Prune and Tune Ensembles: Low-Cost Ensemble Learning With Sparse Independent Subnetworks

Tim Whitaker, Darrell Whitley
Department of Computer Science
Colorado State University
Fort Collins, CO 80525
timothy.whitaker@colostate.edu, whitley@cs.colostate.edu

Abstract

Ensemble Learning is an effective method for improving generalization in machine learning. However, as state-of-the-art neural networks grow larger, the computational cost associated with training several independent networks becomes expensive. We introduce a fast, low-cost method for creating diverse ensembles of neural networks without needing to train multiple models from scratch. We do this by first training a single parent network. We then create child networks by cloning the parent and dramatically pruning the parameters of each child to create an ensemble of members with unique and diverse topologies. We then briefly train each child network for a small number of epochs, which now converge significantly faster when compared to training from scratch. We explore various ways to maximize diversity in the child networks, including the use of anti-random pruning and one-cycle tuning. This diversity enables "Prune and Tune" ensembles to achieve results that are competitive with traditional ensembles at a fraction of the training cost. We benchmark our approach against state of the art low-cost ensemble methods and display marked improvement in both accuracy and uncertainty estimation on CIFAR-10 and CIFAR-100.

Introduction

Ensemble learning has long been an active area of research in machine learning (Hansen and Salamon 1990; Krogh and Vedelsby 1994) and recently ensembles of neural networks have been used to win many high profile machine learning competitions (Vorhies 2016). Combining the predictions of several neural networks is a simple way to improve generalization on unseen data. However, as datasets and neural networks grow larger and more complex, traditional ensemble methods become prohibitively expensive to implement.

Several approaches have been introduced to reduce the large computational costs associated with building ensembles of neural networks. Methods such as pseudo-ensembles, temporal ensembles, and evolutionary ensembles either share parameters or training epochs among ensemble members in order to reduce the need to train several independent networks from scratch. While these methods reduce computational cost, they tend to be limited in ensemble size, member diversity and convergence efficiency. This is a critical problem as ensemble performance increases with the number of well trained and diverse models it contains (Bonab and Can 2016; Oshiro, Perez, and Baranauskas 2012).

We introduce an improved low-cost method for dynamically constructing an ensemble of size M. Rather than training several independent networks from scratch, we instead start by training only a single large parent network. We then clone the trained network M times and randomly prune the weights of each clone to create M significantly smaller child networks, each with a unique connective topology. Each child is then fine tuned for a small number of training epochs. We explore several methods to maximize diversity among the generated child networks, including *anti-random pruning* (Malaiya 1995; Wu et al. 2008) and *one-cycle tuning* (Smith and Topin 2018).

We call the resulting ensemble a *Prune and Tune Ensemble* (PAT Ensemble). Our approach is highly flexible and offers several unique benefits over other low-cost ensemble methods. Since ensemble members are created independently of the parent's training phase, any network architecture, pre-trained or not, can be used as a parent without modification. This allows for very large ensembles as child networks can be dynamically generated with little additional computation. Using pruning methods to create sparse children significantly reduces memory usage and computational cost with no discernible loss in accuracy (Blalock et al. 2020). Our examination of the loss landscapes of child networks show that the combination of anti-random pruning and one-cycle tuning ensures that children converge to unique local optima despite inheriting their parameters from the same network.

We compare Prune and Tune Ensembles to popular low-cost ensemble learning algorithms using several deep neural network architectures on benchmark computer vision datasets. We conduct hyperparameter ablation studies with child sparsity, pruning structures, tuning schedules, and ensemble sizes up to 128 members. We demonstrate that Prune and Tune Ensembles are both highly efficient and highly robust in low training budget (16 epochs) and high training budget (200 epochs) environments. With the experiments introduced in this paper, Prune and Tune Ensembles prove to be more accurate and more diverse than current state-of-the-art low-cost ensemble algorithms, while using significantly fewer parameters.

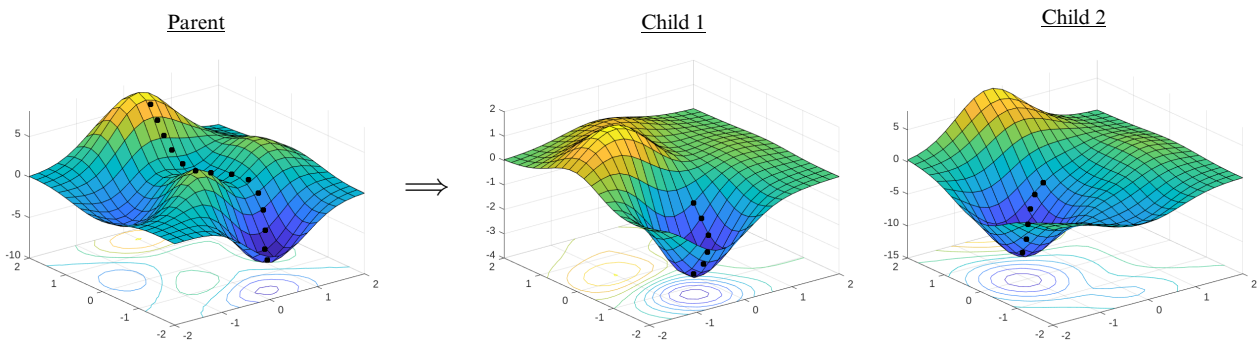

Figure 1: Prune and Tune Ensembles create diverse children via transformations that affect the underlying optimization landscapes. Each child explores a unique loss landscape and converges rapidly due to the parameters inherited from the parent.

Related Work

Several approaches have been introduced to reduce the computational cost of constructing ensembles while seeking the associated benefits in generalization. These approaches can be broadly categorized as pseudo-ensembles, temporal ensembles and evolutionary ensembles.

Pseudo-Ensembles involve the differential learning of parameters under the guise of a single monolithic architecture. Dropout is the canonical example where neurons are randomly masked for each mini batch during training (Srivastava et al. 2014). DropConnect masks connections during training instead of neurons (Wan et al. 2013) and Stochastic Depth Networks mask entire layers of deep residual networks (Huang et al. 2016). TreeNets use multiple independent branches and output heads (Lee et al. 2015). Multi-Input Multi-Output (MIMO) trains a single network on multiple samples simultaneously with distinct input and output heads (Havasi et al. 2021). BatchEnsemble decomposes weight matrices into Hadamard products of a shared weight matrix and a rank-1 matrix for each ensemble member. This allows for simultaneous training of several members on a single forward pass (Wen, Tran, and Ba 2020). HyperEnsembles vary hyperparameter configurations between members (Wenzel et al. 2021). Late Phase Weights is an approach that averages over subsets of weights learned in the late phases of training (von Oswald et al. 2021).

Pseudo-Ensembles are memory efficient, but they tend to be less diverse than true ensembles because parameters and network structure are implicitly shared (Bachman, Alsharif, and Precup 2014). This lack of diversity limits the generalization benefits that are achieved by other ensemble methods.

Temporal Ensembles train a single network and save that model's parameters at different points through time. The idea is that the model will converge to several diverse, yet accurate, local optima throughout its training cycle and the combination of those multiple optima will produce better results than the final model. Fast Committee Learning (Swann and Allinson 1998) first introduced this idea by simply choosing time slices through the training process. Recent methods seek better ways to choose which model checkpoints to save. Horizontal Voting Ensembles take the most recent states from a contiguous block of training epochs (Xie, Xu, and Chuang 2013), while Snapshot Ensembles use a cyclic learning rate schedule to alternate between converging to local minima and taking long jumps to new places in the parameter space (Huang et al. 2017). Fast Geometric Ensembles improve on Snapshot Ensembles by looking for minima along high-accuracy pathways (Garipov et al. 2018). FreeTickets saves states of a model trained with a sparse optimization algorithm (Liu et al. 2021).

Temporal Ensembles yield good results with almost no extra training time compared to a single model. However, larger ensembles have diminishing returns. Model states taken early in the training process have poorer accuracy while model states taken later in the training process tend to be highly correlated.

Evolutionary Ensembles take a single network and generate explicit child networks according to a perturbative process. These methods have long been popular in the reinforcement learning domain as an alternative to gradient based optimization for small networks. Evolution Strategies (ES, CMA-ES, NES) generate populations of neural networks by adding random noise to the parameters of a parent (Salimans et al. 2017; Hansen, Müller, and Koumoutsakos 2003; Wierstra et al. 2011). Neuroevolution of Augmenting Topologies (NEAT) starts with a population of small networks and slowly complexifies them over time, incorporating concepts like speciation to encourage diversity (Stanley and Miikkulainen 2002). MotherNets is a recent approach for deep neural networks that trains a small parent and hatches children via function preserving network morphisms (Wasay et al. 2018). Child networks are created by adding additional layers and neurons on top of the core trained parent network.

Evolutionary Ensembles produce ensembles that are more diverse than Pseudo-Ensembles and Temporal Ensembles, at the cost of slower convergence and additional computation. Evolutionary Ensembles have been mostly restricted to small network sizes, but they can generate new children cheaply, allowing for large ensemble sizes. At this point, there has not been enough work applying evolutionary ensemble methods to deep vision networks to fully evaluate their potential.

Prune and Tune Ensembles

The Parent Network: Prune and Tune Ensembles don't require any special training considerations for the parent network. Any network architecture and optimization method can be used for the parent without modification. Common regularization techniques like batch normalization, weight decay and dropout are fully compatible with our method. We suggest following best practices for training the chosen network architecture. Furthermore, our approach can use pre-trained networks to leverage even greater cost savings in practical cases.

Creating the Ensemble Members: Child networks are created by making copies of the parent network and pruning the weights according to random or complementary "anti-random" sampling methods. The topological transformations achieved with pruning allows each child network to explore an optimization space unique to its own architecture and they converge very fast due to their inherited parameters.

Modern pruning methods tend to work by deterministically selecting the best parameters to remove from a network. These approaches look at the magnitudes, importance coefficients or contributions to the gradient of the parameters (Blalock et al. 2020; Janowsky 1989; LeCun, Denker, and Solla 1990; Lee, Ajanthan, and Torr 2019; Frankle and Carbin 2019). While these methods produce very small and accurate single networks, we hypothesize that random pruning is more effective when constructing diverse ensembles.

Due to the nature of extremely high dimensional neural networks, the probability is small that we will generate two child networks with a similar topology. In addition, our ensemble creation process is decoupled from the parent network's training process, which allows us to trivially create ensembles of any size.

Random Pruning: These methods remove either connections, neurons, filters or layers randomly from neural networks. This is generally done in either a global or layer-wise fashion, which affects the distribution of pruned weights in networks that are unbalanced. In our experiments, we find little difference between global and layer-wise pruning. The amount of pruning needed to induce the child networks is dependent on the parent network's size and architecture. We compare the impact of connection and neuron pruning on sparsity in our experimental section.

Anti-Random Pruning: A random pruning process can result in children that share portions of their respective parameter space. We introduce a method to improve upon random pruning that we call Anti-Random Pruning. Inspired by the concept of Anti-Random Testing (Malaiya 1995), we attempt to generate child networks that are maximally distant from one another while equally retaining all of the original parameters from the parent.

We perform this procedure using binary bit masks that represent pruned connections. We aim to maximize the total Cartesian distance between bit masks where the Cartesian distance for two binary vectors $A = \{a_1, ..., a_N\}$ and $B = \{b_1, ..., b_N\}$ is defined as:

$$CD(A, B) = \sqrt{|a_1 - b_1| + ... + |a_N - b_N|}$$

We wish to maximize the total distance TD among all ensemble members $\{C_1, .., C_N\}$ such that:

$$TD = \sum_{i=1}^{N} \sum_{j=1}^{N} CD(C_i, C_j)$$

In practice, creating an ensemble that is maximally distant for all members would require an exhaustive search, which is infeasible for high dimensional neural networks. We suggest two alternative approaches that construct distant networks by using complementary masks and partitioning.

For any bit string M, the most distant counterpart is one in which the polarity of every bit is flipped: $M' = 1 - M$. We use this assumption to build an anti-random ensemble by first generating a random bit mask with 50% sparsity. We construct the anti-network by taking the complement of the bit mask. Each parameter of the parent model appears in only one of the two children.

The result is two child networks that are as distant from each other as possible while still inheriting parameters from the same parent. Given a parent network W_p and a binary mask generated randomly with a 50% sparsity target, $M = \{m_0, m_1, ..., m_n\}$, the resulting child networks are described as:

$$C_1 = W_p \circ M \qquad \text{and} \qquad C_2 = W_p \circ M'.$$

This can be repeated N times to create an ensemble of size $2N$. It is important to note that this approach requires 50% sparsity to create ensembles members of equal size.

We extend this idea to larger parent networks using *anti-random partitioning*. For example, for an ensemble of size N, each child network can sample $\frac{1}{N}$ parameters from the parent network, without replacement. This yields an ensemble E that is a disjoint union (denoted $\coprod$) of its member networks $E = \coprod_i E_i$. Using this approach, *every parameter in the original parent model is equally represented in the ensemble.*

Tuning The Ensemble Members: Each child network inherits its parameters from an accurate parent network. Thus, child networks converge to good solutions after only a few training epochs. Due to the differences in the topology of each child network, the resulting models converge to unique places in the optimization space.

Bagging and Boosting: In order to encourage more diversity among ensemble members, it's possible to implement traditional ensemble techniques like bagging or boosting, where each child is tuned on a different subset of the training data. In this paper we explore bagging (Breiman 1996), where a child network is trained on 90% of the available training data. The subset composed of 90% of the training data is randomly chosen for each child to promote diversity.

Traditional Fine Tuning: This method for fine tuning the members of an ensemble involves additional training using a constant learning rate equal to the last learning rate used for the parent network. We use this method of tuning for small training budgets.

One-Cycle Tuning: We use a one-cycle learning rate schedule (Smith and Topin 2018) to maximize diversity among child networks. This schedule is composed of three

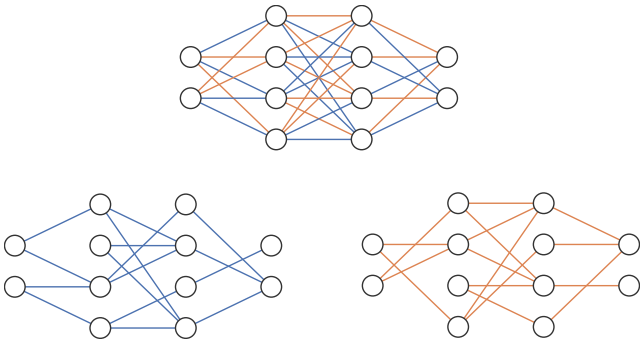

Figure 2: Parent network split into two child networks using Anti-Random Sampling. Every parameter in the parent's hidden layers appears in one of the two child networks exactly once.

phases: a warm up, a cool down, and an annihilation. The warm up phase starts with a small learning rate that ramps up to a large learning rate. The cool down phase decays from the large learning rate down to a value several times smaller than the initial learning rate. The annihilation phase then decays the rate to 0. The one cycle policy encourages diversity among child networks as the large learning rates allow each child to move a greater distance from the parent network before converging. The benefits of one-cycle tuning are more noticeable with larger training budgets (Le and Hua 2021).

We use a revised one-cycle schedule that makes use of a two-phase cosine annealing instead of the original three-phase linear schedule (Le and Hua 2021). We use a warm up phase for 10% of the training budget which quickly ramps up from the lowest learning rate (η_{min}) used in the main training phase to the highest (η_{max}); after this the learning rate decays to a very small value on the order of $1e-7$. The learning rate is updated per batch, where T_{cur} is the current iteration and T_{max} is the total number of iterations:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + cos((T_{cur}/T_{max})\pi))$$

Making Predictions: The two most popular methods for combining model predictions in ensembles include majority vote and weighted model averaging (Fragoso, Bertoli, and Louzada 2017; Džeroski and Ženko 2004; Laan, Polley, and Hubbard 2007). We use model averaging, which is standard practice for comparable ensemble methods on image classification (Huang et al. 2017; Garipov et al. 2018; Liu et al. 2021; Havasi et al. 2021).

To achieve better normalization, we average the softmax of each ensemble member's outputs to ensure that all ensemble members produce outputs of the same scale.

$$y_e = argmax(\frac{1}{S}\sum_{i=1}^{S}\sigma(f_i(x)))$$

where y_e is the ensemble prediction, S is the number of members in the ensemble (the ensemble size), σ is the softmax function and $f_i(x)$ is the output of the individual ensemble member i.

Experimental Configurations

All experiments are designed to be accessible and easily replicable with affordable hardware. Datasets and models are open source and linked in the appendix.

Datasets: We use the computer vision datasets, CIFAR-10 and CIFAR-100 (Krizhevsky 2012). CIFAR consists of 60,000 small natural colored images of 32x32 pixels in size. Those 60,000 images are split up into 50,000 training images and 10,000 testing images. CIFAR-10 samples from 10 classes of images, while CIFAR-100 samples from 100 classes of images. CIFAR-100 is more difficult than CIFAR-10 as each class will have only 500 training samples compared to 5,000 in CIFAR-10. In addition, we evaluate our large experiments on corrupted versions of CIFAR-10 and CIFAR-100 (Hendrycks and Dietterich 2019). These additional test sets are generated by adding 20 different kinds of image corruptions (gaussian noise, snow, blur, pixelation, etc.) at five different levels of severity to the original CIFAR test sets. The total number of images in each of these additional sets is 1,000,000.

Models: We use a variety of popular deep convolutional architectures, chosen both for historical significance as well as popularity and performance.

LeNet (LeCun et al. 1989): LeNet is one of the earliest successful convolutional architectures. LeNet-5 consists of two convolutional layers, two fully connected dense layers, and one output layer. We make several adjustments that are common practice in modern vision networks. These changes include the use of ReLU for the activation functions and the use of maxpooling layers after the convolutional layers. LeNet-5 contains a total of $\sim$ 140,000 parameters.

ResNet (He et al. 2016): These models were introduced in 2015 and have become a popular vision architecture in deep learning. ResNets can produce significantly deeper neural networks by utilizing residual blocks and shortcut connections between groups of layers. We use ResNet-18, an 18 layer version introduced in the original ResNet paper. This variant contains $\sim$ 11 million parameters.

DenseNet (Huang et al. 2018): These models were introduced in 2017 and build upon the ideas of shortcut connections introduced in the ResNet paper. DenseNet directly connects each layer to every other successive layer within a dense block. At each layer, the inputs are a concatenation of all previous layer outputs. We use DenseNet-121, a variant that contains 121 layers in total and is the smallest architecture introduced in the original DenseNet paper. This variant contains $\sim$ 7 million parameters.

Wide ResNet (Zagoruyko and Komodakis 2016): These models were introduced in 2016 to address the problem of diminishing feature reuse in deep residual networks. Wide ResNets increase the width of residual blocks and show excellent performance with significantly fewer layers than their ResNet counterparts. We use WRN-28-10, a 28 layer variant introduced in the original Wide ResNet paper. This model contains $\sim$ 36 million parameters.

Hardware: All models are trained on a single Nvidia GTX-1080-Ti GPU.

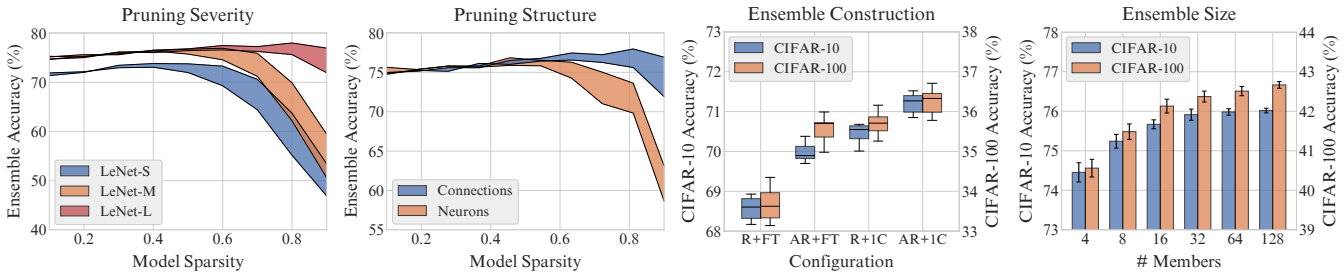

Figure 3: Hyperparameter ablations on CIFAR-10/CIFAR-100 with various LeNet configurations. The first figure reports the effect of sparsity on ensemble performance for models of different size. The second compares neuron and connection elimination for the large model. The third figure explores both random/anti-random pruning and constant/cyclic tuning schedules over 10 runs. The final figure evaluates the effect of ensemble size up to $M = 128$.

Experimental Results

The following sections detail several experiments conducted in order to evaluate the efficacy of Prune and Tune Ensembles at very different scales.

First, we perform several hyperparameter ablations and evaluate the impact of those choices on generalization using variations of the LeNet model.

Second, we compare Prune and Tune Ensembles to alternative methods on CIFAR-10 and CIFAR-100 using an extremely small training budget of only 16 total epochs.

Third, we compare Prune and Tune Ensembles to state-of-the-art low cost ensemble learning methods with a large training budget of 200 total epochs. Along with accuracy, we report uncertainty estimations on corrupted versions of the CIFAR datasets (Nado et al. 2021; Hendrycks and Dietterich 2019).

Hyperparameter Ablations

We explore the impact of several hyperaparameters on our ensembles, including: amount of sparsity, random/anti-random pruning, connection/neuron elimination, constant/cyclic tuning schedule and ensemble size. We use variations of the LeNet-5 model architecture where we vary the number of neurons in each hidden layer. We call these small, medium and large models the *LeNet-S* (253,290 parameters), *LeNet-M* (1,007,306 parameters), and *LeNet-L* versions (4,017,546 parameters). See appendix for additional details on the layer configurations.

We conduct our experiments on CIFAR-10 and CIFAR-100. We use Stochastic Gradient Descent (SGD) with Nesterov momentum for our evaluation of random/anti-random pruning and constant/cyclic tuning schedule. We use an initial learning rate of $\eta_1 = 0.1$ for 50% of the training budget which decays linearly to $\eta_2 = 0.001$ at 90% of the training budget. The learning rate is kept constant at $\eta_2 = 0.001$ for the final 10% of training. Children are tuned with either a constant learning rate of $\eta = 0.01$ for 5 epochs or with a one-cycle schedule that ramps up from $\eta_1 = 0.001$ to $\eta_2 = 0.1$ at 10% of tuning, then decaying to $\eta_3 = 1e - 7$ at the end of 5 epochs. For all other experiments, we use ADAM with a learning rate of $\eta = 0.001$.

To evaluate sparsity, we take each model variation (*LeNet-S*, *LeNet-M*, and *LeNet-L*) and train the parent network for 8 epochs. An ensemble of 8 children are created that are each tuned for 3 epochs. We report the ensemble accuracy at both 1 and 3 epochs of tuning for child sparsity ranging from 10% to 90%. All models are pruned using global connection elimination.

To evaluate neuron and connection elimination, we train the *LeNet-L* parent network for 8 epochs. An ensemble of 8 children is created, where each is pruned using either connection or neuron elimination. Each child is tuned for 3 epochs. We report ensemble accuracy for both neuron and connection elimination at various levels of sparsity between 10% and 90%.

To evaluate ensemble size, we take the *LeNet-M* network and produce up to 128 candidate member networks using 50% sparsity connection pruning and 1 epoch of constant rate tuning ($\eta = 0.001$). We evaluate ensemble performance over 10 runs as we increase the size of the ensemble from 4 to 128.

We finally evaluate random/anti-random pruning and constant/cyclic tuning. We use the *LeNet-L* network and create an ensemble of two models using each combination of random (R) or anti-random (AR) pruning and constant fine-tuning (FT) or one-cycle tuning (1C). We record results for 10 sample runs of each configuration.

Figure 3 shows the results of all ablation experiments. The leftmost figure illustrates the capacity that larger models have for learning more efficiently. Ensemble accuracy improves up to a critical region as child networks become more sparse, *without any additional epochs of training*. This indicates that more sparsity can yield greater diversity in the ensemble.

The second figure shows that connection pruning allows for more sparsity when compared to neuron pruning. We see no difference in performance for sparsity values up to 50%, but we note that the performance collapses for neuron pruning much sooner than for connection pruning.

The third figure shows that the combination of anti-random pruning and one-cycle tuning outperforms other configurations. We see significant evidence of enhanced diversity between two models with the addition of these two techniques on both CIFAR-10 and CIFAR-100.

The rightmost figure shows that ensembles consistently perform better as we increase the number of ensemble members. These results suggest significant advantages in using methods that can dynamically generate ensemble members, as performance monotonically increases with size.

Comparisons for Small Training Budgets

Ensemble learning literature has not focused on evaluating approaches for very small training budgets, but we believe there is significant value in doing so. We take inspiration from Stanford's DAWNBench experiment, the goal of which is to train a single model to a target accuracy in as little time as possible (Coleman et al. 2017). Thus, we conduct a comparison among methods that limits the total training budget to only 16 epochs. Each method is ran with a sample size of 10 runs, except for both versions of Prune and Tune, which are ran 30 times each.

The **Independent Model** is a standard neural network model trained for all 16 epochs. The **Bagged Ensemble** contains 8 full size models, each trained for two epochs. Each member is trained on a 90% sample of the full training set. The **Dropout Model** is identical to the Independent Model except dropout layers are inserted after every convolutional and linear layer. These dropout layers use the same 50% sparsity value that we use for Prune and Tune, to keep the number of parameters consistent. For **Prune and Tune**, we train a single parent network for 8 epochs, we then create 8 children using random connection pruning with a 50% sparsity target. For **Prune and Tune (AR)** we create 8 children using anti-random (AR) pruning: we create 4 pairs of network/anti-networks each with 50% sparsity. We fine tune each child for only 1 epoch. Each approach uses the ADAM optimizer with a fixed learning rate of $\eta = 0.001$.

We report the mean accuracy and standard error achieved for each method in Table 1. One may be surprised by the poor performance of Dropout. We note that the configuration of our implementation is not typical, but we chose to do this to closely match the configuration of Prune and Tune Ensemble children. This combined with the small budget leads to a much slower convergence for these models. This experiment is designed to create an initial baseline for small training budgets that can be further explored in future work.

In these small fixed training budget experiments, comparisons using a simple t-test shows that the Prune and Tune ensemble methods are significantly better than all other methods (at p=0.0001) over 30 runs.

Comparisons for Large Training Budgets

We next evaluate Prune and Tune Ensembles with a larger training budget of 200 epochs with WideResNet-28-10. We take the training configuration, ensemble size and parameter setting directly from studies of three state-of-the-art low-cost ensemble methods: MotherNets (Wasay et al. 2018), Snapshot Ensembles (Huang et al. 2017), and Fast Geometric Ensembles (Garipov et al. 2018). We also compare our results with published results of several recent low-cost ensemble methods including: TreeNets (Lee et al. 2015), BatchEnsemble (Wen, Tran, and Ba 2020), FreeTickets (Liu et al. 2021), and MIMO (Havasi et al. 2021).

Table 1: Accuracy of various methods for 16 epochs of training. Ensembles contain eight models.

Model	Method	CIFAR-10	CIFAR-100
LeNet-L	Independent	70.29 ± 0.21	34.39 ± 0.18
	Dropout	70.06 ± 0.32	28.98 ± 0.33
	Bagged	70.97 ± 0.13	32.40 ± 0.11
	PAT (R)	75.51 ± 0.06	41.68 ± 0.14
	PAT (AR)	$\mathbf{75.75 \pm 0.06}$	$\mathbf{41.85 \pm 0.13}$
ResNet-18	Independent	80.99 ± 0.11	49.26 ± 0.09
	Dropout	72.97 ± 0.41	37.69 ± 0.28
	Bagged	79.88 ± 0.05	46.18 ± 0.06
	PAT (R)	84.61 ± 0.07	57.08 ± 0.07
	PAT (AR)	$\mathbf{84.68 \pm 0.04}$	$\mathbf{57.26 \pm 0.06}$
DenseNet-121	Independent	84.14 ± 0.09	56.62 ± 0.09
	Dropout	79.59 ± 0.36	44.02 ± 0.58
	Bagged	81.16 ± 0.04	48.48 ± 0.06
	PAT (R)	86.26 ± 0.07	62.02 ± 0.07
	PAT (AR)	$\mathbf{86.35 \pm 0.04}$	$\mathbf{62.09 \pm 0.06}$

All methods compared use WideResNet-28-10 and Stochastic Gradient Descent with Nesterov momentum $\mu = 0.9$ and weight decay $\gamma = 0.0005$ (Sutskever et al. 2013). The Prune and Tune ensemble size and training schedule is as used in previous comparisons (Wasay et al. 2018; Garipov et al. 2018). We use a batch size of 128 for training and use random crop, random horizontal flip, and mean standard scaling data augmentations for all approaches (Garipov et al. 2018; Havasi et al. 2021; Liu et al. 2021; Huang et al. 2017). The parent learning rate uses a step-wise decay schedule. An initial learning rate of $\eta_1 = 0.1$ is used for 50% of the training budget which decays linearly to $\eta_2 = 0.001$ at 90% of the training budget. The learning rate is kept constant at $\eta_2 = 0.001$ for the final 10% of training.

The **Independent Model** is a single WideResNet-28-10 model trained for 200 Epochs. The **Dropout Model** enables dropout layers between convolutional layers in the residual blocks at a rate of 0.3. (Zagoruyko and Komodakis 2016). **Snapshot Ensembles** (SSE) use a cosine annealing learning rate with an initial learning rate $\eta = 0.1$ for a cycle length of 40 epochs (Huang et al. 2017). **Fast Geometric Ensembles** (FGE) use a pre-training routine for 156 epochs. A curve finding algorithm then runs for 22 epochs with a cycle length of 4, each starting from checkpoints at epoch 120 and 156. **TreeNets**, (Lee et al. 2015), **BatchEnsemble** (Wen, Tran, and Ba 2020) and **MIMO** (Havasi et al. 2021) are all trained for 250 epochs. **FreeTickets** introduces several configurations for building ensembles. We include their two best configurations for Dynamic Sparse Training (DST, M=3, S=0.8) and Efficient Dynamic Sparse Training (EDST, M=7, S=0.9).

Prune and Tune Ensembles (PAT) train a single parent network for 140 epochs. Six children are created with anti-random pruning (50% sparsity) and tuned with a one-cycle learning rate for 10 epochs. The tuning schedule starts at $\eta_1 = 0.001$, increases to $\eta_2 = 0.1$ at 1 epoch and then decays to $\eta_3 = 1e - 7$ using cosine annealing for the final 9 epochs.

Table 2: Results for ensembles of WideResNet-28-10 models on both CIFAR-10 and CIFAR-100. Methods with * denote results obtained from (Havasi et al. 2021; Liu et al. 2021). Best low-cost ensemble results are **bold**. cAcc, cNLL, and cECE correspond to corrupted test sets. We report the mean values over 10 runs for PAT.

Methods (CIFAR-10/WRN-28-10)	Acc ↑	NLL ↓	ECE ↓	cAcc ↑	cNLL ↓	cECE ↓	FLOPs ↓	Epochs ↓
Independent Model*	96.0	0.159	0.023	76.1	1.050	0.153	3.6e17	200
Monte Carlo Dropout*	95.9	0.160	0.024	68.8	1.270	0.166	1.00x	200
TreeNet (M=3)*	95.9	0.258	0.018	75.5	**0.969**	0.137	1.52x	250
SSE (M=5)	96.3	0.131	0.015	76.0	1.060	0.121	1.00x	200
FGE (M=12)	96.3	0.126	0.015	75.4	1.157	0.122	1.00x	200
PAT (M=6) (AR + 1C)	**96.48**	**0.113**	**0.005**	**76.23**	0.972	**0.081**	**0.85x**	**200**
BatchEnsemble (M=4)*	96.2	0.143	0.021	77.5	1.020	0.129	4.40x	250
MIMO (M=3)*	96.4	0.123	0.010	76.6	0.927	0.112	4.00x	250
EDST (M=7)*	96.4	0.127	0.012	76.7	0.880	0.100	0.57x	850
DST (M=3)*	96.4	0.124	0.011	77.6	0.840	0.090	1.01x	750
Dense Ensemble (M=4)*	96.6	0.114	0.010	77.9	0.810	0.087	1.00x	800

Methods (CIFAR-100/WRN-28-10)	Acc ↑	NLL ↓	ECE ↓	cAcc ↑	cNLL ↓	cECE ↓	FLOPs ↓	Epochs ↓
Independent Model*	79.8	0.875	0.086	51.4	2.700	0.239	3.6e17	200
Monte Carlo Dropout*	79.6	0.830	0.050	42.6	2.900	0.202	1.00x	200
TreeNet (M=3)*	80.8	0.777	0.047	**53.5**	**2.295**	0.176	1.52x	250
SSE (M=5)	82.1	0.661	0.040	52.2	2.595	0.145	1.00x	200
FGE (M=12)	82.3	0.653	0.038	51.7	2.638	0.137	1.00x	200
PAT (M=6) (AR + 1C)	**82.67**	**0.634**	**0.013**	52.70	2.487	**0.131**	**0.85x**	**200**
BatchEnsemble (M=4)*	81.5	0.740	0.056	54.1	2.490	0.191	4.40x	250
MIMO (M=3)*	82.0	0.690	0.022	53.7	2.284	0.129	4.00x	250
EDST (M=7)*	82.6	0.653	0.036	52.7	2.410	0.170	0.57x	850
DST (M=3)*	82.8	0.633	0.026	54.3	2.280	0.140	1.01x	750
Dense Ensemble (M=4)*	82.7	0.666	0.021	54.1	2.270	0.138	1.00x	800

There is the potential to further improve performance by exploring larger ensemble sizes and different ratios of parent to child training time.

We report the mean accuracy (Acc), negative log likelihood (NLL), and expected calibration error (ECE) over 10 runs on both CIFAR-10 and CIFAR-100 along with their corrupted variants (Nado et al. 2021; Hendrycks and Dietterich 2019). We also report the total number of floating point operations (FLOPs) and epochs used for training each method. We organize our tables into two groups based on training cost. The first group consists of low-cost training methods that take approximately as long as a single network would take to train. The second group of methods use either significantly more epochs or compute per epoch to achieve comparable performance. MIMO and BatchEnsemble both make use of batch repetition to train on more data while keeping the number of epochs low. FreeTickets (DST and EDST) use very sparse networks to keep FLOP counts low while using many more training epochs.

Prune and Tune Ensembles (using anti-random pruning and one-cycle tuning) outperform all low-cost ensemble methods and are competitive with methods that train for significantly longer. Our approach produces well calibrated, robust and diverse ensembles with excellent performance on out of distribution corrupted datasets.

Conclusions

The Prune and Tune Ensemble algorithm is flexible and enables the creation of diverse and accurate ensembles at a significantly reduced computational cost. We do this by 1) training a single large parent network, 2) creating child networks by copying the parent and significantly pruning them using random or anti-random sampling strategies, and 3) fine tuning each of the child networks for a small number of training epochs.

With the experiments introduced here, Prune and Tune Ensembles outperform low-cost ensembling approaches on benchmark image classification datasets with a variety of popular deep neural network architectures. Our approach achieves comparable accuracy to ensembles that are trained on 4-5x more data. Prune and Tune Ensembles not only improve upon the training-time/accuracy trade-off, but also reduce memory and computational cost thanks to the use of sparse child networks.

Prune and Tune Ensembles can offer a new lens through which we can analyze generalization and diversity among subcomponents of deep neural networks. We hope to further explore larger scale benchmarks, anti-random pruning methodologies and applications to new problem domains in future work.

References

Bachman, P.; Alsharif, O.; and Precup, D. 2014. Learning with Pseudo-Ensembles. *ArXiv preprint:1412.4864*.

Blalock, D.; Ortiz, J. J. G.; Frankle, J.; and Guttag, J. 2020. What is the State of Neural Network Pruning? In *Proceeding of theMachine Learning and Systems Conference*.

Bonab, H.; and Can, F. 2016. A Theoretical Framework on the Ideal Number of Classifiers for Online Ensembles in Data Streams. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2053–2056.

Breiman, L. 1996. Bagging predictors. *Machine learning*, 24(2): 123–140.

Coleman, C. A.; Narayanan, D.; Kang, D.; Zhao, T.; Zhang, J.; Nardi, L.; Bailis, P.; Olukotun, K.; Ré, C.; and Zaharia, M. 2017. DAWNBench : An End-to-End Deep Learning Benchmark and Competition. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, ML Systems Workshop*.

Džeroski, S.; and Ženko, B. 2004. Is Combining Classifiers with Stacking Better than Selecting the Best One? *Mach. Learn.*, 54(3): 255–273.

Fragoso, T. M.; Bertoli, W.; and Louzada, F. 2017. Bayesian Model Averaging: A Systematic Review and Conceptual Classification. *International Statistical Review*, 86(1): 1–28.

Frankle, J.; and Carbin, M. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. arXiv:1803.03635.

Garipov, T.; Izmailov, P.; Podoprikhin, D.; Vetrov, D.; and Wilson, A. G. 2018. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. *arXiv preprint:1802.10026*.

Hansen, L. K.; and Salamon, P. 1990. Neural Network Ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10): 993–1001.

Hansen, N.; Müller, S. D.; and Koumoutsakos, P. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation*, 11(1): 1–18.

Havasi, M.; Jenatton, R.; Fort, S.; Liu, J. Z.; Snoek, J.; Lakshminarayanan, B.; Dai, A. M.; and Tran, D. 2021. Training independent subnetworks for robust prediction. arXiv:2010.06610.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hendrycks, D.; and Dietterich, T. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. *Proceedings of the International Conference on Learning Representations*.

Huang, G.; Li, Y.; Pleiss, G.; Liu, Z.; Hopcroft, J. E.; and Weinberger, K. Q. 2017. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2018. Densely Connected Convolutional Networks. *arXiv preprint:1608.06993*.

Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; and Weinberger, K. Q. 2016. Deep Networks with Stochastic Depth. In *European Conference on Computer Vision (ECCV)*, 646–661.

Janowsky, S. A. 1989. Pruning versus clipping in neural networks. *Phys. Rev. A*, 39: 6600–6603.

Krizhevsky, A. 2012. Learning Multiple Layers of Features from Tiny Images. *Technical Report, University of Toronto*.

Krogh, A.; and Vedelsby, J. 1994. Neural Network Ensembles, Cross Validation and Active Learning. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, 231–238. Cambridge, MA, USA: MIT Press.

Laan, M.; Polley, E.; and Hubbard, A. 2007. Super Learner. *Statistical applications in genetics and molecular biology*, 6: Article25.

Le, D. H.; and Hua, B.-S. 2021. Network Pruning That Matters: A Case Study on Retraining Variants. In *International Conference on Learning Representations*.

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4): 541–551.

LeCun, Y.; Denker, J.; and Solla, S. 1990. Optimal Brain Damage. In Touretzky, D., ed., *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Lee, N.; Ajanthan, T.; and Torr, P. H. S. 2019. SNIP: Single-shot Network Pruning based on Connection Sensitivity. *arXiv preprint: 1810.02340*.

Lee, S.; Purushwalkam, S.; Cogswell, M.; Crandall, D.; and Batra, D. 2015. Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks. *arXiv preprint:1511.06314*.

Liu, S.; Chen, T.; Atashgahi, Z.; Chen, X.; Sokar, G.; Mocanu, E.; Pechenizkiy, M.; Wang, Z.; and Mocanu, D. C. 2021. FreeTickets: Accurate, Robust and Efficient Deep Ensemble by Training with Dynamic Sparsity. arXiv:2106.14568.

Malaiya, Y. 1995. Antirandom testing: getting the most out of black-box testing. In *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95*, 86 – 95. ISBN 0-8186-7131-9.

Nado, Z.; Band, N.; Collier, M.; Djolonga, J.; Dusenberry, M.; Farquhar, S.; Filos, A.; Havasi, M.; Jenatton, R.; Jerfel, G.; Liu, J.; Mariet, Z.; Nixon, J.; Padhy, S.; Ren, J.; Rudner, T.; Wen, Y.; Wenzel, F.; Murphy, K.; Sculley, D.; Lakshminarayanan, B.; Snoek, J.; Gal, Y.; and Tran, D. 2021. Uncertainty Baselines: Benchmarks for Uncertainty & Robustness in Deep Learning. *arXiv preprint arXiv:2106.04015*.

Oshiro, T. M.; Perez, P. S.; and Baranauskas, J. A. 2012. How Many Trees in a Random Forest? In Perner, P., ed., *Machine Learning and Data Mining in Pattern Recognition*, 154–168. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-31537-4.

Salimans, T.; Ho, J.; Chen, X.; Sidor, S.; and Sutskever, I. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv:1703.03864.

Smith, L. N.; and Topin, N. 2018. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958.

Stanley, K. O.; and Miikkulainen, R. 2002. Evolving Neural Networks through Augmenting Topologies. *Evol. Comput.*, 10(2): 99–127.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In Dasgupta, S.; and McAllester, D., eds., *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, 1139–1147. Atlanta, Georgia, USA: PMLR.

Swann, A.; and Allinson, N. 1998. Fast committee learning: preliminary results. *Electronics Letters*, 34: 1408–1410.

von Oswald, J.; Kobayashi, S.; Sacramento, J.; Meulemans, A.; Henning, C.; and Grewe, B. F. 2021. Neural networks with late-phase weights. arXiv:2007.12927.

Vorhies, W. 2016. Want to Win Competitions? Pay Attention to Your Ensembles. *Data Science Central Blog: URL https://www.datasciencecentral.com/profiles/blogs/want-to-win-at-kaggle-pay-attention-to-your-ensembles*.

Wan, L.; Zeiler, M.; Zhang, S.; Cun, Y. L.; and Fergus, R. 2013. Regularization of Neural Networks using DropConnect. In Dasgupta, S.; and McAllester, D., eds., *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, 1058–1066. Atlanta, Georgia, USA: PMLR.

Wasay, A.; Hentschel, B.; Liao, Y.; Chen, S.; and Idreos, S. 2018. MotherNets: Rapid Deep Ensemble Learning. *arXiv preprint:1809.04270*.

Wen, Y.; Tran, D.; and Ba, J. 2020. BatchEnsemble: An Alternative Approach to Efficient Ensemble and Lifelong Learning. arXiv:2002.06715.

Wenzel, F.; Snoek, J.; Tran, D.; and Jenatton, R. 2021. Hyperparameter Ensembles for Robustness and Uncertainty Quantification. arXiv:2006.13570.

Wierstra, D.; Schaul, T.; Glasmachers, T.; Sun, Y.; and Schmidhuber, J. 2011. Natural Evolution Strategies. arXiv:1106.4487.

Wu, S.; Sridhar, J.; Malaiya, Y.; and Jayasumana, A. 2008. Antirandom Testing: A Distance-Based Approach. *Journal of VLSI Design*.

Xie, J.; Xu, B.; and Chuang, Z. 2013. Horizontal and Vertical Ensemble with Deep Representation for Classification. *arXiv preprint: 1306.2759*.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks. *arXiv preprint: 1605.07146*.