FINE GRANULARITY IS CRITICAL FOR INTELLIGENT NEURAL NETWORK PRUNING

ALEX HEYMAN

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

JULY 2023

Abstract

Neural network pruning is a popular approach to reducing the computational costs of training and/or deploying a network, and aims to do so while minimizing accuracy loss. Pruning methods that remove individual weights (fine granularity) yield better ratios of accuracy to parameter count, while methods that preserve some or all of a network's structure (coarser granularity, e.g. pruning channels from a CNN) take better advantage of hardware and software optimized for dense matrix computations. We compare intelligent iterative pruning using several different criteria sampled from the literature against random pruning at initialization across multiple granularities on two different image classification architectures and tasks. We find that the advantage of intelligent pruning (with any criterion) over random pruning decreases dramatically as granularity becomes coarser. Our results suggest that, compared to coarse pruning, fine pruning combined with efficient implementation of the resulting networks is a more promising direction for improving accuracy-to-cost ratios.

Table of Contents

List of Figures

Chapter 1

Introduction

In the past decade, deep neural networks (DNNs) have become the dominant form of machine learning model and entered widespread practical use [1]. However, their computational requirements in terms of memory use, processing time, and energy consumption during training and inference alike are also greater than their predecessors and act as a bottleneck for their breadth of deployment and real-time inference capabilities [2]. For these reasons, numerous techniques have been developed to reduce the computational costs of DNN models, among the most popular of which is *pruning*. Pruning consists of algorithmically identifying parts of an existing network that are believed to be unhelpful or unimportant to its performance at its task, and removing them from the network [3, 4].

Pruning may be performed immediately after a network is initialized or early in its training, which reduces both training and inference costs, but some pruning methods operate on thoroughly trained networks and aim to reduce inference costs only [5]. A disadvantage of early pruning is that the subnetworks it leaves behind may have their response to future training disrupted in a way that lowers the pruned network's final accuracy [6]. Pruning of thoroughly trained networks is often performed iteratively in cycles of pruning and re-training [7].

Pruning may occur at a variety of different granularities. The finest granularity possible - removing individual weights, sometimes called *unstructured pruning* - contrasts with coarser-granularity *structured pruning* such as removing entire nodes from a multilayer perceptron or removing entire kernels or channels from a convolutional network. [8] Coarser pruning leaves weight matrices with their nonzero values organized into denser blocks, up to and including leaving entire rows or columns of zero values that can be removed, facilitating cost reduction in typical contemporary computing environments optimized for operations on dense matrices [4, 9]. Meanwhile, fine pruning often results in negligible cost reduction in those same environments [10, 11], though some efforts have been made recently toward accelerating sparse matrix computation [12, 13]. Previous work has found, however, that coarse pruning cannot remove as many parameters from a network as fine pruning without decreasing final accuracy [8]. Additionally, pruning channels from CNNs is known to compromise accuracy in a way that finer pruning does not, and this effect applies consistently to a variety of specific

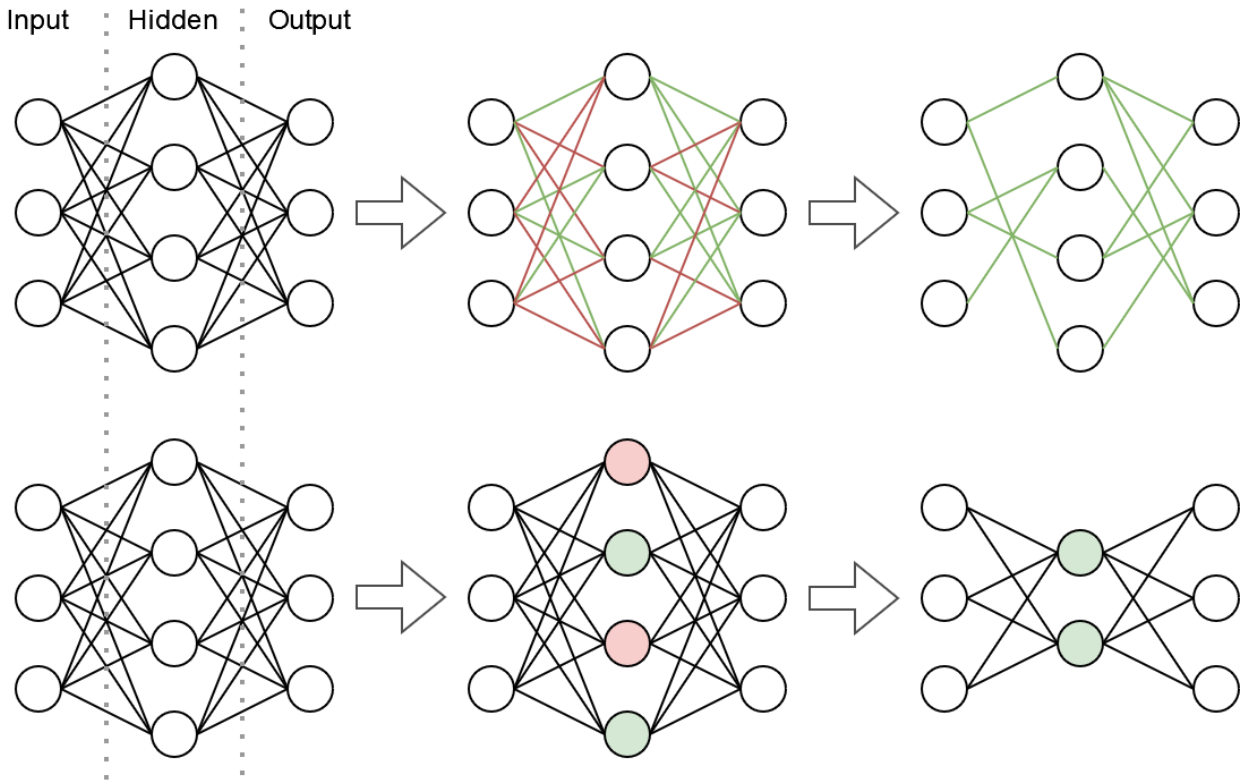


Figure 1.1: An illustration of pruning individual weights (top) vs. pruning entire hidden nodes (bottom) from a simple multiplayer perceptron. In each case here, 50% of the prunable network components are removed globally. Notice that when removing nodes, the pruning granularity is coarse enough that the network is guaranteed to remain a fully connected multilayer perceptron after pruning. For convolutional networks, possible pruning granularities include (but are not limited to) individual weights, two-dimensional kernels, and channels.

channel pruning methods proposed across the literature [2].

Frankle and Carbin [14] propose the "lottery ticket hypothesis", which holds that randomly initialized networks such as MLPs and CNNs contain subnetworks called "winning tickets" that can match the test accuracy of the original network within an equal number of training steps, despite comprising only a fraction of the original network's parameters. They provide evidence for this hypothesis with experiments in which they fully train a network, prune the fraction of its weights with the smallest magnitudes, reset the remaining weights to their original post-initialization values, and iteratively repeat this process until a subnetwork the fraction of the original's size remains. They then train this subnetwork one final time and find that - at least for some original network architectures and training schedules, and barring a very low fraction of remaining weights - the subnetwork's final test accuracy is at least that of the original network, indicating that the subnetwork was a winning ticket. Notably, when the subnetworks are randomly reinitialized, their accuracy drops, and the phenomenon in which a large fraction of weights can be removed with no decline in accuracy

at all no longer manifests. These reinitialized subnetworks outperform networks with weights pruned randomly at initialization in some architecture/task/training conditions, but not others. These results imply that much of the subnetworks' success results from the specific parameter values they were initialized with, but their graph structure also has value by itself at least sometimes.

Siswanto, Frankle, and Carbin [8] extend this work to multiple granularities by iteratively pruning network weights in "blocks" of varying dimensions, with the blocks pruned at each iteration being the ones with the lowest *mean* magnitude of the weights within them. The authors find that as block size increases, test accuracy relative to weight preservation rate decreases, and in particular the minimum preservation rate necessary to match the original network's test accuracy decreases. The differences across block sizes mostly disappear when the pruned networks are randomly reinitialized, though, meaning that as block size increases, the accuracy advantage of a pruned network over a random reinitialization of itself decreases. This implies that the specific parameter initialization is more important at finer granularity.

However, these two papers are limited as evaluations of the potential of pruning in that they only test one pruning strategy - iterative pruning based on (mean) weight magnitude - which leaves open the possibility that a different strategy might yield better ratios of accuracy to preservation rate, especially at coarser granularities. Furthermore, Siswanto, Frankle, and Carbin [8] do not compare their pruned networks' accuracies against networks pruned randomly at initialization the way Frankle and Carbin [14] do, meaning that the relationship between pruning granularity and intelligent pruning strategies' advantage over random pruning at initialization is left unexplored. This is of particular concern because, when pruning is coarse enough that it leaves a network with the same structure (e.g. a CNN with channels removed is still a CNN), random pruning at initialization is equivalent to simply initializing a narrower network to begin with, which is less computationally costly than any form of pruning.

We investigate this relationship by comparing random pruning at initialization against intelligent iterative pruning across multiple granularities. We test our intelligent pruning with multiple pruning criteria drawn from the literature at each granularity, with several criteria in total used at the granularities where they are applicable. Our results largely match those of Frankle and Carbin [14] and Siswanto, Frankle, and Carbin [8] where they overlap, but we find that there exist criteria able to outperform the mean-magnitude criterion in some coarse-granularity conditions. However, even these criteria do not come close to matching the advantage versus random pruning at initialization that intelligent *weight* pruning has, and at granularity coarse enough to fully preserve structure, all tested criteria exhibit little advantage versus random pruning. Our findings indicate that typical networks' most valuable subnetworks extend *sparsely* throughout them, and thus that fine pruning supported by efficient sparse computation holds more potential for improving accuracy-to-cost ratios than coarse pruning does.

Chapter 2

Methods

2.1 Networks and Tasks

We test pruning on two different networks, each trained on a standard image classification task:

- Resnet-20, a residual convolutional network designed by He et al. [15] for CIFAR-10, which is also the task we train it on. Resnet-20 consists of one initial convolutional layer, nine residual blocks of two convolutional layers each, and one fully connected layer at the end. We copy our hyperparameters from a condition where Frankle and Carbin [14] found that winning tickets emerge in Resnet-20. (Note that the authors refer to it as Resnet-18 in that paper, but we refer to it as Resnet-20 to align with He et al. [15].)

  Specifically, we apply Xavier normal (also called Gaussian Glorot) initialization to Resnet-20, then train it for 30,000 iterations using SGD with a momentum of 0.9 and weight decay of 10^{-4}. Learning rate begins at 0 and linearly "warms up" to 0.03 over the first 20,000 iterations, then drops to 0.003 for the next 5,000 and 0.0003 for the final 5,000. The dataset is normalized to have a mean of 0 and a variance of 1 across each color channel independently, and training input is then augmented with random horizontal flips and random translations of up to 4 pixels with empty space filled with 0 values. Batch size is 128.

- The five-hidden-layer "deep big simple neural net" designed by Ciresan et al. [16] for MNIST, which is also the task we train it on. This network, which we will call DBSN for short, is a multilayer perceptron with hidden layer sizes 2500, 2000, 1500, 1000, 500. The original paper does not specify all important hyperparameters, so we copy as much as we can from it while filling in holes and making adjustments with network accuracy and ease of implementation in mind.

  Specifically, we initialize weights randomly from a uniform distribution between -0.05 and 0.05, and initialize biases to 0. We then train for 30,000 iterations with an Adam

optimizer [17] with a learning rate of 2×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The dataset's pixel values are linearly transformed to be between -1 (black) and 1 (white). Training input is augmented with random rotations of up to 15 degrees in either direction, except when the digit is 1 or 7, in which case the random rotation is up to 7.5 degrees in either direction. This is followed by the random scaling from the original paper with $\gamma_x = \gamma_y = 20$; i.e. input images are scaled to anywhere between 80% and 120% of their original size in the horizontal and vertical dimensions independently. Batch size is 128.

2.2 Pruning Granularities

For each pruning granularity investigated for each network, we test a range of values (preservation rates) for what fraction of the prunable network components are left behind after pruning is complete. These values were chosen based on preliminary experiments testing how heavily our methods could prune our networks before inviting a high risk of catastrophic accuracy failure.

For Resnet-20, we test the following granularities:

- Weight pruning (fine granularity), implemented by forcing the values of pruned weights to 0. For consistency with the other granularities, which are only applicable to convolutional layers and thus ignore Resnet-20's fully connected final layer, this weight pruning ignores it as well. Preservation rates tested are powers of $\frac{1}{2}$ with exponents between 1 and 7 inclusive.

- Pruning of the two-dimensional convolutional kernels that each connect a single channel in one layer to a single channel in the next layer (medium granularity). Pruning is implemented by forcing all weights in pruned kernels to 0. Preservation rates tested are powers of $\frac{1}{2}$ with exponents between 1 and 6 inclusive.

- Pruning of entire channels (structure-preserving coarse granularity), implemented by forcing the activations of all nodes in pruned channels to 0. Input channels are ignored (and the output layer is ignored as well, as it is not organized into channels to begin with). Preservation rates tested are powers of $\frac{1}{\sqrt{2}}$ with exponents between 1 and 6 inclusive.

For DBSN, we test the following granularities:

- Weight pruning (fine granularity), implemented by forcing the values of pruned weights to 0. Preservation rates tested are powers of $\frac{1}{2}$ with exponents between 1 and 7 inclusive.

- Pruning of entire nodes (structure-preserving coarse granularity), implemented by forcing the activation of pruned nodes to 0. Input and output nodes are ignored. Preservation rates tested are powers of $\frac{1}{\sqrt{2}}$ with exponents between 1 and 9 inclusive.

2.3 Pruning Details and Criteria

For each of the two networks, we generate five random initializations. For each initialization, we train it once with no pruning, then once for each combination of pruning granularity, pruning criterion applicable at that granularity (or random pruning at initialization), and preservation rate tested for that granularity. Training data order is randomized individually for every run.

For random pruning at initialization, we select a random subset of the prunable components to keep from each layer independently, each subset being equal in size to the preservation rate multiplied by the total number of components in the layer. The purpose of doing this, rather than selecting a single random subset of the desired size from all components globally, is to reduce the variance of the randomly pruned networks' behavior - particularly at coarser granularities, since larger components are few in number in certain layers of our networks. In expectation, a random subset of all components will contain an equal fraction of the components in each layer, the same as our selection method. (This is not to be confused with a subset generated by repeatedly selecting a layer uniformly at random and then selecting a component within that layer uniformly at random, which in expectation will contain a larger fraction of the components from layers with fewer total components.)

The procedure for our intelligent pruning is based on the iterative pruning from Frankle and Carbin [14]. The network is fully trained, every prunable component is assigned a numerical score by the pruning criterion, the components with the globally lowest scores are pruned away, all remaining parameters are reset to their values immediately after initialization, and this is repeated for a total of five iterations before the network is fully trained one last time and the accuracy on the testing portion of the dataset is measured. The fraction of remaining prunable components pruned away at each iteration is constant, so the total fraction of remaining components decreases exponentially along the iterations down to the final preservation rate. (See Algorithm 1.) We chose this procedure - gradual pruning with resetting and full training in between - to allow our networks to gradually adjust their parameter values to their shrinking size without getting stuck in local optima that they reached by training when they were larger. Our preliminary experiments found that our networks' final test accuracies were much lower when we pruned them all at once instead of iteratively.

Pruning criteria tested are as follows:

- Magnitude pruning ala Frankle and Carbin [14]. A weight's score is equal to its absolute value. Applied to weight granularity for both Resnet-20 and DBSN.

- Mean magnitude pruning ala Siswanto, Frankle, and Carbin [8]. A component's score is equal to the mean of the absolute values of all weights within it. Applied to kernel and channel granularities for Resnet-20 and node granularity for DBSN. For channels/nodes, the weights "within" the component are defined as all of the *incoming* weights.

- The sensitivity criterion used in the at-initialization weight pruning method SNIP [7]. (The criterion itself will henceforth be referred to as just "SNIP".) Here, a weight's score

Algorithm 1 Our iterative intelligent pruning

Require: Network N with parameters P and prunable components C; score-assigning function s; preservation rate r

```
P_0 ← GenerateRandomInitialization(N)
P ← P_0
for i = 1 to 5 do
    FullyTrain(N)
    S ← s(N)                                   ▷ Assign scores to components
    d ← |C| · r^(1/5)          ▷ Calculate desired # of remaining components after this round
    O ← List of all c ∈ C sorted in increasing order of S[c]
    Prune away from N the first |C| − d elements of O
    for p ∈ P.keys do                          ▷ Reset remaining parameters
        P[p] ← P_0[p]
    end for
end for
FullyTrain(N)
```

is the average over a batch of training data of the absolute value of the derivative of the loss with respect to a virtual parameter, set equal to 1, by which the weight's value is multiplied. This is intended to approximate the effect on the loss if the weight were to be removed from the network. We apply SNIP to weight granularity for Resnet-20 and DBSN, but also extend it to the other granularities by having the virtual parameter instead multiply all of the weights in a kernel, the activation of a node, or all of the node activations in a channel. Our SNIP's batch is a random subset of the training dataset of size 2,000. (See Algorithm 2.)

- The statistical *AStd* criterion from Zhou, Liu, and Wang [18]. (The criterion will henceforth be referred to as "Stat" for short.) Here, a channel's score is determined by a combination of the average and the standard deviation of its node activations, using the formula

$$S_i = -\alpha \frac{1}{\mu_i + \epsilon} + \frac{\sigma_i}{\beta}$$

where μ_i is the average, σ_i is the standard deviation, α and β are hyperparameters, and ϵ is a small but nonzero value designed to prevent division-by-zero problems (we set it to 10^{-8}). The original paper is unclear about whether the standard deviation is taken over the activations of different nodes, over the activations of each node from different inputs, or both, and code was not provided. We decided it was most sensible to take the standard deviation of the activations of each node over the training dataset and then average those standard deviations together to calculate σ_i. Similarly, our μ_i is the mean of the activations of all of the channel's nodes across all of the training samples.

Algorithm 2 Score-assigning function for the "SNIP" criterion

Require: Network N with prunable components C; loss function $\mathcal{L}$; training dataset $\mathcal{D}$; batch size b
  $\mathcal{D}^b \leftarrow$ random subset of $\mathcal{D}$ with size b
  for $c \in C$ **do**
    $V[c] \leftarrow 1$ ▷ Create virtual parameter for c
    $S[c] \leftarrow 0$ ▷ Initialize score for c to 0
  end for
  for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}^b$ **do**
    $\hat{\mathbf{y}} \leftarrow N(\mathbf{x})$ with $(C \cdot V)$ standing in for C
    for $c \in C$ **do**
      $S[c] \leftarrow S[c] + \left|\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial V[c]}\right|$ ▷ Sum derivative magnitudes across batch samples
    end for
  end for
  for $c \in C$ **do**
    $S[c] \leftarrow S[c]/|\mathcal{D}^b|$ ▷ Divide by $|\mathcal{D}^b|$ to turn the sums into averages
  end for
  return S

(See Algorithm 3.) We set $\alpha = \beta = 1$, as we found that tended to yield the highest final test accuracy in preliminary experiments. We apply Stat to channel granularity for Resnet-20, as well as node granularity for DBSN by treating each node as the only node in its own "channel". Preliminary experiments found that Stat pruned the first layers of Resnet-20's residual blocks much more heavily than the second layers, resulting in accuracy far below that of our random pruning; thus, for Resnet-20, we force Stat to prune the same fraction of channels from the first layers collectively, the second layers collectively, and the network's single non-residual convolutional layer. (See Algorithm 4.)

- Preliminary experiments with Stat indicated that it sometimes made poor choices about which layers to prune most heavily in ways that SNIP at the same granularities did not, even aside from the residual block issue. Thus, we chose to test the "SNat" criterion: assign scores to components with Stat, but prune the lowest-scoring components on a layer-by-layer basis, with the fractions to prune per iteration at each layer chosen so that each layer would eventually end up with the same fraction of remaining components as when SNIP pruned the network globally. (See Algorithm 5.) We found that SNat outperformed Stat in some preliminary experiments, so we chose to include SNat in our final experiments.

Algorithm 3 Score-assigning function for the "Stat" criterion

Require: Network N with node-containing prunable components C; training dataset $\mathcal{D}$; hyperparameters α, β, ϵ
  for $c \in C$ **do**
    for each node $n \in c$ **do**
      $m[n] \leftarrow \text{mean}(\text{activation}(n \mid \mathbf{x}) \text{ for } (\mathbf{x}, \mathbf{y}) \in \mathcal{D})$
      $s[n] \leftarrow \text{std}(\text{activation}(n \mid \mathbf{x}) \text{ for } (\mathbf{x}, \mathbf{y}) \in \mathcal{D})$
    end for
    $\mu \leftarrow \text{mean}(m[n] \text{ for } n \in c)$
    $\sigma \leftarrow \text{mean}(s[n] \text{ for } n \in c)$
    $S[c] \leftarrow -\alpha \frac{1}{\mu+\epsilon} + \frac{\sigma}{\beta}$ ▷ Score for c
  end for
  return S

Algorithm 4 Iterative pruning with the "Stat" criterion (Resnet-20)

Require: Network N with parameters P and layers of prunable components L; preservation rate r
  $P_0 \leftarrow \text{GenerateRandomInitialization}(N)$
  $P \leftarrow P_0$
  $C_{\text{first}} \leftarrow \cup\{L[i] \mid i \geq 1 \text{ and } i \text{ is odd}\}$
  $C_{\text{second}} \leftarrow \cup\{L[i] \mid i \geq 1 \text{ and } i \text{ is even}\}$
  $G \leftarrow \{L[0], C_{\text{first}}, C_{\text{second}}\}$ ▷ Groups of components to prune with equal preservation rates
  for $i = 1$ to 5 **do**
    $\text{FullyTrain}(N)$
    for $g \in G$ **do**
      $S \leftarrow \text{AssignStatScores}(g)$ ▷ Assign scores to components
      $d \leftarrow |g| \cdot r^{1/5}$ ▷ Calculate desired # of remaining components after this round
      $O \leftarrow$ List of all $c \in g$ sorted in increasing order of $S[c]$
      Prune away from N the first $|g| - d$ elements of O
    end for
    for $p \in P.\text{keys}$ **do** ▷ Reset remaining parameters
      $P[p] \leftarrow P_0[p]$
    end for
  end for
  $\text{FullyTrain}(N)$

Algorithm 5 Iterative pruning with the "SNat" criterion

Require: Network N with parameters P and layers of prunable components L; preservation rate r

```
P_0 ← GenerateRandomInitialization(N)
P ← P_0
for ℓ ∈ L do
    r_ℓ ← FractionLeftBySNIP(ℓ, r)        ▷ From a previous run on the same initialization
end for
for i = 1 to 5 do
    FullyTrain(N)
    for ℓ ∈ L do
        S ← AssignStatScores(ℓ)                                  ▷ Assign scores to components
        d ← |ℓ| · (r_ℓ)^{1/5}    ▷ Calculate desired # of remaining components after this round
        O ← List of all c ∈ ℓ sorted in increasing order of S[c]
        Prune away from N the first |ℓ| − d elements of O
    end for
    for p ∈ P.keys do                                            ▷ Reset remaining parameters
        P[p] ← P_0[p]
    end for
end for
FullyTrain(N)
```

2.4 Secondary Experiments

The primary experiments detailed in this paper emerged from a general research interest in the properties of well-pruned subnetworks that make them especially accurate for their size. We observed in preliminary experiments that our intelligent pruning methods tended to prune some layers more heavily than others, as well as that, in at least some conditions, the subnetworks our methods left behind exhibited better final test accuracy than randomly pruned networks even when their parameters were reinitialized (as Frankle and Carbin [14] also found). Hence, we took an interest in how much of the value of the structure of these subnetworks lies simply in the quantity of components remaining in each layer.

To this end, for each intelligently pruned subnetwork generated by our primary experiments at weight granularity, we performed the following steps:

- Generate a fresh random initialization of the original unpruned network.
- Prune the fresh initialization randomly layer-by-layer, with each layer being left with the same fraction of weights that remained in the corresponding layer of the intelligently pruned subnetwork.
- Fully train the pruned fresh initialization and record the final test accuracy.

For comparison, we also repeated these steps once per intelligently pruned subnetwork with the fresh initialization copying the subnetwork's *exact* structure, ala Frankle and Carbin [14], instead of just the per-layer preservation rates.

2.5 Implementation

All of our final experiments were implemented in PyTorch and run on the Cedar cluster located at Simon Fraser University in late April and early May 2023. Each instance of a network (whether Resnet-20 or DBSN) was allocated 6 CPUs, 1 GPU, and 32,000 megabytes of memory, and 30 minutes per full training run regardless of any pruning or other computations performed before or after (which we found were quick in comparison). In total, Resnet-20 was fully trained 1740 times and DBSN was fully trained 1725 times. Based on the specs of the most powerful GPUs in the cluster (NVIDIA V100, maximum 10^{15} FLOPS), then, we can calculate a loose upper bound for the number of floating-point operations required for our final experiments, specifically 10^{15} FLOPS $\times$ 1800 seconds per training run $\times$ 3465 training runs $= 6.237 \times 10^{21}$ floating-point operations. Our preliminary experiments were typically not repeated for multiple network initializations, so their cost per tested condition was much lower; we estimate that in total they required no more than half the floating-point operations of our final experiments. Thus, our research as a whole used less than 10^{22} floating-point operations.

Code for reproducing our final experiments can be found at `https://github.com/AlexHeyman/GranularityPruning`.

Chapter 3

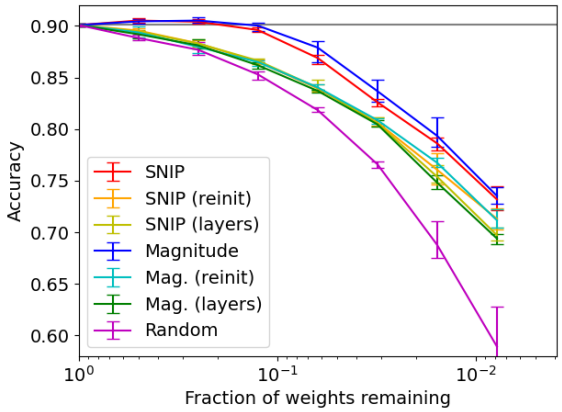
Results

3.1 Primary Experiments: Criticality of Fine Granularity for Intelligent Pruning

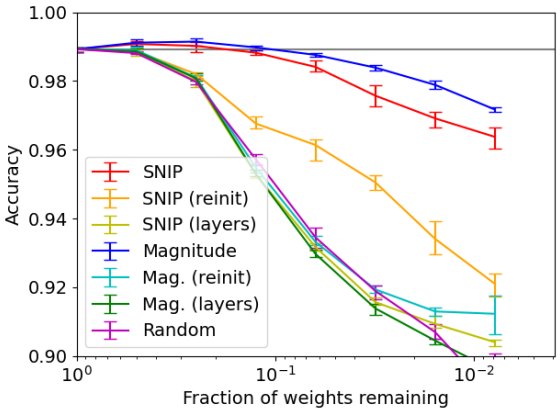
Line plots of the relationship between preservation rate and final test accuracy for each combination of network, granularity, and pruning criterion can be seen in Figure 3.1. The top of each error bar is the highest post-pruning accuracy achieved by any of the network's five initializations, the bottom of the bar is the lowest, and the lines themselves plot the means across all initializations. The horizontal gray line on each plot is the mean accuracy of the initializations when trained with no pruning. At coarse granularities combined with very low preservation rates, some of the pruned networks experience catastrophic failure, resulting in accuracies below the bottom of the plots.

Consistent with Frankle and Carbin [14], we find that magnitude-based weight pruning can remove about 87.5% of weights from Resnet-20 without significantly reducing accuracy. However, random weight pruning reduces accuracy with a mere 50% of weights removed, and its accuracy consistently remains below that of magnitude pruning, with the gap only widening as preservation rate decreases. SNIP performs slightly worse than magnitude pruning here, but is still able to prune most of the network's weights without significantly reducing accuracy. We find similar results for weight pruning on DBSN, with the intelligent-random gap widening steeply below about 25% of weights remaining.

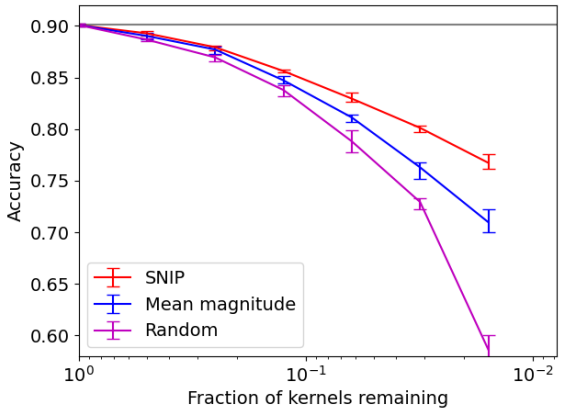
Pruning entire kernels from Resnet-20 results in reduced accuracy with only 50% of kernels removed (and thus 50% of weights removed, since all of Resnet-20's kernels are the same size), regardless of whether pruning is intelligent or random. Thus, we could not find any winning tickets at this granularity. SNIP performs better than mean-magnitude pruning here, and both have an advantage over random pruning, though that advantage is smaller than at weight granularity. The only exception is when only 1.5625% of kernels remain, and random pruning's accuracy drops sharply while the intelligent methods' accuracies do not. This is likely because our random pruning is forced to prune the same fraction of components from each layer, and Resnet-20's first layer only has 48 kernels (3 input channels $\times$ 16 output



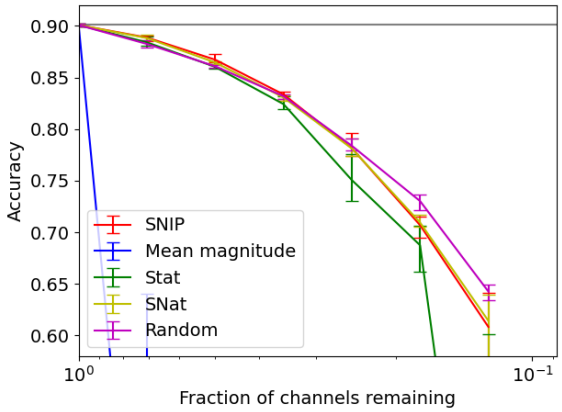
(a) Resnet-20 - Weights



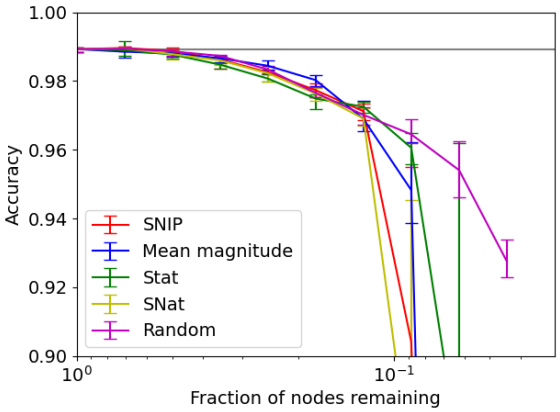
(b) DBSN - Weights



(c) Resnet-20 - Kernels



(d) Resnet-20 - Channels



(e) DBSN - Channels

Figure 3.1: Plots of the relationship between preservation rate and final test accuracy for each pruning criterion, for each network and granularity.

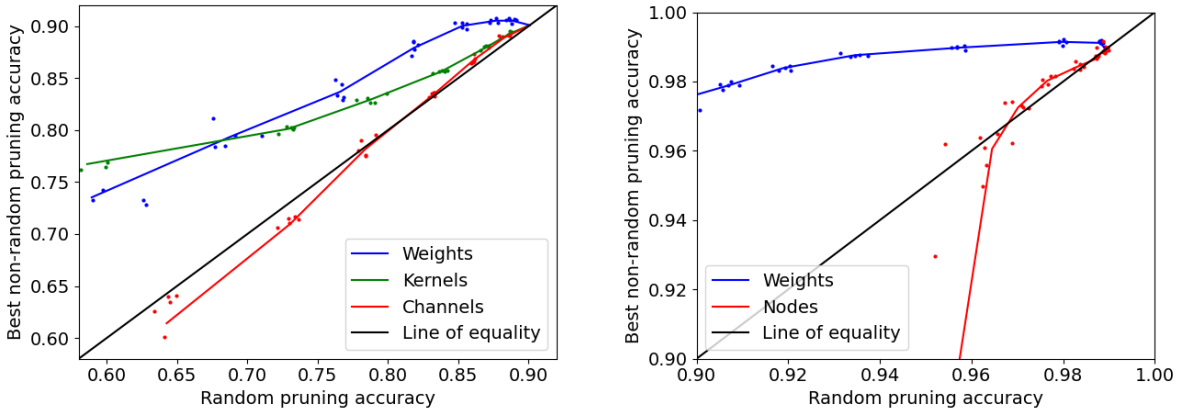
channels). $1.5625\% \times 48 = 0.75$, which our code rounds to 1 kernel remaining; thus, the information from two of the three color channels is completely lost. Intelligent pruning methods, however, are free to leave more kernels intact in the first layer and pay for it by removing more kernels in later layers where they are more numerous and expendable.

When channels are pruned from Resnet-20, accuracy once again decreases even with intelligent pruning at the highest below-100% preservation rate we test. Networks pruned with the mean magnitude criterion experience catastrophic failure even at high preservation rates; follow-up analysis indicates this is because the layer-wise depth of a channel in Resnet-20 correlates strongly with the mean magnitude of its incoming weights, and thus pruning channels based on mean magnitude results in some layers losing all or almost all of their channels. Above about 36% of channels remaining, SNIP and SNat slightly outperform random pruning, but the gap quickly becomes smaller than even the intelligent-random gap at kernel granularity, before disappearing entirely and even reversing at very low preservation rates. Stat, meanwhile, struggles to outperform random pruning from the start and soon becomes worse than it. Notably, SNat is better than Stat and very close in performance to SNIP at every preservation rate, despite having its actual pruning criterion in common with Stat and only having per-layer preservation rates in common with SNIP. This suggests that the limited success Stat and SNIP experience at high preservation rates has more to do with the number of channels they leave in each layer than anything else.

We find that about 29% of nodes can be pruned away from DBSN, intelligently or randomly, without noticeably decreasing accuracy. Accuracy decreases at lower preservation rates, though, and at all preservation rates, when an intelligent pruning method does outperform random pruning, its advantage is slim. Mean-magnitude pruning does not quickly result in catastrophic failure here, but each intelligent pruning method drops below random pruning and fails catastrophically one by one as preservation rate decreases. Follow-up analysis indicates this is because they begin over-pruning the bottleneck of the last, smallest hidden layer. Once again, SNIP and SNat remain close in performance throughout. They also outperform Stat at high preservation rates, though they fail catastrophically at higher preservation rates than it does.

In Figure 3.2, we directly compare intelligent and random pruning at different granularities for Resnet-20 and DBSN each. Each dot represents one particular initialization pruned at one particular granularity and preservation rate. A dot's X coordinate is the accuracy resulting from random pruning, and its Y coordinate is the best accuracy resulting from any of the applicable intelligent pruning criteria. The points that the lines pass through are each the mean of the random pruning accuracies across all five initializations (X coordinate) and the best mean accuracy of any intelligent pruning method across all five initializations (Y coordinate) for a particular granularity and preservation rate. As preservation rate decreases, the accuracies of the pruned networks tend to decrease, and the dots and lines thus travel down and/or to the left. Plotting the *best* accuracies achieved by any of our intelligent pruning methods allows us to evaluate the potential of all of them together, as compared to random pruning, without being biased by the limitations of any particular intelligent method.

We can see that at granularity coarse enough to fully preserve structure (channels or



(a) Resnet-20

(b) DBSN

Figure 3.2: Comparison of the relationship between final test accuracy from intelligent versus random pruning for different granularities, for each network.

nodes), the advantage of intelligent pruning is slim at best for both networks, whereas the advantage is large at weight granularity for both networks. At the intermediate granularity of pruning kernels from Resnet-20 - leaving the connectivity between pairs of channels in successive layers sparse, but the connectivity within each pair of channels as dense as it always was - the advantage of intelligent pruning is intermediate between when both forms of sparsity are permitted (weights) and when neither form is (channels). These results underscore our main conclusion that pruning has little value without fineness of granularity and thus the ability to sparsify the network.

3.2 Secondary Experiments: Roles of Parameter Initializations and Graph Structure in Well-Pruned Subnetworks

Line plots of the final test accuracies for the random reinitializations and layer-matched random subnetworks derived from our intelligently weight-pruned networks can be seen in Figure 3.1 (a) and (b), as the lines labeled with suffixes "(reinit)" and "(layers)", respectively.

Similar to Frankle and Carbin [14], we find that random reinitializations of Resnet-20 instances subject to magnitude-based weight pruning are close in accuracy to their fully randomly pruned counterparts above about 12.5% of weights remaining (though note that we observe slightly lower fully random accuracies on average than they do, which may be because our method of "fully random" pruning mandates an equal preservation rate at every layer while theirs does not). However, we find that as preservation rate drops, the reinitializations outperform random pruning to an increasing degree, to the point where they become closer in performance to the intelligently pruned networks with their original initializations. These

results also hold for the reinitializations derived from SNIP weight pruning on Resnet-20; in general, their performance is very close to that of the magnitude-based reinitializations. Notably, the layer-matched random subnetworks are much closer in performance to the reinitializations than to the fully random subnetworks, implying that at least for Resnet-20, most of the value of an intelligently weight-pruned subnetwork's graph structure is merely in the number of weights it leaves in each layer.

For DBSN, the magnitude-based random reinitializations and layer-matched random subnetworks are generally very close in performance to the fully random subnetworks, even performing slightly worse in a range of low preservation rates. This suggests that magnitude pruning is relying heavily or entirely on parameter initialization to find valuable subnetworks. The SNIP-derived layer-matched random subnetworks also perform at about fully-random level, though interestingly, the SNIP-derived reinitializations perform significantly and consistently better than random subnetworks below about 25% of weights remaining. This is the largest difference in behavior between magnitude-based and SNIP weight pruning in our results, and the reason for it is unknown to us.

Chapter 4

Discussion

We set out to investigate the relationship between the granularity and efficacy of intelligent pruning, using random pruning at initialization as a point of comparison. We found a strong relationship between granularity and the extent to which the best pruning method we tested outperformed random pruning; specifically, finer granularity resulted in a greater advantage for intelligent pruning, and this advantage was minimal at granularity coarse enough to fully preserve the original network structure (and thus take advantage of all of the same hardware optimizations as the original network). We observed these results across both of the network architectures and tasks we tested.

Our results imply that attempts to isolate exceptionally valuable subnetworks via coarse pruning have fundamentally low potential for success, because under the restrictions of coarse pruning, subnetworks are very limited in *how* exceptionally valuable they can be. The sparse subnetworks isolated by fine pruning have much more potential to improve on unpruned networks in terms of the ratio of accuracy to network size. Therefore, future research seeking to reduce networks' computational costs while preserving their performance should focus on designing new hardware and low-level software to reduce the computational costs of implementing sparse networks.

Notably, two independent lines of evidence from our results (the similar performance of SNIP and SNat as compared to Stat and SNat, and the similar performance of random reinitializations and layer-matched random subnetworks in our secondary experiments) point toward a major role for overall per-layer preservation rates in the value of exceptionally valuable subnetworks in scenarios where specific parameter initializations cannot be exploited. (Siswanto, Frankle, and Carbin [8] notes that the accuracies of coarsely pruned subnetworks do not decrease when they are reinitialized, implying that they do not take advantage of parameter initialization, just like randomly reinitialized weight-pruned networks do not.) An exception to this generalization, however, is the behavior of random reinitializations derived from low-preservation-rate weight pruning with the SNIP criterion (but not with the magnitude criterion) on DBSN (but not Resnet-20). The conditions that do and do not cause this behavior, and the phenomenon of the importance of per-layer preservation rates more generally, may merit further research for the sake of improving network architecture design.

4.1 Limitations

Our work is limited in that it only investigates two network architectures, two tasks, and a small number of pruning methods. However, the networks differ significantly from each other (MLP vs. residual CNN), and the tasks, while both image classification tasks, differ significantly in complexity. Our pruning methods are also diverse - representing measures of importance based on magnitude, gradients, and node activation statistics - and were all derived from widely cited papers. Though we are critical of coarse pruning, we devoted considerable effort in our preliminary experiments to making it work well, as reflected in our tuning and tweaking of Stat and our creation of SNat. Thus, while we cannot be completely certain that there exists no architecture, task, or pruning method that would overturn the generalizations we make here, we have confidence in them.

4.2 Societal Impacts

Research into reducing the computational costs of training and deploying networks has the potential to reduce their electricity consumption, thus making them more environmentally friendly. Networks that use less memory, processing power, and electricity can also fit on physically smaller devices and perform real-time inference more effectively, which has applications in robotics. Some of these potential applications, such as exploration or search-and-rescue robots, would be beneficial, but others, such as autonomous weapons, would be harmful. It is the responsibility of governments, as well as communities of researchers and engineers, to ensure that such technology is applied beneficially and not harmfully.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[2] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. *A Survey on Deep Neural Network Compression: Challenges, Overview, and Solutions*. 2020. arXiv: `2010.03954 [cs.LG]`.

[3] Davis Blalock et al. *What is the State of Neural Network Pruning?* 2020. arXiv: `2003.03033 [cs.LG]`.

[4] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. "Structured Pruning of Deep Convolutional Neural Networks". In: *ACM Journal on Emerging Technologies in Computing Systems* 13.3 (2017).

[5] Ekdeep Singh Lubana and Robert P. Dick. "A Gradient Flow Framework for Analyzing Network Pruning". In: *Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)* (2021).

[6] Jonathan Frankle et al. *Stabilizing the Lottery Ticket Hypothesis*. 2020. arXiv: `1903.01611 [cs.LG]`.

[7] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. "SNIP: Single-shot Network Pruning based on Connection Sensitivity". In: *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)* (2019).

[8] Arlene Siswanto, Jonathan Frankle, and Michael Carbin. *Reconciling Sparse and Structured Pruning: A Scientific Study of Block Sparsity*. Workshop paper at the 9th International Conference on Learning Representations (ICLR 2021). 2021.

[9] Chen Yang et al. "Structured Pruning of Convolutional Neural Networks via L1 Regularization". In: *IEEE Access* 7 (2019), pp. 106385–106394.

[10] Erich Elsen et al. "Fast Sparse ConvNets". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 14617–14626.

[11] Wei Wen et al. "Learning Structured Sparsity in Deep Neural Networks". In: *Advances in Neural Information Processing Systems* 29 (2016).

[12] Jeff Pool. "Accelerating Sparsity in the NVIDIA Ampere Architecture". In: *NVIDIA GPU Technology Conference* (2020).

[13] Scott Gray, Alec Radford, and Diederik P Kingma. *GPU Kernels for Block-Sparse Weights*. 2017. arXiv: `1711.09224 [cs.LG]`.

[14] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)* (2019).

[15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[16] Dan Claudiu Ciresan et al. "Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition". In: *Neural Computation* 22.12 (2010), pp. 3207–3220.

[17] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)* (2015).

[18] Yan Zhou, Guangyi Liu, and Dongli Wang. "A Hybrid Statistics-based Channel Pruning Method for Deep CNNs". In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. 2019, pp. 780–785.