ML Training Under Tight Budget Constraints With Data Pruning and Model Scaling

December 8, 2024

Jiwon Chang **Ethan Chen**

Abstract

As deep learning models demand increasingly large datasets, the sheer size of training data has become a significant bottleneck for training state-of-the-art models. Consequently, considerable attention has been directed toward data-efficient machine learning, which seeks to accelerate model training by selectively removing data points that contribute the least to final model performance. However, prior works assumed that the model architecture is fixed, focusing solely on the impact of data pruning on model performance. Through a grid search experiment on a scalable family of ResNet models, we empirically explore the impact of model size, batch size, and data pruning on training latency and test set accuracy. We show that the optimal training strategy under a strict total latency constraint involves jointly pruning data and scaling down the model.

1 Introduction

Deep learning (DL) models have become a ubiquitous technology. Stakeholders often wish to train and deploy their own models for reasons such as task specialization, data security, and privacy. However, model performance improves logarithmically with dataset size, creating a demand for exponentially larger training sets [16]. This imposes an insurmountable computational burden for all but the largest developers, thereby hindering the democratization of contemporary DL techniques.

A promising solution to this challenge is data pruning, which involves removing data points that do not significantly contribute to the final model accuracy. Data pruning techniques have shown the ability to train both computer vision models and large language models (LLMs) [12, 13].

The goal of data pruning methods is to approximate the gradient of model parameters with respect to the ground truth data distribution using a subset of the training data [11]. This must be achieved without introducing more overhead than is saved through reduced data movement and computation costs during forward and backward passes. Current SOTA pruning methods focus on removing data points associated with small losses, as these correspond to small gradient norms and therefore have minimal impact on the model's decision boundaries [13, 15].

To the best of our knowledge, existing data pruning techniques assume a fixed model architecture and aim solely to refine the data pruning algorithm [1, 9, 11–15]. However, empirical scaling laws indicate that dataset size and model size should scale proportionally to minimize losses effectively, ensuring neither becomes a bottleneck [16].

In this work, we explore how to achieve a Pareto-optimal training strategy by combining data pruning and model scaling. Our findings demonstrate that under strict training budget constraints, the highest model accuracy is achieved by increasing the degree of data pruning while proportionally scaling down the model's width and depth.

2 Prior Work

We first conduct a comprehensive literature review of existing work on data-efficient machine learning (ML) and its limitations.

Low-loss pruning. The recent wave of data pruning research began with Selective Backprop [6]. This method proposed that data points with low loss are relatively unimportant for training. To identify such points, the technique performed an additional forward pass in each epoch, followed by a full sort, to skip backpropagation and gradient updates for a subset of low-loss data points.

Subsequent works in this vein have proposed pruning based on metrics such as low scalar loss, small normed loss vectors, or small gradient norms [9, 13–15]. Pruning schedules may either be static (performed once early in training) [13] or dynamic (adjusted iteratively throughout training) [14, 15]. However, the additional forward pass per epoch introduces significant computational overhead [6]. To mitigate this, researchers have explored leveraging stale loss information, with strategies such as:

- Refreshing stale loss metrics periodically [6].
- Using random sampling to ensure all data points are periodically revisited [9, 14].
- Applying online learning algorithms for real-time updates [15].

Explainable Data Difficulty. In the explainable AI (XAI) literature, several methods quantify the "difficulty" of a data point for a given model and task. Empirical studies reveal strong correlations between various measures of "easy" data points across modalities and architectures [8], including:

- Low loss [8].
- Small gradient norms [13].
- Easy to learn and hard to forget [17].
- Consistently learned early in training and in early model layers [2].
- Consensus among ensemble predictions [3].
- Contribution to hold-out performance [3].

These findings suggest that pruning low-loss data points is a fundamentally sound approach, as the "difficulty" of a data point during training is a consistently measurable one-dimensional concept [8].

High-loss pruning. Recently, researchers have identified benefits to pruning data points with abnormally high losses [9]. The intuition is that such points may be mislabeled, noisy, or outliers that do not significantly contribute to generalization [9]. Removing these points can enhance test-time performance by reducing overfitting to noisy or irrelevant data [9].

Coreset Methods. Another line of work focuses on approximating the total gradient using a small subset of data points by solving combinatorial optimization problems [11, 12]. These methods aim to provably select a subset that closely matches the gradient of the full dataset, providing a more principled alternative to low-loss pruning. While these approaches have achieved SOTA performance in some tasks, they come with significant computational overhead [11]. Specifically, they require access to gradient information and involve solving submodular optimization problems, which typically run in $O(n^2)$ time [10].

3 Problem Definition

Suppose that we are given a prediction task, a family of models that share the same architecture, and a dataset with predefined train/ test splits. Additionally, we are constrained by a strict total training latency budget. Our objective is to select a specific model from the model family and determine a pruning schedule that maximizes test set performance while adhering to the training time constraint.

For simplicity, we focus on image classification tasks. We draw our models from a family of ResNet models [5], where the width and

depth of each basic block are scaled in tandem, similar to the scaling strategy employed in EfficientNet [7]. We constrain the set of possible data pruning strategies to a method similar to Stale Selective Backprop [6], as described in Section 4.1. Additionally, we include minibatch size as a tunable parameter in the optimization problem.

Our general problem definition can be divided into two sub-problems:

1. Understanding the phenomenon: Conducting a scientific experiment to explore the relationships between the independent and dependent variables.
2. Auto-tuning optimization: Developing a method to predict the optimal model and pruning parameters for a given task without incurring significant overhead.

This report focuses on the first sub-problem: understanding the relationships through empirical experimentation. We leave the second sub-problem for future work.

4 Implementation

4.1 Pruning Algorithm

Our method. Our data pruning algorithm is an adaptation of Stale Selective Backprop [6], chosen for its low overhead and simplicity after evaluating various alternatives.

Similar to Stale Selective Backprop, we maintain a list of the most recently observed loss values for every data point in the training set. During each epoch, we subsample a uniformly random subset of the dataset. This subset is then sorted to approximate the threshold for the p^{th} quantile, where $p \in [0, 1]$. We prune all data points whose last observed loss is below this threshold. In expectation, this process prunes Np data points per epoch from a dataset of size N. However, this method does not guarantee an unbiased estimate of the true threshold due to the stale nature of the loss values.

Over time, our method may become less effective at pruning "easy" data points as loss information becomes increasingly stale. Prior research suggests that this degradation can negatively affect model performance, as the difficulty of data points evolves during training [8, 15]. While some works attempt to address this issue using active learning [15], evidence suggests that active learning methods are not always effective [14]. Consequently, we adopt a simpler approach: every k epoch, we train the model on the entire dataset to refresh all stale loss information [6].

4.2 Implementation Details

We implement the pruning logic using two custom Python classes: `PruningDataset` and `ScalablePrunableResNet`.

- `PruningDataset`: This is a PyTorch `Dataset` wrapper around an underlying `Dataset`. It manages the logic for selecting data points to prune and maintaining consistent mappings between external and internal indices. It returns triples of (x, y, idx) for each data point, instead of the standard (x, y) tuple, requiring modifications to the downstream model to accommodate this change.
- `ScalablePrunableResNet`: This class inherits from a `ScalableResNet`. In addition to the standard ResNet functionality and model scaling logic, it includes a hook that updates the `PruningDataset`'s loss information at the end of each epoch.

4.3 Model Scaling

We implement model scaling on ResNet [5] to balance training latency and accuracy. Our implementation involves a custom ResNet model that uniformly scales the width and depth of its basic blocks, following a methodology similar to EfficientNet [7]. We keep the total number of basic blocks and input resolution constant for simplicity.

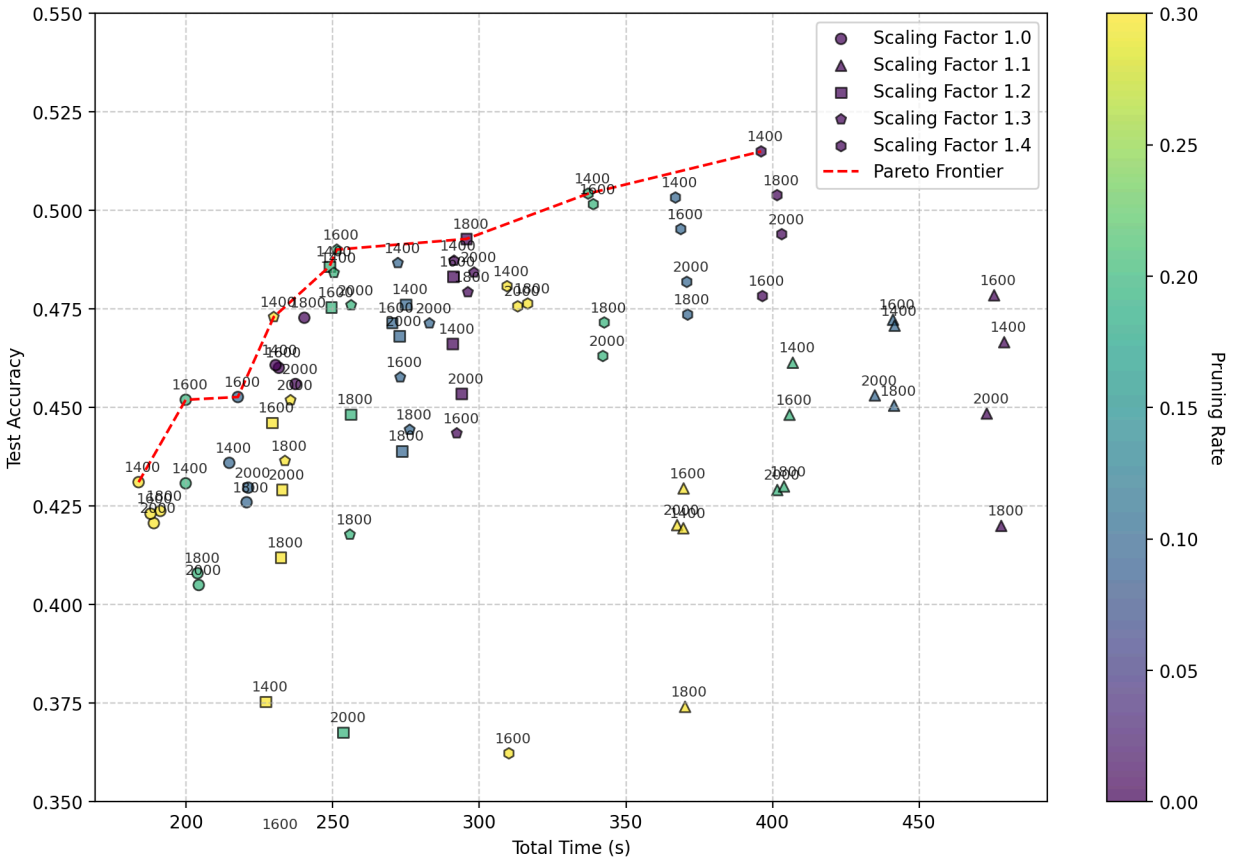

Figure 1: Full results of the experiment. The objectives (total training time, test set accuracy) form the x, y axes. The pruning rate is represented through color. The model scaling factor is represented through marker shape, with larger models having more vertices in the marker. Batch size is annotated near each scatter point.

5 Experiment

We conduct a grid search experiment to study the effect of model scaling, data pruning, and minibatch size on total training latency and downstream task accuracy.

Parameters. The independent variables have the following possible values:

- Model scaling factor: 1.0, 1.1, 1.2, 1.3. 1.4
- Target pruning rate: 0.0, 0.1, 0.2, 0.3
- Batch sizes: 1400, 1600, 1800, 2000

We choose model scaling factors that do not cause issues with large batch sizes, which we found to significantly improve latency. The pruning rate scales up to 0.3 since prior work suggested that pruning rates exceeding it tend to significantly degrade model performance.

We measure the following dependent variables:

- Total training latency, which includes data pruning overhead but does not include validation time.
- Test set task accuracy, which measures the % of images classified correctly in CIFAR100.

We keep the following factors constant:

- Each model is trained with the default learning rate of 1e-3.
- The optimizer is Adam [4] with default parameters in all configurations.
- Hardware utilization (CPU, GPU, RAM) by other processes is kept to a minimum.

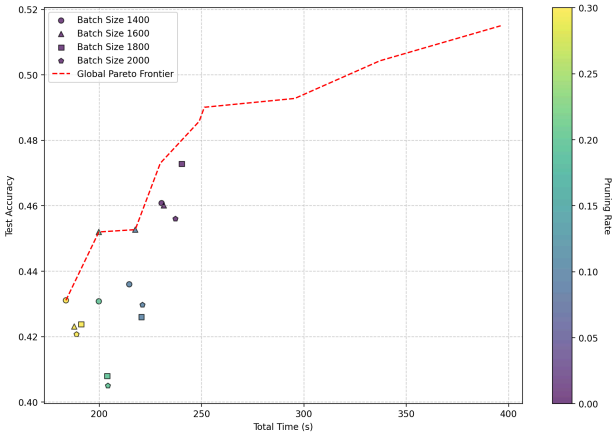

Figure 3: Scaling factor 1.0.

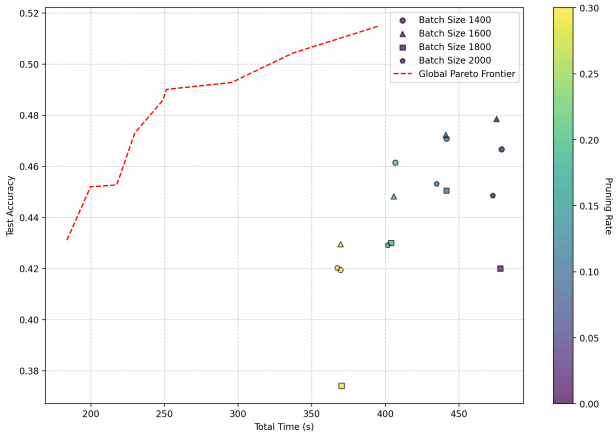

Figure 4: Scaling factor 1.1.

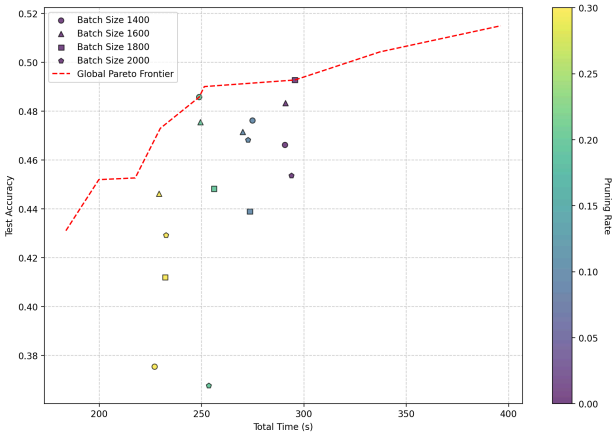

Figure 5: Scaling factor 1.2.

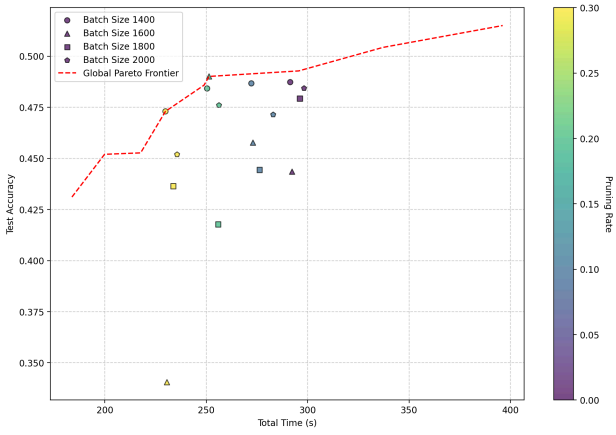

Figure 6: Scaling factor 1.3

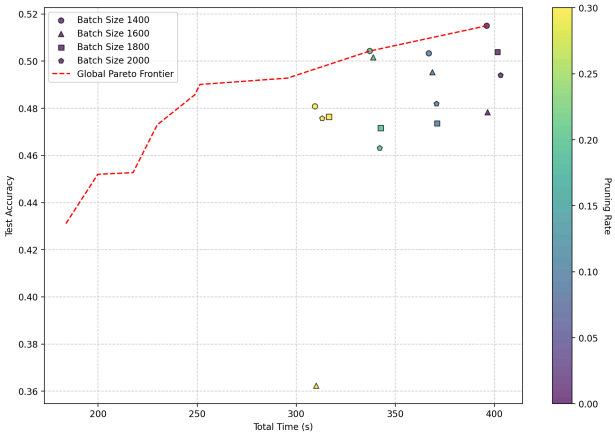

Figure 7: Scaing factor 1.4.

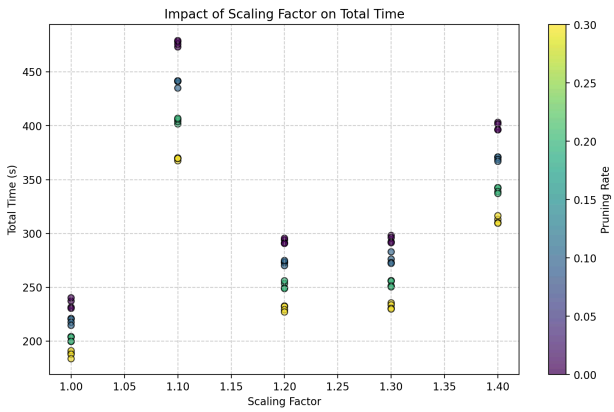

Figure 8: Scaling factor vs training time.

Figure 2: Top left to bottom left: Full results of the experiment split into subfigures based on the model scaling factor. Bottom right: Impact of scaling factor and pruning factor ion training time.

Hardware. The experiments were run on a machine with the following hardware:

- **GPU**: RTX 2080 w/ 8GB vRAM
- **CPU**: Intel(R) Xeon(R) Silver 4114 CPU
- **RAM**: 64 GB
- **OS**: Ubuntu 22.04

- **CUDA**: 12.2

6 Results

Figure 1 shows the full result of the experiment, with the two objectives plotted in the x and y axes, and the independent variables displayed using the color bar, marker shape, and annotations. The Pareto-optimal frontier of the two objectives is shown as a red line.

Figure 1 demonstrates that the Pareto-optimal frontier consists of configurations with a wide range of scaling factors and pruning rates. At the tightest training time budgets, the optimal configurations involve a small scaling factor (1.0) and a pruning rate of 0.2 to 0.3. The optimal configuration for a higher training time budget consists of the highest scaling factor (1.4) and low pruning rates of 0.0 to 0.1.

Figure 3 to Figure 7 shows the same data, split into different scaling factors. With the exception of scaling factor 1.1, the data demonstrates that the Pareto-optimal frontier for higher time budgets consists of larger scaling factors.

Figure 8 more clearly demonstrates the abnormalities in the relationship between the scaling factor and total training time. The figure also demonstrates a general positive correlation between scaling factor and training time, as well as a consistent correlation between pruning rate and training time.

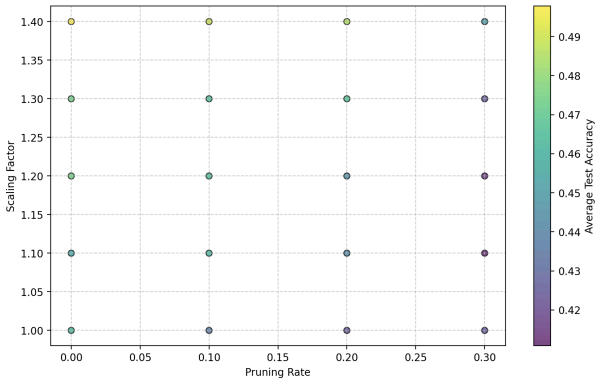

Figure 9: Impact of scaling factor and pruning rate on model accuracy.

Figure 9 displays the impact of scaling factor and pruning rate on model accuracy. While more data is required to accurately interpolate between the available data points, there is clearly a smooth and consistent correlation between the two independent variables on downstream model performance.

7 Discussion

On the 1.1 Scaling Factor Slowdown. There is an unusual slow-down in total training time for models that correspond to a 1.1 scaling factor. To diagnose this issue, we utilized Pytorch Profiler. We found that the offending models spent a considerable amount of CUDA time in the backpropagation phase of training. The 1.1 scaling factor model for a batch size of 1500 and pruning rate of 0.3 spent a total of 49.7 seconds computing gradients for convolution operations. In comparison, the 1.2 scaling factor model run with an identical batch size and pruning rate spent a total of 5.34 seconds for the same computations, despite being a larger model. The difference in backpropagation time stems from a greatly increased number of matrix multiplication kernels launched by the 1.1 scaling factor model during the backward propagation. `volta_gcgemm_32x32_tn` was launched over 400,000 times during backpropagation on the 1.1 model. In comparison, the same kernel was launched only 816 times on the 1.2 model. We could not source down the exact cause of the excessive number of kernel launches. We hypothesize that the dimensions of each convolution layer in the basic blocks do not align well.

Negative result on runtime & accuracy modeling. We had two main goals. The first goal was to model the multi-objective optimization problem (w.r.t. model performance and training latency) as a function of model parameters and data pruning parameters. The second goal was to use the model to solve an optimiza-

tion problem. The optimizer would recommend, for a given architecture and dataset, the optimal configuration for a given training time constraint. We failed to regress any model that captures an acceptable amount of information, so we omit the second half of our original goal in this report.

8 Conclusion

We investigate the Pareto frontier of the following multi-objective optimization problem: How can we achieve the optimal tradeoff between total training time latency and downstream task accuracy in ML models by adjusting the model size and data pruning rate? Through a grid search experiment, we demonstrate that the Pareo-optimal frontier involves *simultaneously* scaling down the model size and data pruning rate. This ensures that model parameters and dataset size are not the bottleneck for the other.

9 Artifact Availability

The artifact has been made publically available on GitHub through the following link:

https://github.com/
Myocardinal/DataPruningExperiment

Bibliography

[1] Zachary Ankner, Cody Blakeney, Kartik Sreenivasan, Max Marion, Matthew L Leavitt, and Mansheej Paul. 2024. Perplexed by Perplexity: Perplexity-Based Data Pruning With Small Reference Models. *arXiv preprint arXiv:2405.20541* (2024).

[2] Robert Baldock, Hartmut Maennel, and Behnam Neyshabur. 2021. Deep learning through the lens of example difficulty. *Advances in Neural Information Processing Systems* 34, (2021), 10876–10889.

[3] Nicholas Carlini, Ulfar Erlingsson, and Nicolas Papernot. 2019. Distribution density, tails, and outliers in machine learning: Metrics and applications. *arXiv preprint arXiv:1910.13427* (2019).

[4] P Kingma Diederik. 2014. Adam: A method for stochastic optimization. *(No Title)* (2014).

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016. 770–778.

[6] Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, and others. 2019. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762* (2019).

[7] Brett Koonce and Brett Koonce. 2021. EfficientNet. *Convolutional neural networks with swift for Tensorflow: image recognition and dataset categorization* (2021), 109–123.

[8] Devin Kwok, Nikhil Anand, Jonathan Frankle, Gintare Karolina Dziugaite, and David Rolnick. 2024. Dataset Difficulty and the Role of Inductive Bias. *arXiv preprint arXiv:2401.01867* (2024).

[9] Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, and others. 2022. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, 2022. 15630–15649.

[10] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Von-

drák, and Andreas Krause. 2015. Lazier than lazy greedy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

[11] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. 2020. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, 2020. 6950–6960.

[12] Dang Nguyen, Wenhan Yang, Rathul Anand, Yu Yang, and Baharan Mirzasoleiman. 2024. Mini-batch Coresets for Memory-efficient Training of Large Language Models. *arXiv preprint arXiv:2407.19580* (2024).

[13] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. 2021. Deep learning on a data diet: Finding important examples early in training. *Advances in neural information processing systems* 34, (2021), 20596–20607.

[14] Ziheng Qin, Kai Wang, Zangwei Zheng, Jianyang Gu, Xiangyu Peng, Zhaopan Xu, Daquan Zhou, Lei Shang, Baigui Sun, Xuansong Xie, and others. 2023. Infobatch: Lossless training speed up by unbiased dynamic data pruning. *arXiv preprint arXiv:2303.04947* (2023).

[15] Ravi S Raju, Kyle Daruwalla, and Mikko Lipasti. 2021. Accelerating deep learning with dynamic data pruning. *arXiv preprint arXiv:2111.12621* (2021).

[16] Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. 2022. Beyond neural scaling laws: beating power law scaling via data pruning. *Advances in Neural Information Processing Systems* 35, (2022), 19523–19536.

[17] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. 2018. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159* (2018).