 Accuracy, and Fine Tuning Top-1 Accuracy results of different replacement architectures.

Arch	Loss	Top-1	FT Top-1
Two Layer	3.028E-05	86.40%	87.32%
Three Layer	7.410E-05	85.28%	85.37%
Two Layer w/Skip	3.536E-05	86.28%	86.98%
Three Layer w/Skip	3.877E-05	86.16%	87.03%

deep enough to have significant problems with disappearing gradients. Since the two layers architecture yields the best performance, we use it as the replacement block for convolution layers in the rest of the experiments.

Order of Layer-wise Distillation

In the subsequent Parallel Blockwise Distillation section, we will see that the layers can be distilled independently. However, experiments of training the layers in order still tell us how error propagates when it is dependent on other distillation processes. In the following experiment, we look at which order model layers should be replaced if done sequentially. The obvious choices are either top-down or bottom-up replacement. To determine the best strategy, we took the pre-trained VGG16 model on CIFAR-10 with an original accuracy of 86.45% as the baseline model. Each targeted layer for replacement is trained for 50 epochs on the intermediate layer activation. The layer is then inserted into the model, and the loss on the test set is recorded with the replacement blocks. The model is then fine-tuned for five epochs with all layers but the replacement block frozen. After the fine-tuning process, the weights that resulted in the lowest loss, either from fine-tuning or layer distillation, are kept. After the individual layer training, the resulting model becomes the “teacher model", and the next layer to be replaced was chosen.

As seen from Table 4.2, the top-down strategy results in the highest accuracy. It

Table 4.2: Results of Different Replacement Strategies

Model	Loss	Top-1
Original	0.50657	86.45%
Bottom Up	0.76339	81.67%
Top Down	0.61581	86.59%

Table 4.3: Comparison to L1-norm based Filter Pruning [1]. Results are calculated for the CIFAR-10 dataset.

Model	Top-1 (%)	MACCs	Reduced %	Parameters	Reduced %
VGG-16	93.3	3.13 x 10^8		1.5 x 10^7	
ResNet-50	**95.3**			2.38 x 10^7	
VGG-16 [1]	**94.0**	2.06 x 10^8	34	5.4 x 10^6	64
VGG-16 (ours)	92.7	7.93 x 10^7	**74.6**	3.57 x 10^6	**74.4**
ResNet-50 (ours)	94.5			1.49 x 10^7	**37.0**

is even higher than the original model, which could result from the base model not being trained optimally. It is also worth noting that even though the top-down strategy yields a higher accuracy than the original model, it also has a higher loss. This may indicate that it is less robust to adversarial attacks and is less sure of its predictions. It could also be that the top-down strategy alleviates the overfitting issue of the dataset.

A possible reason for the different accuracies is the fine-tuning. The bottom-up strategy gives less or no opportunity to fix mistakes in the layer representations. When training and replacing happen in a top-down manner, if the previous layer reaches some sub-optimal local minimal that shifts the layer representations, the next layer still has a chance to repair it and restore accuracy through fine-tuning.

Comparison to Structured Pruning

To the best of our knowledge, no current work compares directly to the proposed craft distillation method. Quantization [14] achieves similar model size reductions in memory, although it retains the same number of parameters and can increase the throughput of the calculations by using FP16 or Int8 data types. But it doesn't alter the structure of the original model or the total number of arithmetic operations. Unstructured pruning [14] does modify the model's structure but requires specialized hardware to run effectively. Structured pruning is probably the fairest comparison to craft distillation because it is performed on a layer-by-layer basis of a pre-trained model, and it does not require specialized hardware to work.

Table 4.3 and 4.4 illustrate the comparison results of craft distillation to the L1-norm based filter pruning [1] for compressing VGG-16 on the CIFAR-10 dataset and our results on ResNet-50. While the structured pruning method achieves a higher accuracy (94%) than our method (92.7%), we can reduce the total number of flops by 74.6%. Table 4.4 shows the detailed comparison of each individual layer. It appears that structured pruning does not prune each layer by the same percentage. It leaves several layers unpruned entirely. In craft distillation, however, there is no option of what percentage of weights will be removed from a layer. A layer is either completely replaced or is not replaced at all. We argue that craft distillation is more efficient than structured pruning to compress CNN models with negligible accuracy degradation. The essence of craft distillation is distinct from structured pruning as well.

Table 4.4: Individual layer comparison to L1-norm based Filter Pruning [1]. The two rightmost columns indicate the reduction in the percentage of MACCs for both methods. Craft distillation significantly reduces MACCs (by 74.6%) even though it does not apply to the fully connected layers.

layer type	w_i x h_i	MACCs	MACC% [1]	MACC% (ours)
Conv_1	32 x 32	1.8E+06	50%	0%
Conv_2	32 x 32	3.8E+07	50%	74.7%
Conv_3	16 x 16	1.9E+07	0%	64.3%
Conv_4	16 x 16	3.8E+07	0%	76.2%
Conv_5	8 x 8	1.9E+07	0%	65.5%
Conv_6	8 x 8	3.8E+07	0%	77.0%
Conv_7	8 x 8	3.8E+07	0%	77.0%
Conv_8	4 x 4	1.9E+07	50%	66.1%
Conv_9	4 x 4	3.8E+07	75%	77.4%
Conv_10	4 x 4	3.8E+07	75%	77.4%
Conv_11	2 x 2	9.4E+06	75%	77.4%
Conv_12	2 x 2	9.4E+06	75%	77.4%
Conv_13	2 x 2	9.4E+06	75%	77.4%
Linear	1	2.6E+05	50%	0%
Linear	1	5.1E+03	0%	0%
Total		3.1E+08	34%	74.6%

Parallel Blockwise Distillation

In this section, we extend the work from the Craft Distillation section and propose a novel parallel algorithm for block-wise knowledge distillation, which works as illustrated in Figure 4.1. First, it identifies all compressible layers in the teacher model and creates independent tasks for replacing these layers (one task for each layer). All tasks will be distributed to multiple GPUs via a group of TensorFlow instances and MPI processes by following a specific scheduling algorithm such as round robin, bin packing, or work stealing. Each process trains its replacement blocks independently on the activations of the layers to be replaced. Once all layers have been trained, the main MPI process gathers the weights from all processes and reassembles the compressed DNN. Last but not least, the reassembled student model will be fine-tuned to minimize the accuracy loss from the teacher model.

Independent Blockwise Distillation

Progressive blockwise distillation [75] works similarly to the Craft Distillation method described above. Still, a key difference is it defines groups of layer blocks from a teacher model and creates a less computationally intense set of layers to replace them. Another critical difference is they essentially use filter pruning to reduce computational complexity instead of replacing the layer blocks with a completely different design. Let us consider a teacher network T that can be represented as a composite function of its k subnetwork blocks:

$$T = c \circ f_k \circ f_{k-1} \circ ... \circ f_1 \tag{IV.1}$$

Knowledge distillation aims to derive a smaller student network S where its k network blocks are simplified networks but can mimic the larger networks in the teacher model.

Table 4.5: Summary of Notations

Term	Definition
T	Teacher Model
S	Student Model
block	sequential group of 1 or more layers
L^k_{local}	Local loss for block k
L^k_{cls}	Cross Entropy Loss
λ_{local}	hyper-parameter used in [75]

$$S = c \circ g_k \circ g_{k-1} \circ ... \circ g_1 \tag{IV.2}$$

In [75], the distillation process is conducted progressively in a "bottom-up" fashion, which refers to replacing the blocks in the order of (1, 2, ..., k). The blocks are trained on both the local activations of the blocks being replaced in the teacher model and the cross entropy loss from the student with the ground truth labels. Once a layer is trained, it is frozen in the student model, and the subsequent replacement layer is prepared. This requires each layer to be trained sequentially.

Let us consider the case of replacing block k in our model. Let f_k be the function that maps an input image to the activations at block k. Let g be the replacement block for block k. The local loss function can be represented as:

$$L^k_{local} = \frac{1}{N} \sum_{i=1}^{N} ||f_k(x_i) - g_k \circ f_{k-1}(x_i)||^2 \tag{IV.3}$$

And the combined loss would be defined as:

$$L^k = \lambda_{local} L^k_{local} + L^k_{cls} \tag{IV.4}$$

Where λ_{local} is a hyper-parameter to balance the loss terms. x_i is an individual

training image, and N is the total number of training images.

We improved the progressive blockwise distillation method [75] from the following perspectives. First, we consider only the local loss (equation IV.3) between the student model and the teacher model's feature maps. This makes it unnecessary to store a full copy of either the teacher or the student model into GPU memory. We can recreate the subset of teacher model layers up until and including the block to be replaced. The activations from the proceeding block are used as inputs for the student block as described in equation IV.3, and the activations from the original block being replaced become our labels.

Second, instead of replacing the blocks in a progressive bottom-up fashion, all candidate blocks are trained independently and simultaneously. This not only eliminates most unwanted dependencies but also allows us to reduce the accumulated error from the model. Once every replacement block is trained, the algorithm evaluates each one to see if they reach or exceed a predetermined accuracy threshold. Only the blocks that meet the predetermined accuracy threshold will be added to the reconstructed model. Then, the final student model will be produced after fine-tuning to regain accuracy.

Parallel Blockwise Distillation

Once the blockwise distillation process and replacement block architecture are determined, the last missing puzzle is how to parallelize the training process.

Algorithm 1 explains the parallel blockwise distillation method in detail. The first step is to identify all layers of the model to be replaced. Our algorithm considers every convolutional layer except 1x1 convolutions. This is because the 1x1 convolution is already more efficient than depthwise separable convolution. As depicted in Figure 4.4, each identified replacement block is viewed as an independent task. All tasks will be allocated to the available GPUs based on a selected scheduling

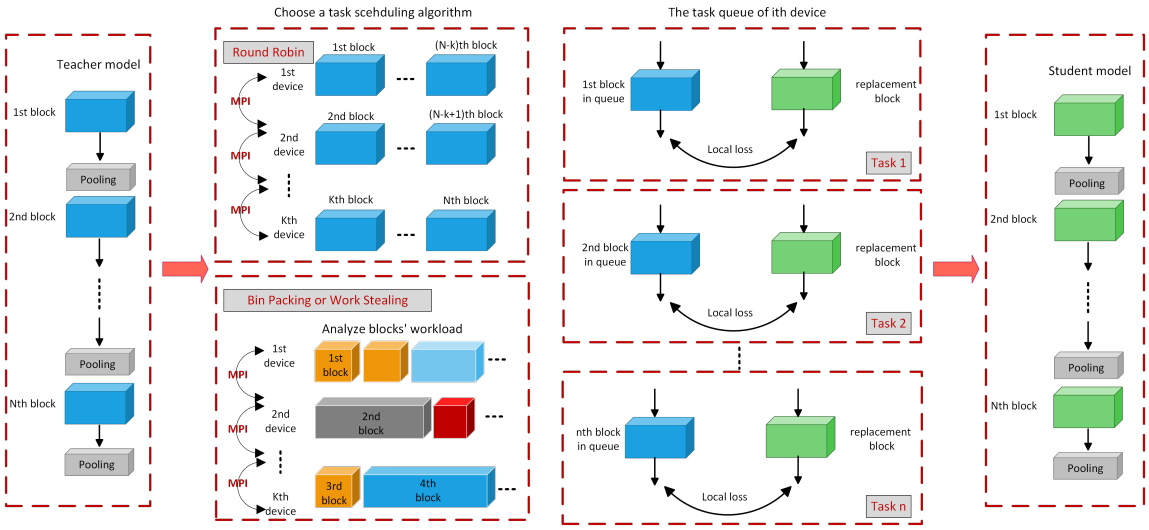


Figure 4.4: Overview of the parallel blockwise distillation process

Algorithm 1 Parallel Blockwise Distillation

Input : Teacher Network $T = c \circ f_k \circ f_{k-1} \circ ... \circ f_1$
Output : Student Network $S = c \circ g_k \circ g_{k-1} \circ ... \circ g_1$

identify all layers to be compressed
broadcast layer assignment to workers
for each worker **do**
    for all allocated layers **do**
        train each layer as described in IV
        if epoch %2 == 0 **then**
            evaluate model with student block
        end if
    end for
end for
gather all weights to process 0
for all layers **do**
    if accuracy > threshold **then**
        replace teacher block with student block
    end if
end for
fine-tune resulting model
return S

algorithm. Users can choose either a naive scheduling algorithm (e.g., round robin) or a more advanced scheduling algorithm (e.g., bin packing or work-stealing).

The choice of scheduling algorithms could influence the speedup because inappropriate scheduling algorithms may lead to load-balancing problems. The round-robin scheduling algorithm works well for DNNs with many blocks with a similar training time. However, for DNNs like VGG16, where the training time of each layer is vastly different, and the total number of replacement blocks is relatively small, naive scheduling such as round robin will likely cause load balancing issues. In this case, a more advanced scheduling algorithm, such as bin packing or work stealing, is needed. The bin-packing algorithm considers tasks with various execution times as items with varied weights, and the number of GPUs are viewed as the number of bins. The goal of bin packing scheduling is to allocate tasks to available GPUs so that each GPU has a balanced workload. More formally, given n bins of unlimited capacity and m tasks of weight w_m, the bin packing algorithm distributes the m tasks so that for every bin b_n their sum $\sum_{m \in b} w_m$ are as even as possible. In our algorithm, we leverage the bin packing implementation in the python library[76], which uses the heuristic Worst Fit Decreasing (WFD) approach that first sorts the items to be placed in decreasing order, then puts them one by one into the next most empty bin until all tasks are distributed. Bin packing scheduling is very effective, but it requires prior knowledge of each task's execution time. Generally speaking, if a model is compressed frequently, it is worthwhile to get such prior knowledge because obtaining the training time once could benefit all compression processes after that. However, when prior knowledge is unavailable, or the cost of receiving such information is too high, bin packing scheduling becomes less ideal. The work-stealing scheduling policy can address this issue as the load balancing can be achieved dynamically while running the model. In a work-stealing scheduler, each GPU has a queue of tasks. When a GPU runs out of work, it checks the queues of other GPUs and steals their tasks from

the tail of the queue. As a result, work-stealing achieves dynamic load balancing by distributing the tasks evenly over all GPUs. Our algorithm uses the work-stealing scheduling policy implemented by Dask [77], which is deployed on the Chameleon system [78].

Each layer is trained using local loss as described in equation IV.3, which helps significantly to reduce the amount of subnetwork graph that must be loaded in memory to perform the block distillation. This allows us to use larger batch sizes and perform the distillation on a single GPU. It also results in very fast training time per block because of the limited number of layers that require gradient information during the back prorogation phase. However, this also requires a careful balancing act to load and remove the necessary components of the computation graph at each part of the training phase. We load our teacher network and save a temporary copy of the teacher subnetwork up until the considered kth block for replacement. Then the GPU memory must be explicitly flushed. After that, the teacher subnetwork and the student block are loaded again and trained for two epochs. Then our algorithm checks how the distillation performs by evaluating the trained accuracy of each replacement block in the student model. Once all GPUs have finished their workload, the main MPI process with rank = 0 does a single gather from all GPUs, and layers are compared with a user-defined threshold for accuracy. Only student blocks that exceed the accuracy threshold will be used in the model reassembly process. When the compressed model is assembled, it is fine-tuned for a few epochs to recover accuracy and returned as the final student model output.

The performance of a parallel algorithm is usually affected by dependencies, synchronization overhead, load unbalancing, and speed gap between CPUs and GPUs. Our algorithm addresses each bottleneck as follows. First, by extracting local information, our algorithm makes each replacement block training an independent task, which eliminates most dependencies and only requires two synchronizations (one

at the beginning to dispatch tasks to GPUs and one at the end to assemble the student model). Second, by leveraging different scheduling algorithms (round robin or bin packing), the workload of each GPU is well balanced. Third, we can adjust the number of threads used for CPU data preprocessing to match the GPU speed on consuming training data. In our experiments, we observed that the recommended default settings in TensorFlow are not ideal, so we adjusted the number of CPU threads to achieve better speedup. Successfully tackling all these bottlenecks makes our proposed algorithm a very appealing solution to parallel knowledge distillation. Even better, our algorithm scales automatically on single machines with multiple GPUs and a multi-node GPU system. This scalability is transparent to users because they do not need to modify their hyper-parameters.

Experimental Results

Hardware Configuration

Experiments were conducted on two different systems. The first system is a single server with an AMD Ryzen Threadripper 2950x processor (16 physical cores with hyperthreading support for 32 threads), 4 Nvidia RTX 2080TI GPUs, and 128GB of DDR4 Memory in quad-channel configuration. The second one contains multiple nodes from the NSF Chameleon system [78]. Each Chameleon node has 192GB of DDR4 Memory, 2 Intel Xeon Gold 6126 Processors (12 physical cores, each with hyperthreading support for 24 threads), and an Nvidia RTX6000 GPU. These Chameleon nodes are connected as a cluster using a 10Gb network.

System Profiling Tool

A system profiling tool is developed to collect real-time information about CPU utilization, CPU power consumption, and multiple GPUs' utilization and power consumption. To ensure the profiling tool is lightweight, we choose a sampling

frequency of 1Hz, which has a negligible impact on the knowledge distillation process. The profiling for GPUs is via the Nvidia Management Library(NVML), which is already well-known; thus, the details are skipped here. We provide more details about profiling the AMD Ryzen Threadripper processor as follows.

Power consumption. The power consumption data is collected from the Model Specific Register(MSR) files. We can access different registers by seeking the MSR number as an offset. For example, we can go to each *cpunum* MSR directory and read 8 bytes starting from the offset $0xC001029A$ to get per CPU core energy. The package energy (i.e., total energy consumed on a single CPU chip) can be obtained from the offset $0xC001029B$. Since the recorded energy value is accumulative, we measure delta energy consumption over a sampling time period to calculate the average power consumption using the following equation:

$$CPU_{avgpower} = \Delta Energy * frequency \tag{IV.5}$$

CPU utilization. CPU usage information is obtained from the */proc/stat/* file, which holds various information about the kernel activities. To measure CPU utilization, we can sum up $CPUtime$ spends on user mode and kernel mode divided by the total CPU time spends within a sampling interval. The average CPU usage can be calculated with the equation below:

$$CPU_{avgusage} = \frac{(\Delta CPU_{user} + \Delta CPU_{kernel})}{\Delta CPU_{total}} \tag{IV.6}$$

Training Details

The CIFAR10 implementations are trained for 30 epochs per layer (although most converged within 5-10 epochs) and fine-tune the assembled model for 20 epochs. ImageNet models use provided Keras applications and pre-trained weights. The layers

are each trained for roughly the equivalent number of total steps as the CIFAR10 models, which results in about 1.5 epochs on the larger dataset. Models are then fine-tuned for two epochs over the full dataset when assembled. ResNet includes many 1x1 convolutional layers, which are not replaced in our algorithm. During the fine-tuning stage of ResNet, all 1x1 convolution layers are frozen and fully connected layers so that only the replacement blocks are fine-tuned. Further implementation details can be found on the Github at https://github.com/codestar12/Parallel-Independent-Blockwise-Distillation.

Speedup and Accuracy

Table 4.6: Speedup and Efficiency of ResNet and VGG on CIFAR10 (Single AMD Server - Bin Packing Scheduling)

# of GPUs	Time (s)	Speedup	Efficiency	DNN Model
4	2749.81	3.53	0.96	ResNet
3	3876.03	2.50	0.83	ResNet
2	5060.97	1.92	0.88	ResNet
1	9693.24	1	1	ResNet
4	1274.77	3.08	0.77	VGG
3	1603.12	2.45	0.82	VGG
2	2214.57	1.77	0.89	VGG
1	3920.53	1	1	VGG

In this section, we present experimental results to verify the proposed algorithm can achieve near-linear speedup without compromising accuracy. Table 4.6 and 4.7 depict the speedup and efficiency (the ratio of speedup over the number of GPUs) for VGG and ResNet when using our parallel knowledge distillation algorithm on the CIFAR10 dataset. We observe a 1.92, 2.50, and 3.53 speedup for ResNet and 1.77, 2.45, and 3.08 speedup for VGG when using two, three, and four GPUs, respectively, on the single AMD server with 4 GPUs. Meanwhile, we achieve even higher speedup

Table 4.7: Speedup and Efficiency of ResNet and VGG on CIFAR10 (Distributed Cluster - Work Stealing Scheduling)

# of GPUs	Time (s)	Speedup	Effeciency	DNN Model
4	2094.21	3.87	0.9685	ResNet
2	4064.43	1.996	0.998	ResNet
1	8113.087	1	1	ResNet
4	1556.94	2.93	0.732	VGG
2	2456.59	1.856	0.927	VGG
1	4558.57	1	1	VGG

Table 4.8: Speedup and Efficiency of ResNet on ImageNet (Single AMD Server - Bin Packing Scheduling)

# of GPUs	Time (s)	Speedup	Efficiency	DNN Model
4	13871.72	2.75	0.69	ResNet
3	16893.19	2.26	0.75	ResNet
2	20895.00	1.83	0.91	ResNet
1	38171.40	1	1	ResNet

(3.87 for 4 GPUs) and efficiency (0.97 for 4 GPUs) for ResNet running on the distributed cluster with work-stealing scheduling enabled (see Table 4.9). Table 4.9 shows that our algorithm can also achieve near linear speedup (1.998, 2.92, and 3.64 for two, three, and four GPUs) when training the student VGG using the ImageNet dataset, which is better than the results from the single AMD server (see Table 4.8). This is primarily due to the 16-core CPU's inability in the single server to simultaneously process a larger volume of ImageNet images for multiple GPUs.

In terms of accuracy, we compare our algorithm with the progressive blockwise distillation algorithm [75], which is the state-of-the-art knowledge distillation algorithm. Since the versions of trained VGG and ResNet models used in our experiment differ from those used in [75], the accuracy of our teacher models is not the same as the accuracy of teacher models used in [75]. For a fair comparison, we

Table 4.9: Speedup and Efficiency of ResNet on ImageNet (Distributed Cluster - Work Stealing Scheduling)

# of GPUs	Time (s)	Speedup	Efficiency	DNN Model
4	9234.09	3.639	0.910	ResNet
3	11517.76	2.918	0.973	ResNet
2	16821.35	1.998	0.999	ResNet
1	33605.60	1	1	ResNet

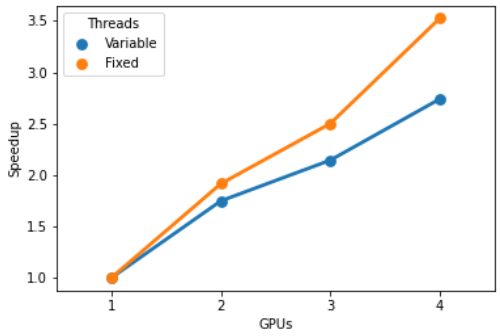
Table 4.10: Comparison of Top-1 Accuracy on CIFAR10 & CIFAR100

	Model	Top-1	Accuracy Change
CIFAR10	Teacher VGG [75]	86.61%	
	Student VGG [75]	83.56%	-3.05%
	Teacher VGG Ours	87.32%	
	Student VGG Ours	86.59%	-0.73%
	Teacher ResNet	90.36%	
	Student ResNet	88.77%	-1.59%
CIFAR100	Teacher ResNet [79]	71.21%	
	Student ResNet [79]	70.77%	-0.44%
	Teacher ResNet Ours	72.14%	
	Student ResNet Ours	70.53%	-1.61%

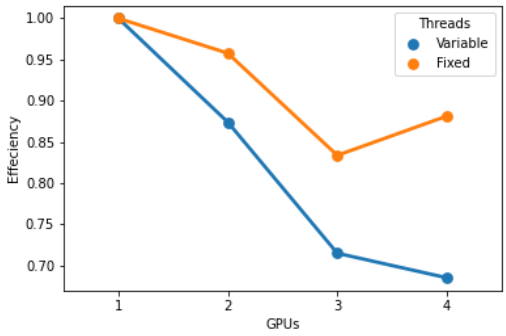
highlight the accuracy changes in the last column of Table 4.10, from which we can see that the accuracy drops by 3.05% for VGG when using the algorithm proposed in [75] on the CIFAR10 dataset. However, the accuracy degrades by only 0.73% for VGG and 1.59% for ResNet when using our algorithm. Table 4.11 shows that the accuracy of our algorithm on the ImageNet dataset decreases by merely 0.6% for VGG and even increases by 1.91% for ResNet.

Table 4.11: Comparison of Top-1 Accuracy on Imagenet

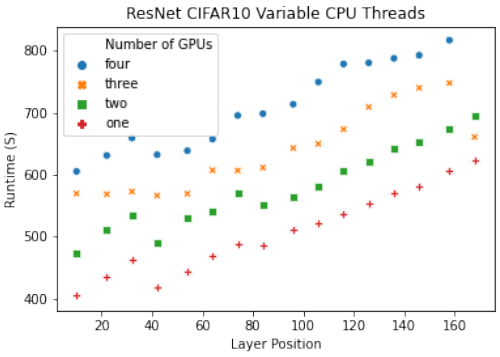
	Model	Top-1	Accuracy Change
ImageNet	Teacher VGG [75]	68.28%	
	Student VGG [75]	70.28%	+2.00%
	Teacher VGG Ours	65.7%	
	Student VGG Ours	65.1%	-0.6%
	Teacher ResNet	70.56%	
	Student ResNet	72.47%	+1.91%



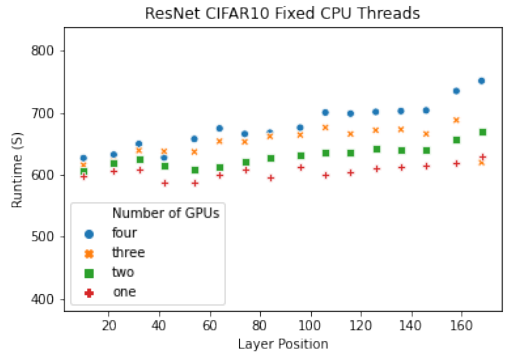
(a) Speedup of ResNet on CIFAR10



(b) Efficiency of ResNet on CIFAR10



(c) ResNet CIFAR10 layer runtimes with a variable number of threads

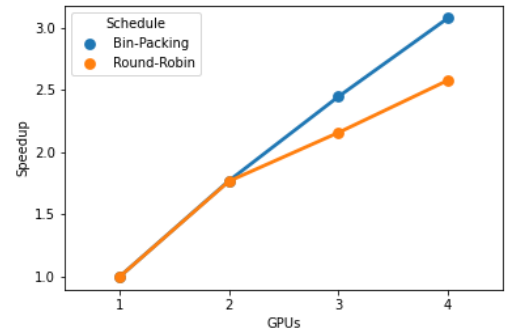


(d) ResNet CIFAR10 layer runtimes with a fixed number of threads

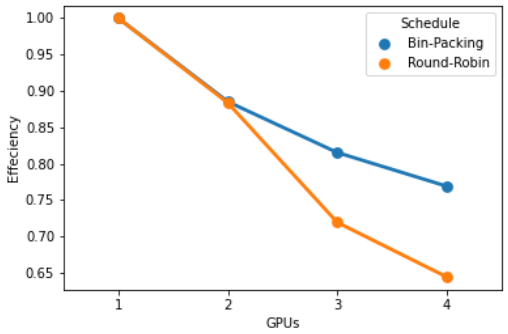
Figure 4.5: Comparison of ResNet speedup when using fixed or variable number of threads to preprocess data.

Impact of Load Balancing

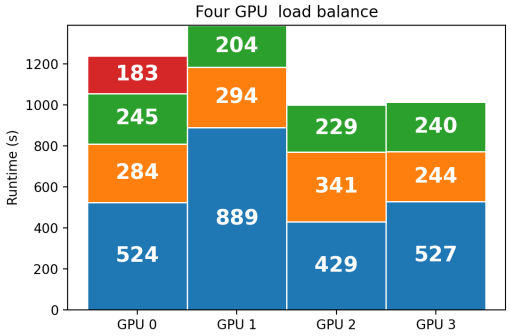
Another critical factor that affects speedup is load balancing, which is primarily determined by the characteristics of tasks and how these tasks are scheduled for



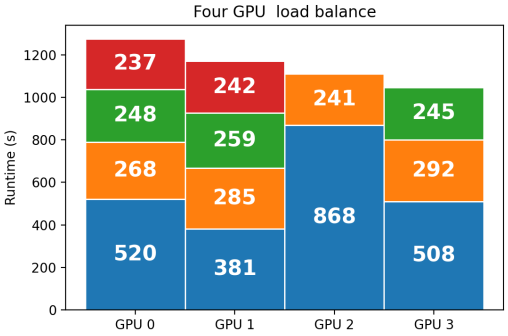
(a) Speedup of VGG16 on CIFAR10



(b) Efficiency of VGG16 on CIFAR10



(c) Round Robin task allocation for VGG16



(d) Bin Packing task distribution for VGG16

Figure 4.6: Comparison of VGG16 speedup when using Round Robin or Bin Packing to schedule layers.

execution. The naive round-robin scheduling achieves good load balancing for ResNet because tasks in ResNet have similar runtimes. However, naive scheduling can seriously impact speedup for DNNs with much fewer blocks to replace and more significant variations in feature map size like VGG16. Figure 4.6c shows how round-robin scheduling assigns the 13 VGG tasks (the number inside each task indicates its runtime in seconds) to four GPUs, from which we can see that the workload of GPU0 and GPU1 is much higher than the workload of GPU2 and GPU3. On the other hand, the bin packing scheduling allocates tasks more evenly (see Figure 4.6d), which significantly increases the speedup of 4 GPUs from around 2.5 to 3.0 and efficiency from 0.65 to 0.8. However, bin packing scheduling requires prior knowledge

about the runtime of each parallel task, which may not be available. In that case, work-stealing scheduling can dynamically achieve load balancing.

Energy Savings

Energy consumption is another concern when training and compressing large DNNs. Since DNNs are often trained and evaluated frequently in production applications, even relatively small improvements in energy efficiency can lead to a significant reduction in energy and carbon emissions over time. In this section, we use the greenup metric to compare the energy consumption of our parallel compression algorithm with the serial implementation. Greenup was first introduced by Abdulsalam et al. in [80] and defined as follows:

$$Greenup = Energy_{serial}/Energy_{parallel} \tag{IV.7}$$

Since the total energy consumption is the accumulated product of runtime over power, all factors that can affect runtime or power can influence total energy consumption. For example, the Dynamic Voltage and Frequency Scaling (DVFS) techniques have been enabled on both CPU and GPU as default settings, which constantly try to reduce power when workload drops. Other factors may have conflicting effects on runtime and power. For example, using more GPUs to achieve a larger speedup will help reduce runtime but will also increase both the CPU power and GPU power.

Table 4.12 summarizes the total energy and greenup when running our parallel algorithm on the single AMD server using different numbers of GPUs on ResNet and VGG, respectively. Our algorithm achieves a maximum of 1.29x greenup (i.e., 29% energy savings) on ResNet and a maximum of 1.19x greenup (i.e., 19% energy savings) on VGG, both when using four GPUs. The energy savings mainly come from dramatically shortened training time, as discussed in Table 4.6.

Table 4.12: Greenup of ResNet on CIFAR10

# of GPUs	Energy (kJ)	Greenup	DNN Model
4	977.39	1.29	ResNet
3	1067.82	1.18	ResNet
2	1106.68	1.18	ResNet
1	1263.43	1	ResNet
4	893.44	1.19	VGG
3	929.30	1.14	VGG
2	950.70	1.12	VGG
1	1061.98	1	VGG

We also notice that VGG has less energy reduction than ResNet. It is partially because VGG16 has fewer tasks that can be evenly distributed on multiple GPUs, as illustrated in Figures 4.6 (c) and (d). As a result, GPUs that finish their tasks much earlier would have to stay idle, waiting for other GPUs to complete. In our experiments, we find that idle devices also consume considerable energy over time without contributing useful work. Moreover, since the CIFAR10 dataset is relatively small for the distillation process, the overall system utilization is not high ($< 60\%$ most of the time). For distillation on larger datasets (e.g., ImageNet), the execution time of each task will be much longer, and system utilization will be higher. Consequently, more energy savings are expected because the load imbalancing issue and the DVFS techniques will generate less impact on greenup.

Conclusions and Future Work

Knowledge distillation is a promising technique to compress large deep neural networks (DNNs) by replacing the complex sub-networks in the teacher model with simplified sub-networks in the student model. However, existing knowledge distillation algorithms take a long time to train. In this chapter, we propose a novel

parallel block-wise distillation algorithm that can significantly reduce the distillation process's training time and energy consumption. The experimental results running on an AMD server with four Geforce RTX 2080Ti GPUs show that our algorithm can achieve 3x speedup plus 19% energy savings on VGG distillation and 3.5x speedup plus 29% energy savings on ResNet distillation, both with negligible accuracy loss. The speedup of ResNet distillation can be further improved to 3.87 when using four RTX6000 GPUs in a distributed cluster. In addition, our method can leverage different scheduling algorithms (e.g., bin packing or work stealing) based on the nature of the target DNNs to achieve good load balancing. More importantly, our algorithm can scale automatically and transparently when more GPUs are available without requiring users to tune their hyper-parameters.

Our current work can be further extended in two directions. First, we confirm that our method works well on the convolutional layers of VGG and ResNet in this work. We believe it can be applied more broadly, and future work can be done to evaluate its effectiveness on other models or different types of layers. Second, since our method uses only local loss and does not require labeled data, we plan to investigate further if our approach can work with unlabeled data and unsupervised learning.

V. REDUCE, REUSE, RECYCLE: IMPROVING TRAINING EFFICENCY WITH DISTILLATION

Introduction

Neural network training has a waste problem. Considerable attention has been given to the financial cost, electricity consumption, and carbon footprint of neural networks [81, 82, 83]. At the same time, many of these figures under-represent the full cost of training a network because they only measure the costs of the final training run. Far more resources are dedicated to *searching* for the optimal hyper-parameters for a given model, dataset, or recipe of techniques [84, 85]. During this search phase, trained models' performance metrics are used to reduce the hyperparameter search space; the representations that models have learned are rarely leveraged to improve training. This leads us to define the *Iterated Runs Problem*: How can previous training runs be used to improve the efficiency[1] of a subsequent training run?

Several machine learning research areas—some of them seemingly quite disparate—address the Iterated Runs Problem, including sample pruning and core-set selection [87, 88, 89, 90, 91, 92, 93, 94, 95], active learning [96], model averaging [97, 98, 99], and knowledge distillation (KD) [17, 100]. Work in these areas is not always conducted or framed with the goal of improving training efficiency. Some approaches are scientifically valuable but impractical. For example, [92]'s memorization and influence scores for data pruning require training several models equal to the size of the training dataset. Similarly, many works within knowledge distillation seek to compress models or maximize model quality without considerations for resource usage during training. Work that studies distillation for training efficiency

[1]We define efficiency here as in [86]: achieving a target level of model quality using fewer resources (e.g. GPU hours) than a baseline, or achieving an increased level of model quality using the same resources as a baseline.

typically measures units of optimization steps [55, 101, 102, 103, 104]. However, optimization steps overlooks computational burden of distillation and training, and accelerator resources are typically priced in units of time, not optimization steps.

Framing knowledge distillation as a solution to the Iterated Runs Problem and a means to improve training efficiency leads to a number of research questions:

- Does distillation improve training efficiency?
- Traditional distillation paradigms distill for all of training, but is this necessary to obtain the benefits of distillation? Can we optimize traditional distillation paradigms to reduce resource usage while retaining model quality improvements?
- Hyperparameter sweeps can still result in suboptimal models. Can they be salvaged to make useful teachers?
- Previous work has also shown that distilling from ensembles of models can yield benefits beyond those of distilling from a single model [102, 105, 106, 104], but distilling from multiple models imposes increased computational cost. Can we obtain the benefits of having multiple teacher models without paying the full computational cost?
- What is the utility of extending training time?

We conducted a series of experiments to investigate the utility of distillation for improving training efficiency in an Iterated Runs scenario using ResNet-50 [58] trained on ImageNet [107] and BERT [108] trained with a masked language objective on C4 [109] and evaluated on GLUE [110]. These experiments used a hyperparameter sweep across four learning rates, then distilled a fifth model of the same architecutre with one (or more) of the trained models. Our results are as follows:

- Distillation consistently improves training efficiency. We found distillation can speed up training up to 1.96x[2] in ResNet-50 trained on ImageNet, up to 1.20x on BERT when evaluated on masked pretraining accuracy, and up to 1.42x on BERT when evaluated on GLUE.

- Optimal distillation schedules vary across models. Distilling for the entirety of training is optimal for ResNet-50. In contrast, training BERT with distillation for all of the training *decreases* efficiency. Instead, distillation for BERT yields optimal results when only performed for the first 20-50% of training.

- Model quality does not guarantee teacher quality. Training with distillation is almost always more efficient than without, even when using a poor-quality model as a teacher in both ResNet-50 and BERT.

- Randomly sampling one teacher model from a pool of teachers on each iteration provides similar quality gains as those obtained from using that same pool of models as an ensemble of teachers on every iteration in ResNet-50 trained on ImageNet. This effectively reduces the runtime cost of teacher ensembles from $O(N)$ to $O(1)$.

- By combining our methods with extended training durations, we set new performance records for ResNet-18 and ResNet-34 on ImageNet, without the use of supplementary training data or augmentations. We further improve these results by using teachers that are themselves trained with distillation, demonstrating that the benefits of distillation are “heritable" between generations of student models. We refer to the practice of using distilled teachers as *Matryoshka Distillation*.

These results show that distillation can substantially improve training efficiency in image classification and language modeling. Furthermore, our results show that

[2]All speedups are measured on 8x NVIDIA A100 accelerators

distillation improves the efficiency of training, regardless of the quality of the teacher model. We also show that the benefits of distillation on training speed and model quality are fungible. Thus, our proposed optimizations to distillation protocols can improve model quality and reduce training costs. Taken together, this work emphasizes the utility of distillation for improving the efficiency of training deep neural networks.

Related Work

Knowledge Distillation

Knowledge distillation [17] is a method that improves and compresses models [100]. Teacher models are often larger than student models, though large size discrepancies can reduce the benefits of distillation [73]. Distillation is also included in many recipes that push the limits of model quality [111, 112, 113], and many approaches utilize distilled models of similar or identical architecture [55, 101, 102, 105, 106].

Teacher Ensembling and Self-Distillation

The Iterated Runs Problem emphasizes self-distillation, distilling from previous checkpoints in a training run [101, 102, 105, 114], and distilling from ensembles of teachers, which has been shown to improve model quality effectively [115, 105, 103, 116, 114, 117, 106, 104]. While our work explores both aspects of this question approaches combine both self-distillation and ensembling [105, 114, 106].[118] suggests that ensembles benefit from response diversity but did not examine distillation specifically.

Distillation in Language Models

While many studies have examined distillation to improve the quality of vision models, fewer works have studied language models [114]. Most research on the distillation of language models focuses on improving model compression rather than model quality [119, 120, 121, 122, 123, 124].

Distillation for Stepwise Training Speedups

Numerous works have claimed that distillation improves training efficiency by demonstrating reductions in optimization steps used to achieve a given level of model quality relative to a baseline [55, 101, 102, 103, 104]. Notably, [104] showed that much of the benefit of distillation can be obtained at reduced computational cost (in optimization steps) by distilling intermittently (i.e., once every K steps). They also showed that randomly sampling of teacher models from a pool of teachers at each step is nearly as effective as distilling from the entire ensemble of teachers. Unfortunately, measuring "efficiency" in optimization steps overlooks the increased computational cost of distillation, leaving it unclear whether these approaches improve resource usage efficiency.

Methodology and Experimental Setup

We designed a series of experiments examining the utility of distillation to improve training efficiency. We measured efficiency as defined in [86]: a model is efficient if it achieves the same level of quality as a baseline but with fewer resources (e.g., GPU hours), or a greater level of quality for the same amount of resources as the baseline (i.e., a Pareto improvement). Our experiments addressed the following questions:

- Can distillation improve training efficiency?

- Should you distill for all of training?
- Does model quality predict its quality as a teacher?
- Are multiple teachers more helpful than a single teacher?
- Can extended training help distillation?

Our experiments used models with identical architectures, optimizers, and dataset. For each experiment, train the *same* model architecture to the same or higher quality with fewer resources. We use the term “distillation" even though the student and teacher model architectures are identical in an experiment. Our goal is to examine whether the teaching signal can improve training efficiency. For image classification, we measured efficiency as speedup to reach the Top-1 accuracy of the baseline. For BERT we evaluated both Masked Language Modeling (MLM) accuracy when distilling and pre-trained model performance on downstream tasks from GLUE [110]. We *did not* conduct experiments that perform model compression.

We used only response-based distillation, not internal features, in order to compare approaches across domains and model architectures. We compared the use of both Kullback-Leibler (KL) divergence loss $\mathcal{L}_{kl}$ as described in[17] and MSE loss $\mathcal{L}_{MSE}$. Student models were trained using a linear combination of the KD losses $\mathcal{L}_{kd}$ as cross-entropy loss, $\mathcal{L}_{ce}$.

$$\mathcal{L} = \lambda \mathcal{L}_{kd} + \mathcal{L}_{ce}$$

where λ controls the weight of the KD loss term.

Data and Models We performed model distillation on two domains and tasks: image classification with ResNet-50 [58] trained on ImageNet [107] and Masked Language Modeling with BERT [108] pre-trained on C4 [109] and finetuned on GLUE [110]. For ResNet-50, we followed the procedure described by [58] using 224 x 224 test resolution. Notably, we used SGDW [125] as the optimizer and cosine annealing [126]

as the learning rate scheduler. We additionally extend this setup to ResNet-18 and ResNet-34, to test the effect of longer training duration. For BERT pre-training, we used AdamW [125] and linear decay. Instead of a fixed warmup length, we scaled the warmup period by the percentage of training duration. Details are in table 5.1.

Table 5.1: Shared Training hyperparameters for teacher models and students.

Model	ResNet-50	BERT
Batch size	2048	4096
Training Duration (baseline model)	90 epochs	286.72M sequences
Max Sequence Len	N/A	128
Optimizer	SGDW	AdamW
Weight Decay	5.00E-04	1.00E-05
Momentum	0.875	N/A
Warmup	8 epochs	6% of training duration
Scheduler	Cosine Annealing	Linear Decay

Table 5.2: Training hyperparameters used for teacher models.

Hyper Parameters	Model			
ResNet	B1	B2	B3	B4
lr	1	2.045	0.01	0.1
BERT	A1	A2	A3	A4
lr	5.00E-04	5.00E-04	1.00E-04	1.00E-04
wd	1.00E-05	1.00E-04	1.00E-05	1.00E-04

Hardware and Trainer We conducted our experiments using Composer [127], a PyTorch [128] library for efficient training. Our plots are visualized using Seaborn [129]. All experiments were conducted on 8x NVIDIA A100 80GB.

Simple Distillation Setup

To understand the benefits of distiation to improve efficiency, we conducted a hyperparameter sweep across five learning rates (all other hyperparameters kept constant). We then selected the highest-quality model from the sweep as the baseline

and the teacher model to train a student model with distillation. There are two variants of this experiment: one holds the learning rate schedule fixed, while the other accounts for changes to the learning rate schedule and shortens the training duration. We then compared the wall clock time-to-train for the baseline model to reach its final eval accuracy to the wall clock time-to-train for the distilled model to reach the baseline model's final eval accuracy.

Scheduled Distillation Setup

While many distill for the entirety of the training, distilling for a subset of training may be more efficient. In these experiments, we allowed student models to train for a percentage of the total training duration (e.g. 25%, 50%) and then stop distilling. These experiments were motivated by work highlighting the impact of interventions during the early phase of training [130, 131, 132, 133, 134, 135] and preliminary results implying the viability of scheduled distillation to reduce computational costs [104].

Teacher Selection Setup

While there are varying results regarding the correlation between the quality of a model and the quality of the same model as a teacher [136], it's unclear whether sub-optimal models can improve training efficiency. If sub-optimal teacher models consistently have a negative impact on training efficiency, then training with distillation could take longer to reach baseline accuracy than training without distillation.

To investigate the impact of model quality on training efficiency, we apply the process of training with distillation for the same set of training durations (and scaled learning rate schedules) described in the last section, utilizing each possible teacher model our hyperparameter sweep.

While publicly-available checkpoints allow for easy access to possible teachers, it's also possible that one could obtain a teacher model as a result of a hyperparameter sweep. To account for the cost of training teachers, we quantified the total efficiency improvement from distillation by adding the wall-clock training time for the teacher to the the wall-clock train time for the student with distillation.

Ensembling Setup

Using a single teacher fails to leverage the resources spent on training the remaining models that *are not* used as teachers. Additionally, numerous studies have shown that using ensembles of teachers can improve model quality more than using a single teacher [115, 105, 103, 116, 114, 117, 106, 104]. It's possible that the quality improvement from adding more teachers outweighs the additional computational burden and that composition of the teacher ensemble affects training efficiency. If *consistency* is important, the static and constant response provided of greedily selected ensemble of teachers will be the most effective (as noted in [113]). Alternatively, sampling from a larger population of *diverse* teachers may be the most efficient configuration.

Given the research on teacher ensembles and in the spirit of the Iterated Runs Problem, we studied the efficiency of teacher ensembles for distillation in three paradigms.

Our first method selected the models with the best accuracy one at a time. Testing all possible combinations of teachers would be computationally prohibitive, but scaling up the size of our teacher ensembles in a greedy fashion from the models with the highest accuracy is a reasonable and practical approach to building such an ensemble.

Our second method ensembled on-the-fly by loading all models onto GPU memory, then choosing the desired number at random without replacement on each

step and averaging their responses. This does slightly impact hardware performance as more memory must be used even for the models not selected on a given step.

Our third method sampled a single teacher model at random at each training step. It's possible that exposure to the same variety of models is a sufficient approximation of a larger ensemble without the additional computational overhead.

In these experiments, we again used the same set of teacher models from Table 5.3. We also add an additional teacher model trained with mix-up to improve model diversity. We now average the ensemble responses such that our new loss term is:

$$\mathcal{L} = \lambda \mathcal{L}_{mse}(z_t, z_s) + \mathcal{L}_{ce}$$

Where z_s is the student logit vector and $z_t = \frac{1}{n}\sum_1^n z_n$ is the mean teacher logit vector for n number of teachers.

We compare these models to a selection of results from our previous section: 1. A naive baseline distilled from the model with the best accuracy. 2. The top performing distilled model from each training duration. 3. The top performing teacher model from each training duration as a baseline.

The naive approach distills using the teacher with the best-reported accuracy at 90 epochs of training on ImageNet. The baselines from Section V serve as stronger baselines which would be impractical and computationally prohibitive for most applications.

Extended Distillation Setup

Drawing inspiration from the work of [113], we tested the utility of extending the duration of training time. apply our recipe to ResNet-18 and ResNet-34, and substantially increase our training times. We performed the same hyperparameter sweep over learning rates used for ResNet-50 teachers A1-A4 and create an ensemble of teacher models for both ResNet-18 and ResNet-34. We did not tune these or any

other hyperparameters to improve the quality of these models.

Experimental Results

Our experiments tested whether distilation can improve efficiency. As described in [86], if a distilled model longer wallclock time to reach the baseline model's final accuracy, it is not efficient. If the distilled model to reaches the baseline final accuracy faster than the baseline model or reaches a higher accuracy at the same training time, distillation improves training efficiency. Section V tests a simple distilation setup, while later sections propose modifications that may provide further gains in efficiency.

Table 5.3: Results of hyperparameter sweep of teacher models on both ResNet-50 ImageNet and BERT on C4 and KD. The teacher model is the highest-quality model (B1 for ResNet-50, A1 for BERT) at standard training length (90 epochs for ResNet-50, 286.72M sequences for BERT). KL: KL-Divergence distillation loss; MSE: Mean-squared error distillation loss.

ResNet-50 - ImageNet	Model Epochs	B1	B2	B3	B4	KD - KL	KD - MSE
Top-1 Val Accuracy	22	72.31%	72.96%	39.53%	68.16%	74.82%	75.64%
	45	75.43%	75.64%	60.85%	73.46%	76.51%	76.73%
	90	76.62%	76.40%	68.85%	75.86%	77.333%	77.21%
	135	76.79%	76.52%	71.27%	76.85%	77.42%	77.31%
	180	76.87%	76.51%	72.29%	77.17%	77.53%	77.37%
BERT - C4 MLM	Model Training Steps	A1	A2	A3	A4	KD - KL	
MLM Val Accuracy	17500	63.11%	62.59%	53.38%	48.89%	64.94%	
	35000	65.49%	64.77%	58.52%	51.54%	66.71%	
	52500	66.59%	65.79%	61.13%	52.28%	67.43%	
	70000	67.31%	66.41%	62.71%	52.40%	67.88%	
	87500	67.77%	66.77%	63.79%	52.49%	68.23%	

Simple distillation improves efficiency in ResNet-50 but not BERT

Our experiments demonstrate that simple distillation can substantially improve training efficiency for ResNet-50 trained on ImageNet, but may not improve training

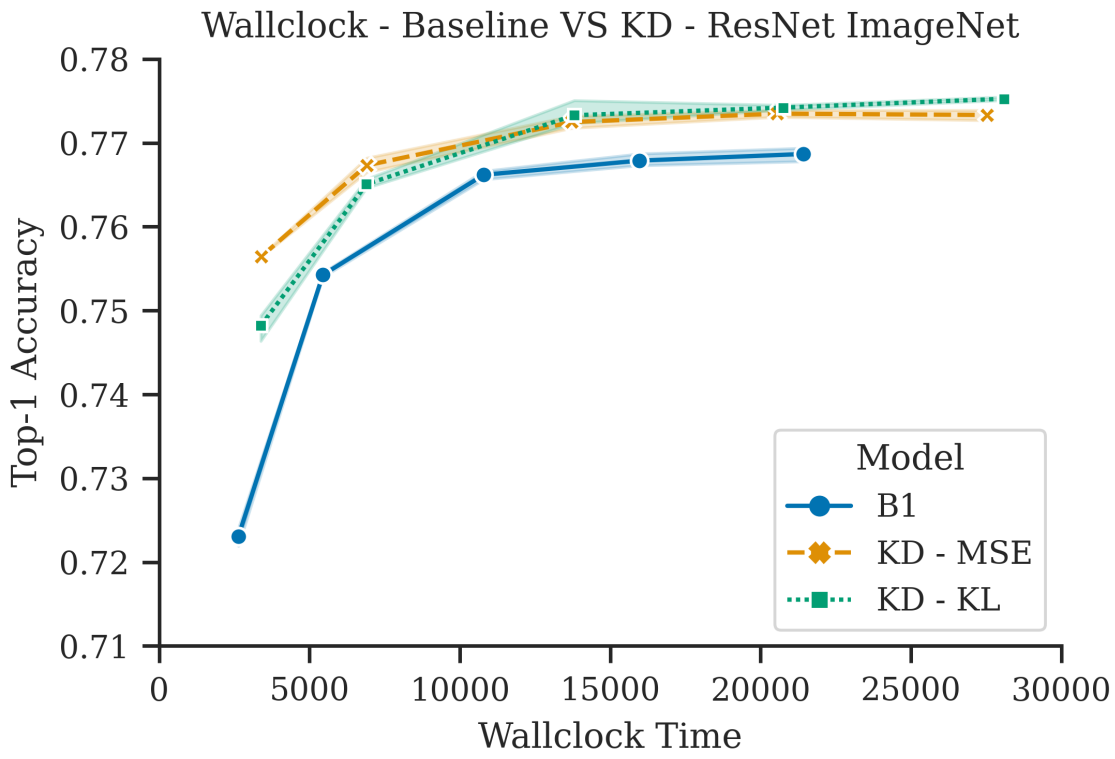


Figure 5.1: ResNet-50 trained on ImageNet with vs. without distillation. Wallclock time-to-train (x-axis) comparison of teacher model B1 vs student models trained with B1 as a teacher using MSE and KL losses. Points denote models trained for the number of epochs reported in Table 5.3.

efficiency for BERT trained on C4. These results suggest that additional modifications may be need to improve efficiency.

In ResNet-50 trained on ImageNet, distillation, holding the learning rate schedule fixed, substantially improves eval accuracy but is much slower than training the baseline model. After 90 epochs, the baseline model (B1) reached 76.6% eval accuracy in 179.8 minutes. To achieve the same level of accuracy, the distilled model took 200 minutes but as at epoch 80 of 90. At the end of 90 epochs, the distilled model reached a final eval accuracy of 77.2%.

When we trained the distilled model for 45 epochs and scaled the learning rate decay, we found that we were able to reach the final accuracy of the 90-epoch baseline in 91.6 minutes, a 1.96x speed-up (Figure 5.1).

We repeated these experiments using BERT trained on C4 (see Section V), holding the learning rate schedule fixed. Training our baseline on 286,720,000 sequences of 128 tokens took 11.3 hours and achieved 67.31% val MLM accuracy and 83.37% accuracy on GLUE. As with ResNet50, naïvely applying distillation yielded worse results: the distilled model achieved baseline MLM accuracy in 18 hours.

We also performed distillation using MSE loss student-teacher loss instead of KL-divergence/student-teacher loss. For ResNet on ImageNet, we found that MSE is a substantial improvement over KL-divergence for shorter training (22, 45 epochs), but performed slightly worse for longer training (Figure 5.1). However, we observed that MSE has very little observable impact on BERT pretraining.

Our experiments demonstrate that distillation can substantially improve training efficiency for ResNet-50 trained on ImageNet, but that distillation may not improve training efficiency for BERT trained on C4. However, it's possible that our distillation configuration is sub-optimal, and there are potential efficiency gains that remain to be realized.

Early-phase-only distillation is optimal for BERT but not ResNet-50

Our results suggest that constant BERT benefits from partial distillation, while ResNet-50 benefits from constant distillation. Stopping distillation early for BERT *always* resulted in better or equal MLM accuracy than distilling for the duration of training (see Table 5.5). Although the optimal distillation duration varies for different training durations and teacher models, we found that the best percentage for training with distillation tended to fall within the range of 15-40%. For ResNet-50 trained on ImageNet, we found very little efficiency improvement from stopping distillation compared to distilling the whole training duration (see Table 5.4). This is slightly contradictory to the results in [137], but maybe a result of a difference in choice of learning rate schedules as the is a significant boost in training accuracy near the end

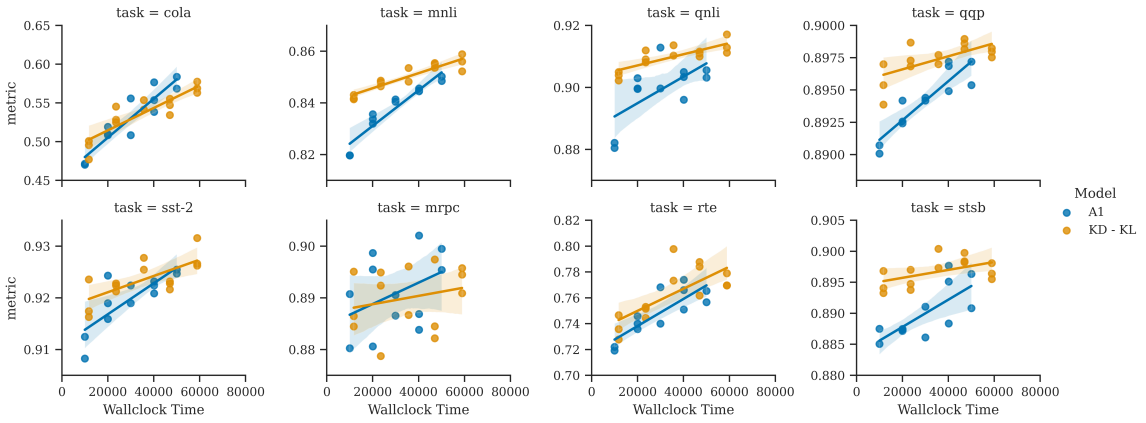


Figure 5.2: Wallclock comparisons of individual GLUE tasks for the highest quality baseline model (A1) and distilled model using A1 as a teacher (KD - KL) when pretraining for different durations. Benefits from distillation are not equal across all tasks. Generally, tasks with larger finetuning datasets saw the most benefit. Note: the x-axis denotes the pretraining duration, not the duration of training on GLUE.

Table 5.4: Results from Early Stopping KD for ResNet-50 Trained on ImageNet.

	Epochs	Training Duration	Distillation Pct	Top-1
Baseline	45	5455.4	NA	75.64%
	90	10799.2	NA	76.62%
Distillation	45	5812.5	25%	75.95%
	45	6175	50%	76.46%
	45	6887	100%	77.48%

of training with cosine schedulers.

Based on these BERT results, we ran distilation for the first 30% of training, then re-ran our experiments training BERT on C4 with distillation. Using this setup, we reached the same MLM accuracy as the baseline teacher model up to 1.20x faster (Figure 5.3a). The student model also reaches an equal GLUE score as the baseline up to 1.42x faster than the baseline (Figure 5.3b).

Downstream GLUE tasks with larger datasets most benefited from distilation, while with smaller and less stable datasets benefited less. Figure 5.2 shows that applying distillation in the early phase of training allowed QNLI, and STSTB to

Table 5.5: Results from Early stopping on BERT trained on C4. Stopping early cuts training duration significantly and improves MLM accuracy for the same number of training steps.

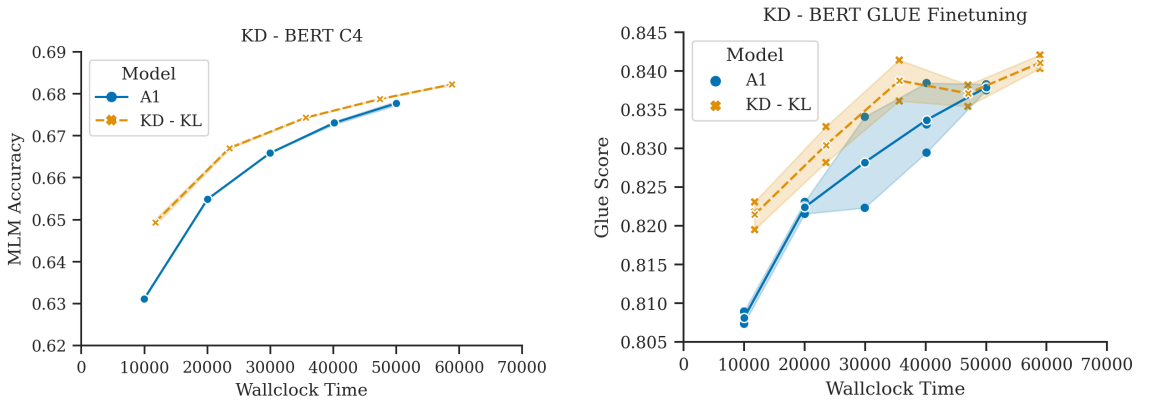
	Training Steps	Training Duration	Distillation pct	Weight	MLM Acc
Baseline	35000	20025.79	NA	NA	65.49%
	52500	29945.81	NA		66.59%
	70000	40156.39	NA		67.31%
Distillation	35000	37351	100%	1.00E-04	65.65%
				1.00E-05	66.22%
				1.00E-06	66.15%
	35000	23534.05	30%	5.00E-04	66.71%
	52500	35640.97			67.43%

match or exceed the baseline model in only 25% of the training steps. MNLI, QQP, SST-2 were able to match or exceed the teacher model in 50% of the training steps.

Sub-optimal models can be ideal teachers

Our teacher selection experiments allow us to make the following two conclusions. First, the risk of a training outcome worse than training without distillation is very low. Even when selecting the worst teacher, distillation is likely to be an efficiency improvement. Second, the quality of a model does not consistently predict the quality of that model as a teacher for distillation. When accounting for the cost of training the teacher model, it can still be more efficient to train a teacher and distill a student than train a single model.

We present our ResNet-50 results in Figure 5.4. Three of the four models trained with distillation reached the accuracy of their teacher models in shorter time. None of the teacher models resulted in a step-wise improvement. There was low correlation between teacher model accuracy and teaching ability. Our best model from the first results section (B1) was the lowest performing teacher on 2 of 5 training durations (135, 180). Our third-best model(B4) became more competitive as the training duration increased to 180 epochs.



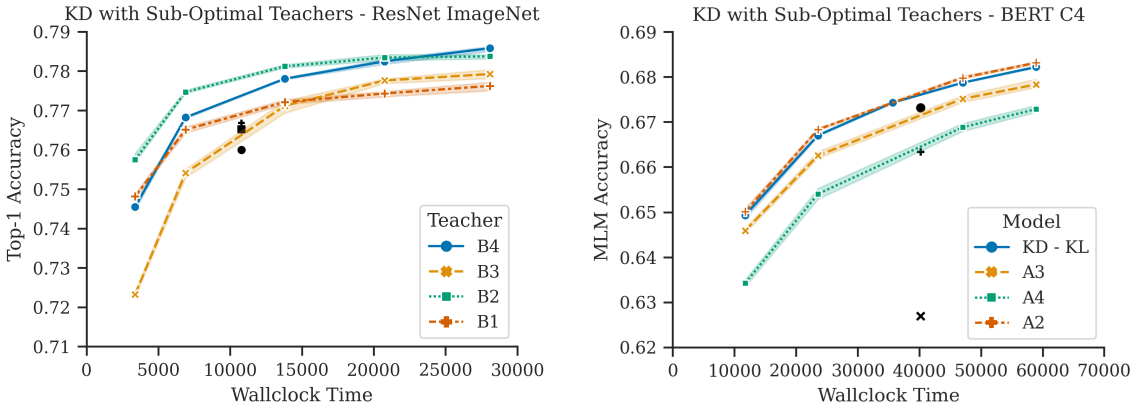
(a) MLM Accuracy of pretraining on C4 dataset with BERT utilizing Early-phase-only distillation. Individual points along each line denote models trained for the number of sequences reported in Table

(b) GLUE score of BERT when applying the optimization of stopping distillation early. When trained for as many steps as the teacher model there is a slight dip in GLUE score. Note: distillation was only used during pretraining; GLUE finetuning was performed without distillation.

Figure 5.3: Quality vs. wall-clock-time when training with vs. without knowledge distillation in ResNet50 and BERT.

We saw similar outcomes for masked language pre-training on BERT (Figure 5.4b). Two of the models trained with distillation reached the accuracy of the best non-distilled model (A1) faster than the best non-distilled model, and *all* of the models trained with distillation reached the accuracy of the remaining three non-distilled models faster than the non-distilled models themselves. Again, the highest quality, non-distilled model (A1) was not the best teacher (A2). Additionally, the student taught with the lowest-quality non-distilled model (A4) ultimately reached an MLM accuracy that is nearly 10 percentage points higher than its teacher at equal wallclock time.

Accounting for Teacher Training the Cost As shown in Table 5.6, for half of our teacher models it is more efficient to train the teacher model for 90 epochs and then distill it into a student model for 45 epochs than to continue training the teacher model for a total of 180 epochs. These results indicate that even in the scenario where a teacher model is unavailable, distillation can still be efficient.



(a) For ResNet models, we see that not only is the best teacher *not* the one with the highest accuracy, but also depends on the training duration.

(b) For BERT we see similar results with A2 being the best choice of Teacher.

Figure 5.4: Wallclock time vs. accuracy plots for distilling ResNet and BERT using teachers with lower accuracy. For both domains, we see the best teacher model is *not* the teacher with the highest accuracy. Teacher accuracies and wallclock times are marked in black with matching markers. (A4 omitted for scaling)

Efficient teacher ensembling via random selection

For all our experiments, we observed an efficiency improvement over our baseline model (Figure 5.5). In shorter training regimes (22 and 45 epochs), we found that selecting the best model from a single teacher performs best (Best Single Teacher). However, this approach required an exhaustive search for the best teacher: not just for the task, but also the training duration (as we have shown in the previous section). We also found that distilling from a greedy ensemble of teacher models is not a Pareto improvement as compared to selecting the best single teacher, but randomly sampling the ensemble on the fly *is*.

We found that random sampling a single teacher per step from an ensemble of teachers is able to reach the Top-1 Accuracy of our best baseline model trained for 90 epoch **1.85**x faster. For all ranges of values, sampling a teacher at random for distillation resulted in a speedup of **1.32-3.16**x for all training duration.

Table 5.6: Accuracy and runtime duration of distilled models when accounting for the total cost of training. The top shows teacher epochs + student epochs and total duration. The bottommost row shows the teacher training duration. Even when including the duration of training for the teacher models, half of our models would be more efficient with distillation than additional training.

Method	Epochs	duration	Teachers/baseline			
			B1	B2	B3	B4
KD	90 + 45	17654.5	76.51	77.47	75.41	76.82
	90 + 90	24595.1	73.33	78.12	77.12	77.81
Baseline	180	21418	76.87	76.51	72.29	77.17

We also found that the choice of the loss function is not trivial. When using KL-divergence, the inclusion of specific models hurt performance. Those models dragged the accuracy down to the worst single-model distillation performance. MSE on the other hand, while never as good at distilling with only a single teacher, was more robust to teacher selection.

Extended distillation achieves state of the art on ResNet-18/34

In the preceding experiments, we examined the optimal combination of teachers for Pareto-optimal distillation. We also applied our recipe to ResNet-18 and ResNet-34, and substantially increase our training times.

Simply increasing training duration and randomly selecting a single teacher per step for distillation achieved state-of-the-art results for both ResNet-18 and ResNet-34 (73.11% and 76.75%, respectively) at 224px resolution. Details can be seen in Table 5.7. We note that, unlike previous works, our technique did not use any additional data or augmentations.

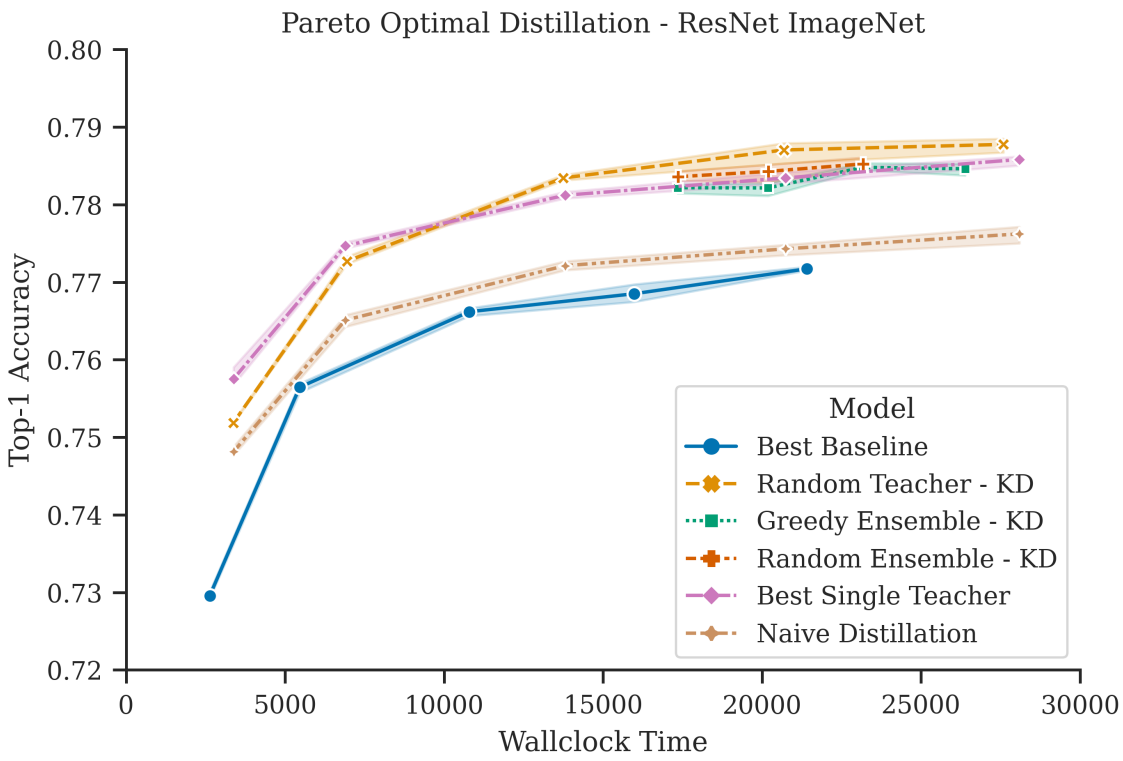


Figure 5.5: Pareto curve comparing single model distillation, multi-model distillation, and randomly sampling teachers. Dark green: baseline (no kd); blue: single model distillation; orange: 90 epochs with 2, 3, or 4, teachers chosen greedily (MSE loss); light green: 90 epochs with 2, 3, or 4, teachers randomly-selected from 5 (MSE) loss; pink: randomly select 1 teacher from 5, trained for 22, 45, 90, 135, or 180 epochs

Matryoshka Distillation

Distillation improves model quality. But we showed that the quality of a model does not necessarily correspond to its quality as a teacher. This leads to the question of whether the benefits of distillation are cumulative. Are distilled models better teachers than non-distilled models? Or are distillation-related improvements in model quality not heritable between generations of students? We investigated this by training students with teachers that were themselves trained with distillation, which we refer to as *Matryoshka Distillation*. We found that ensemble diversity is still important and we use the same learning rates as the original teacher ensemble. This

Table 5.7: Comparison of Distillation and other state-of-the-art resnet results. Distilling from an ensemble of teachers with random selection achieves a new state-of-the-art for ResNet18 and ResNet-34 on ImageNet trained without additional data. The best results from other methods are highlighted in blue, best overall results are in bold. (note: † indicates results reported in [138])

Method	Technique	Top-1 ResNet-18	Top-1 ResNet-34	Epochs
Baselines	Torchvision	69.75	73.31	90
	Ours	70.55	74.01	90
KD	KD † [17]	71.37		100
	CRD [139]	71.17		100
	FT † [140]	71.13		100
	PAD-L2 [141]	71.71		100
Semi-Sup	Billion-scale [142]	72.6		
Data Aug.	Strikes Back [143]	71.5	76.4	600
	SAMix [144]	72.33	76.35	300
Ours	Radom Selection	72.93	76.75	720
		73.11		900
	Matryoshka KD	73.04	**76.88**	720
		73.27		1080

resulted in a further 0.16% and 0.13% accuracy improvement for ResNet-18 and ResNet-34, respectively, on top of our previous SOTA distillation results (Table 5.7).

Discussion

We conducted a series of experiments to investigate the utility of distillation for improving training efficiency using ResNet-50 trained on ImageNet and BERT trained on C4 and evaluated on GLUE. We found that distillation improves training efficiency: it can speed up training by up to 1.96x in ResNet-50 trained on ImageNet and up to 1.42x on BERT when evaluated on GLUE. We also found that distillation schedules matter. Distillation for BERT yields optimal results when it is only

performed for the first 20-50% of training, but that distilling for the entirety of training is optimal for ResNet-50 on ImageNet. Furthermore, we found that model quality does not consistently predict teacher quality. Training with distillation is almost always more efficient than training without distillation, even when using the poorest-quality model as a teacher, in both ResNet-50 and BERT. We were also able to reduce the runtime cost of teacher ensembles from $O(N)$ to $O(1)$ while still retaining their benefits to distillation by randomly sampling one teacher model from a pool of teachers on each iteration. We also observed differences between mean squared error (MSE) and KL-Divergence (KL) distillation loss in ResNet-50 trained on ImageNet. MSE is more robust—it more consistently yields higher quality student models across a wide range of hyperparameter values—but KL-Divergence distillation loss yields the *best* student models. Finally, by using distilled teachers and combining our methods with extended training durations, we were able to set new SOTA performance for ResNet-18 and ResNet-34 on ImageNet without the use of supplementary training data or augmentations.

[113] obtain impressive results with extended distillation using a single teacher, showing that it's possible for the student to reach the accuracy of the teacher. Our work shows that when using identical architectures, it's possible to improve the student **beyond** the accuracy of the teacher.

A caveat to our work is that distillation requires loading a teacher model into GPU memory. Depending on the size of the teacher and student models and the amount of GPU memory, distillation can exceed the GPU memory capacity. In such a scenario, memory-saving techniques such as gradient accumulation may be necessary, which can impose additional computational overhead. Accordingly, distillation may no longer improve efficiency in such a scenario.

Another shortcoming of this work is that we do not provide a precise nor analytical basis for our recommendation about when to stop using distillation when

pretraining BERT. Our recommendation of stopping 20-50% through training is derived entirely from empirical observation.

Our findings demonstrate that distillation consistently improves training efficiency in both image classification and language modeling across a range of training durations and teacher model qualities. We also show that the benefits of distillation on training speed and model quality are fungible, meaning that our proposed optimizations to distillation protocols—using distilled teacher models, randomly sampling from ensembles of teacher models, and distilling for the beginning 30% of training (in BERT)—can be flexibly leveraged to reduce training time or increase model quality, depending on the needs of the practitioner. This work emphasizes the value of distillation for improving the efficiency of training deep neural networks.

VI. CONCLUSIONS

In conclusion, this dissertation has examined neural network model compression from multiple angles. The work has been rooted in practical solutions that can realize model acceleration without specialized hardware and minimize training time and resources while also aiming to shed light on the underlying mechanisms that allow compression techniques to operate.

First, Chapter II investigates the limitations of unstructured neural network pruning to reduce large neural networks' size and computational demands. Specifically, it analyzes three pruning methods using Singular Vector Canonical Correlation Analysis and demonstrates that this process significantly changes the representation of the model. This understanding of the evolving nature of pruned models informs Chapter III. Chapter III examines the bias and errors introduced into neural networks during pruning and how to mitigate them using knowledge distillation. It finds a strong correlation between model similarity and bias in pruned networks and provides insight into which knowledge distillation techniques best preserve the model's original qualities. Chapter IV explores an alternative to structured pruning and traditional knowledge distillation by constructing more efficient replacements of sub-model components and replacing expensive individual sections of the model in a block-wise manner. It then presents how block-wise distillation can be parallelized in local and distributed systems. Chapter V investigates how distillation can accelerate the training of deep neural networks and amortize the training cost of models' hyper-parameters sweeps. It also achieves SOTA performance on ResNet-18 and ResNet-34, significantly improving existing results while using teacher models of much lower accuracy challenging traditional ideas of "compression" and knowledge distillation.

Discussion and Future Research

The throughline in this dissertation's work has been challenging convention and reexamining practices taken for granted in the community that trains neural networks. I firmly believe that much of the conventional wisdom of network compression research is misguided and needs re-evaluation. I am excited by the findings of Chapter V and the following new research directions. It points to 1) The best-compressed model you can find is simply the one that meets your inference needs, and you train to the highest quality. 2) Knowledge Distillation (at least response-based distillation) functions via a mechanism that behaves very differently than many in this field understand. It is, in fact, essential that the labels teachers provide are wrong, and specifically *how* they are wrong. I believe that what is valuable about the predictions of the model used in union with the ground truth labels is that you are receiving an excellent prior of what a model should predict over that dataset. Otherwise, why would it be possible to train a model to a higher quality than its teacher models? Why would it be more successful to use the same model architecture as a teacher compared to one of a higher capacity?

In summary, this dissertation has contributed to many avenues of investigation. A better understanding of how model representations change during compression and how to measure and mitigate unwanted side effects of compression, and efficiency in knowledge distillation. It is my hope that the compilation of research presented in this dissertation provides fresh perspectives for those studying issues in compression, bias, and energy efficiency of AI models.

REFERENCES

[1] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[2] H. Lim, D. Cosley, and S. R. Fussell, "Beyond translation: Design and evaluation of an emotional and contextual knowledge interface for foreign language social media posts," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2018.

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[4] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, "Faster gaze prediction with dense networks and fisher pruning," *arXiv preprint arXiv:1801.05787*, 2018.

[5] N. Wadhwa, R. Garg, D. E. Jacobs, B. E. Feldman, N. Kanazawa, R. Carroll, Y. Movshovitz-Attias, J. T. Barron, Y. Pritch, and M. Levoy, "Synthetic depth-of-field with a single-camera mobile phone," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–13, 2018.

[6] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv preprint arXiv:1812.11941*, 2018.

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[8] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," *arxiv:2204.02311*, 2022.

[9] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, 2022.

[10] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, June 2022.

[11] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[12] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4820–4828, 2016.

[13] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[15] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Conference on Neural Information Processing Systems(NIPS)*, 2015.

[16] W. S. Sajid Anwar, Kyuyeon Hwang, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems*, 2017.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[18] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

[19] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022.

[20] T. Dettmers and L. Zettlemoyer, "The case for 4-bit precision: k-bit inference scaling laws," *arXiv preprint arXiv:2212.09720*, 2022.

[21] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, "Ultra-low precision 4-bit training of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1796–1807, 2020.

[22] C. Blakeney, Y. Yan, and Z. Zong, "Is pruning compression?: Investigating pruning via network layer similarity," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 914–922, 2020.

[23] C. Blakeney, N. Huish, Y. Yan, and Z. Zong, "Simon says: Evaluating and mitigating bias in pruned neural networks with knowledge distillation," *arXiv preprint arXiv:2106.07849*, 2021.

[24] C. Blakeney, G. Atkinson, N. Huish, Y. Yan, V. Metsis, and Z. Zong, "Measuring bias and fairness in multiclass classification," in *2022 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–6, IEEE, 2022.

[25] C. Blakeney, X. Li, Y. Yan, and Z. Zong, "Craft distillation: Layer-wise convolutional neural network distillation," in *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pp. 252–257, IEEE, 2020.

[26] C. Blakeney, X. Li, Y. Yan, and Z. Zong, "Parallel blockwise knowledge distillation for deep neural network compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1765–1776, 2020.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2015.

[29] V. Lebedev and V. Lempisky, "Fast convnets using group-wise brain damage," in *CVPR*, 2016.

[30] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *NIPS*, 2015.

[31] W. Chen, J. Wilson, T. S., W. K. Q., and C. Y., "Compressing neural networks with the hashing trick," in *JMLR workshop*, 2015.

[32] H. Zhou, A. J. M., and P. F., "Less is more: Towards compact cnns," in *ECCV*, 2016.

[33] A. See, M.-T. Luong, and C. D. Manning, "Compression of neural machine translation models via pruning," in *CoNLL*, 2016.

[34] S. Narang, G. F. Diamos, S. Sengupta, and E. Elsen, "Exploring sparsity in recurrent neural networks," in *arXiv preprint arXiv: 1704.05119*, 2017.

[35] Y. Lecun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *NIPS*, 1990.

[36] R. Rigamonti, A. Sironi, V. Lepetit, and F. P, "Learning separable filters," in *CVPR*, 2013.

[37] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolitional neural networks with low rank expansions," in *BMVC*, 2014.

[38] C. Tai, T. Xiao, and X. Wang, "Convolutional neural networks with low-rank regularization," in *arXiv preprint arXiv:1511.06067*, 2015.

[39] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Doubly convolutional neural networks," in *NIPS*, 2016.

[40] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units," in *ICML*, 2016.

[41] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein, "Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability," in *NIPS*, pp. 6076–6085, 2017.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[43] Z. Mariet and S. Sra, "Diversity networks," in *ICLR*, 2016.

[44] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *IEEE Signal Processing Magazine*, 2018.

[45] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," in *arXiv preprint arXiv:1512.08571*, 2015.

[46] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," in *IEEE Access*, 2015.

[47] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019.

[48] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2019.

[49] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[50] S. Hooker, A. Courville, G. Clark, Y. Dauphin, and A. Frome, "What do compressed deep neural networks forget?," *arXiv preprint arXiv:1911.05248*, 2019.

[51] S. Hooker, N. Moorosi, G. Clark, S. Bengio, and E. Denton, "Characterising bias in compressed models," *arXiv preprint arXiv:2010.03058*, 2020.

[52] J. Buolamwini and T. Gebru, "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *Conference on fairness, accountability and transparency*, pp. 77–91, PMLR, 2018.

[53] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," in *ICLR*, 2017.

[54] F. Tung and G. Mori, "Similarity-preserving knowledge distillation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1365–1374, 2019.

[55] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4133–4141, 2017.

[56] N. Passalis and A. Tefas, "Learning deep representations with probabilistic knowledge transfer," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[57] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," *arXiv preprint arXiv:1910.10699*, 2019.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[59] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[60] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[61] R. Movva and J. Y. Zhao, "Dissecting lottery ticket transformers: Structural and behavioral study of sparse neural machine translation," *arXiv preprint arXiv:2009.13270*, 2020.

[62] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *arXiv preprint arXiv:1908.09635*, 2019.

[63] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, "Model cards for model reporting," in *Proceedings of the conference on fairness, accountability, and transparency*, pp. 220–229, 2019.

[64] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford, "Datasheets for datasets," *arXiv preprint arXiv:1803.09010*, 2018.

[65] A. Amini, A. P. Soleimany, W. Schwarting, S. N. Bhatia, and D. Rus, "Uncovering and mitigating algorithmic bias through learned latent structure," in *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 289–295, 2019.

[66] Z. Wang, K. Qinami, I. C. Karakozis, K. Genova, P. Nair, K. Hata, and O. Russakovsky, "Towards fairness in visual recognition: Effective strategies for bias mitigation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8919–8928, 2020.

[67] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, pp. 598–605, 1990.

[68] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the lottery: Making all tickets winners," in *International Conference on Machine Learning*, pp. 2943–2952, PMLR, 2020.

[69] Y. Guo, C. Zhang, C. Zhang, and Y. Chen, "Sparse dnns with improved adversarial robustness," *arXiv preprint arXiv:1810.09619*, 2018.

[70] L. Wang, G. W. Ding, R. Huang, Y. Cao, and Y. C. Lui, "Adversarial robustness of pruned neural networks," 2018.

[71] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *International Conference on Machine Learning*, pp. 3519–3529, PMLR, 2019.

[72] A. Ansuini, E. Medvet, F. A. Pellegrino, and M. Zullich, "On the similarity between hidden layers of pruned and unpruned convolutional neural networks.," in *ICPRAM*, pp. 52–59, 2020.

[73] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 5191–5198, 2020.

[74] W. Son, J. Na, J. Choi, and W. Hwang, "Densely guided knowledge distillation using multiple teacher assistants," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9395–9404, 2021.

[75] H. Wang, H. Zhao, X. Li, and X. Tan, "Progressive blockwise knowledge distillation for neural network acceleration.," in *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2769–2775, 2018.

[76] B. F. Maie, "bin packing python library." `https://pypi.org/project/binpacking/`, 2019.

[77] Dask Development Team, "Work stealing." `https://distributed.dask.org/en/latest/work-stealing.html`, 2016.

[78] "Nsf chameleon cloud system." `https://www.chameleoncloud.org/`.

[79] M. Gao, Y. Shen, Q. Li, J. Yan, L. Wan, D. Lin, C. C. Loy, and X. Tang, "An embarrassingly simple approach for knowledge distillation," *arXiv preprint arXiv:1812.01819*, 2018.

[80] S. Abdulsalam, Z. Zong, Q. Gu, and M. Qiu, "Using the greenup, powerup, and speedup metrics to evaluate software energy efficiency," in *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, pp. 1–8, IEEE, 2015.

[81] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 3645–3650, Association for Computational Linguistics, July 2019.

[82] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, Dec. 2020.

[83] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," *AAAI*, vol. 34, pp. 13693–13696, Apr. 2020.

[84] J. Dodge, S. Gururangan, D. Card, R. Schwartz, and N. A. Smith, "Show your work: Improved reporting of experimental results," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, (Hong Kong, China), pp. 2185–2194, Association for Computational Linguistics, Nov. 2019.

[85] A. Feder Cooper, Y. Lu, J. Z. Forde, and C. De Sa, "Hyperparameter optimization is deceiving us, and how to stop it," *In Submission*, Feb. 2021.

[86] D. Blalock, M. Carbin, L. Florescu, J. Frankle, M. L. Leavitt, T. Lee, M. Nadeem, J. Portes, N. Rao, L. Seguin, C. Stephenson, H. Tang, and A. Venigalla, "On Evaluating and Improving the Efficiency of Deep Networks," tech. rep., 2021.

[87] K. Vodrahalli, K. Li, and J. Malik, "Are All Training Examples Created Equal? An Empirical Study," Nov. 2018. arXiv:1811.12569 [cs, stat].

[88] M. Toneva, A. Sordoni, R. T. d. Combes, A. Trischler, Y. Bengio, and G. J. Gordon, "An Empirical Study of Example Forgetting during Deep Neural Network Learning," *arXiv:1812.05159 [cs, stat]*, Nov. 2019. arXiv: 1812.05159.

[89] S. Swayamdipta, R. Schwartz, N. Lourie, Y. Wang, H. Hajishirzi, N. A. Smith, and Y. Choi, "Dataset Cartography: Mapping and Diagnosing Datasets with Training Dynamics," Tech. Rep. arXiv:2009.10795, arXiv, Oct. 2020. arXiv:2009.10795 [cs] type: article.

[90] C. Coleman, C. Yeh, S. Mussmann, B. Mirzasoleiman, P. Bailis, P. Liang, J. Leskovec, and M. Zaharia, "Selection via Proxy: Efficient Data Selection for Deep Learning," Oct. 2020. arXiv:1906.11829 [cs, stat].

[91] K. Chitta, J. M. Alvarez, E. Haussmann, and C. Farabet, "Training Data Subset Search with Ensemble Active Learning," Nov. 2020. arXiv:1905.12737 [cs, stat].

[92] V. Feldman and C. Zhang, "What Neural Networks Memorize and Why: Discovering the Long Tail via Influence Estimation," Aug. 2020. arXiv:2008.03703 [cs, stat].

[93] M. Paul, S. Ganguli, and G. K. Dziugaite, "Deep Learning on a Data Diet: Finding Important Examples Early in Training," *Advances in Neural Information Processing Systems*, p. 12, 2021.

[94] S. Mindermann, J. Brauner, M. Razzak, M. Sharma, A. Kirsch, W. Xu, B. Höltgen, A. N. Gomez, A. Morisot, S. Farquhar, and Y. Gal, "Prioritized Training on Points that are Learnable, Worth Learning, and Not Yet Learnt," Sept. 2022. arXiv:2206.07137 [cs].

[95] B. Sorscher, R. Geirhos, S. Shekhar, S. Ganguli, and A. S. Morcos, "Beyond neural scaling laws: beating power law scaling via data pruning," Aug. 2022. arXiv:2206.14486 [cs, stat].

[96] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, "A Survey of Deep Active Learning," *ACM Computing Surveys*, vol. 54, pp. 180:1–180:40, Oct. 2021.

[97] M. Wortsman, G. Ilharco, J. W. Kim, M. Li, S. Kornblith, R. Roelofs, R. Gontijo-Lopes, H. Hajishirzi, A. Farhadi, H. Namkoong, and L. Schmidt, "Robust fine-tuning of zero-shot models," Sept. 2021.

[98] M. Matena and C. Raffel, "Merging Models with Fisher-Weighted Averaging," Aug. 2022. arXiv:2111.09832 [cs].

[99] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt, "Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time," *arXiv:2203.05482 [cs]*, Mar. 2022. arXiv: 2203.05482.

[100] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge Distillation: A Survey," *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, June 2021. arXiv: 2006.05525.

[101] C. Yang, L. Xie, C. Su, and A. L. Yuille, "Snapshot Distillation: Teacher-Student Optimization in One Generation," *arXiv:1812.00123 [cs]*, Nov. 2018. arXiv: 1812.00123.

[102] T. Furlanello, Z. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, "Born again neural networks," in *International Conference on Machine Learning*, pp. 1607–1616, PMLR, 2018.

[103] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble Distillation for Robust Model Fusion in Federated Learning," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 2351–2363, Curran Associates, Inc., 2020.

[104] X. Liu, A. Leonardi, L. Yu, C. Gilmer-Hill, M. L. Leavitt, and J. Frankle, "Knowledge Distillation for Efficient Sequences of Training Runs," July 2022.

[105] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, (Seoul, Korea (South)), pp. 3712–3721, IEEE, Oct. 2019.

[106] C. Wang, Q. Yang, R. Huang, S. Song, and G. Huang, "Efficient Knowledge Distillation from Model Checkpoints," Oct. 2022. arXiv:2210.06458 [cs].

[107] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[108] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[109] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv e-prints*, 2019.

[110] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding," Feb. 2019. arXiv:1804.07461 [cs].

[111] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10687–10698, 2020.

[112] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," *arXiv:2012.12877 [cs]*, Jan. 2021. arXiv: 2012.12877.

[113] L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil, and A. Kolesnikov, "Knowledge distillation: A good teacher is patient and consistent," *arXiv:2106.05237 [cs]*, June 2021. arXiv: 2106.05237.

[114] Y. Xu, X. Qiu, L. Zhou, and X. Huang, "Improving BERT Fine-Tuning via Self-Ensemble and Self-Distillation," Feb. 2020. arXiv:2002.10345 [cs].

[115] A. Malinin, B. Mlodozeniec, and M. Gales, "Ensemble Distribution Distillation," Nov. 2019. arXiv:1905.00076 [cs, stat].

[116] U. Asif, J. Tang, and S. Harrer, "Ensemble Knowledge Distillation for Learning Improved and Efficient Networks," Apr. 2020. arXiv:1909.08097 [cs].

[117] Z. Allen-Zhu and Y. Li, "Towards Understanding Ensemble, Knowledge Distillation and Self-Distillation in Deep Learning," July 2021. arXiv:2012.09816 [cs, math, stat].

[118] R. Gontijo-Lopes, Y. Dauphin, and E. D. Cubuk, "No One Representation to Rule Them All: Overlapping Features of Training Methods," *arXiv:2110.12899 [cs]*, Oct. 2021. arXiv: 2110.12899.

[119] S. Sun, Y. Cheng, Z. Gan, and J. Liu, "Patient Knowledge Distillation for BERT Model Compression," Aug. 2019. arXiv:1908.09355 [cs].

[120] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "TinyBERT: Distilling BERT for Natural Language Understanding," Oct. 2020. arXiv:1909.10351 [cs].

[121] W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju, "FastBERT: a Self-distilling BERT with Adaptive Inference Time," Apr. 2020. arXiv:2004.02178 [cs].

[122] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," Feb. 2020. arXiv:1910.01108 [cs].

[123] C. Xu, W. Zhou, T. Ge, F. Wei, and M. Zhou, "BERT-of-Theseus: Compressing BERT by Progressive Module Replacing," Oct. 2020. arXiv:2002.02925 [cs].

[124] W. Zhang, L. Hou, Y. Yin, L. Shang, X. Chen, X. Jiang, and Q. Liu, "TernaryBERT: Distillation-aware Ultra-low Bit BERT," Oct. 2020. arXiv:2009.12812 [cs, eess].

[125] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[126] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with warm restarts," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[127] H. Tang, R. Rahman, M. Patel, M. Nadeem, A. Venigalla, L. Seguin, D. S. Khudia, D. Blalock, M. L. Leavitt, B. Shah, J. Bloxham, E. Racah, A. Jacobson, C. Stephenson, A. Saini, D. King, J. Knighton, A. Ehsani, K. Jariwala, N. Niklas, A. Lamp, I. Shastri, A. Trott, M. Cress, T. Lee, B. Cui, J. Portes, L. Florescu, L. Li, J. Zosa-Forde, V. Ivanchuk, N. Sardana, C. Blakeney, M. Carbin, H. Lupesko, J. Frankle, and N. Rao, "Composer: A PyTorch Library for Efficient Neural Network Training," 2022.

[128] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[129] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. Publisher: The Open Journal.

[130] G. Gur-Ari, D. A. Roberts, and E. Dyer, "Gradient Descent Happens in a Tiny Subspace," Dec. 2018. arXiv:1812.04754 [cs, stat].

[131] A. Achille, M. Rovere, and S. Soatto, "Critical learning periods in deep networks," in *International Conference on Learning Representations*, 2018.

[132] L. Sagun, U. Evci, V. U. Guney, Y. Dauphin, and L. Bottou, "Empirical Analysis of the Hessian of Over-Parametrized Neural Networks," May 2018. arXiv:1706.04454 [cs].

[133] A. S. Golatkar, A. Achille, and S. Soatto, "Time Matters in Regularizing Deep Networks: Weight Decay and Data Augmentation Affect Early Learning Dynamics, Matter Little Near Convergence," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[134] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin, "Linear mode connectivity and the lottery ticket hypothesis," in *International Conference on Machine Learning*, pp. 3259–3269, PMLR, 2020.

[135] J. Frankle, D. J. Schwab, and A. S. Morcos, "The Early Phase of Neural Network Training," *arXiv:2002.10365 [cs, stat]*, Feb. 2020. arXiv: 2002.10365.

[136] G. Kaplun, E. Malach, P. Nakkiran, and S. Shalev-Shwartz, "Knowledge Distillation: Bad Models Can Be Good Role Models," Mar. 2022. arXiv:2203.14649 [cs, stat].

[137] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4794–4802, 2019.

[138] Y. Matsubara, "torchdistill: A Modular, Configuration-Driven Framework for Knowledge Distillation," in *International Workshop on Reproducible Research in Pattern Recognition*, pp. 24–44, Springer, 2021.

[139] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," in *International Conference on Learning Representations*, 2020.

[140] J. Kim, S. Park, and N. Kwak, "Paraphrasing complex network: Network compression via factor transfer," *Advances in neural information processing systems*, vol. 31, 2018.

[141] Y. Zhang, Z. Lan, Y. Dai, F. Zeng, Y. Bai, J. Chang, and Y. Wei, "Prime-aware adaptive distillation," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*, pp. 658–674, Springer, 2020.

[142] I. Z. Yalniz, H. Jégou, K. Chen, M. Paluri, and D. Mahajan, "Billion-scale semi-supervised learning for image classification," 2019.

[143] R. Wightman, H. Touvron, and H. Jégou, "Resnet strikes back: An improved training procedure in timm," *arXiv preprint arXiv:2110.00476*, 2021.

[144] S. Li, Z. Liu, D. Wu, Z. Liu, and S. Z. Li, "Boosting discriminative visual representation learning with scenario-agnostic mixup," *arXiv preprint arXiv:2111.15454*, 2021.