FASTER NEURAL NET INFERENCE VIA FORESTS OF SPARSE OBLIQUE DECISION TREES

Anonymous authors
Paper under double-blind review

ABSTRACT

It is widely established that large neural nets can be considerably compressed by techniques such as pruning, quantization or low-rank factorization. We show that neural nets can be further compressed by replacing layers of it with a special type of decision forest. This consists of sparse oblique trees, trained with the Tree Alternating Optimization (TAO) algorithm, using a teacher-student approach. We find we can replace the fully-connected and some convolutional layers of standard architectures with a decision forest containing very few, shallow trees so that the prediction accuracy is preserved or improved, but the number of parameters and especially the inference time is greatly reduced. For example, replacing last 7 layers of VGG16 with a single tree reduces the inference FLOPs by $7\,440\times$ with a marginal increase in the test error, and a boosted ensemble of nine trees can match the network's performance while still reducing the FLOPs $6\,289\times$. The idea is orthogonal to other compression approaches, which can also be used on other parts of the net not being replaced by a forest.

1 INTRODUCTION

It is well known that large neural nets can be considerably compressed with little accuracy loss, which is especially important for them to be deployed in devices with limited computation. Many research papers in the last few years have investigated various model compression techniques and algorithms, and multiple industrial and academic compression software frameworks are available Zmora et al. (2019); Kozlov et al. (2020); Wu et al. (2018); Idelbayev & Carreira-Perpiñán (2020). By and large, most work has focused on compression forms such as pruning, quantization, low-rank factorization and combinations thereof.

We consider the teacher-student approach, which consists of training a new model (the student) to mimic the predictions of another, given model (the teacher). This is a generic idea dating back to at least the early 1990s Craven & Shavlik (1994) and has been reinvented in multiple settings. It can take many forms depending on the teacher and student models and the specifics of the training, and it has many applications and motivations, such as rule extraction, transfer learning or knowledge distillation Gou et al. (2021).

Our focus is on neural net compression: *we replace portions of the neural net with an oblique decision forest using teacher-student training*. Specifically:

1. *The student is a forest of oblique trees*, which have (sparse) hyperplane decision nodes, rather than the traditional axis-aligned decision nodes, which test a single feature. This is crucial and its success relies on a recently proposed algorithm to train oblique trees, *Tree Alternating Optimization (TAO)* Carreira-Perpiñán & Tavallali (2018). Why not use a traditional forest of axis-aligned trees, as in random forests or gradient boosting, which have seen widespread use (thanks in particular to highly optimized implementations such as XGBoost Chen & Guestrin (2016))? As we show, these appear unable to approach the performance of sparse oblique forests in the design space of accuracy, number of parameters and inference time.
2. *The teacher is a portion of the neural net*, in this paper the last several layers (up to the output layer). The reason is that, while oblique forests are indeed powerful models, they are generally unable to reach the accuracy of the full neural net, at least with certain types of data, such as pixel-space images—for which convolutional layers (which capture spatial invariance and

hierarchical parts-of structure) are hard to beat. (Indeed, if the forest was as good as the neural net, we would not need the latter in the first place.) However, the oblique forests are indeed able to replace significant portions of the net, and the gains in inference time and number of parameters are very considerable.

Our work touches on multiple, existing ideas that we review in section 2, and crucially relies on the Tree Alternating Optimization (TAO) algorithm, which we describe in section 3. Then, we describe our teacher-student procedure (section 4) and evaluate it in different neural net architectures (section 5).

2 Related work

Neural net compression Most current work on neural net compression can be largely categorized into pruning (weight or neuron removal), quantization (making weights take values in a small codebook), decomposition methods (approximating weight matrices as products of smaller matrices), and various combinations of those. See Lianga et al. (2021); Deng et al. (2020) for a review.

Conditional computation neural architectures There is a recent interest in having the inference for an instance use only a small part of the neural net computational graph. It has been shown that, for a given level of accuracy, conditional computation may reduce the network's runtime requirements Shazeer et al. (2017); Hazimeh et al. (2020); Ioannou et al. (2016); Veit & Belongie (2018), but these approaches have not been widely adopted as a standard part of the neural network training pipelines. One reason is the lack of principled, effective algorithms to training networks with conditional computational graphs. Note that decision trees and forests achieve conditional computation naturally.

Tree-based neural architectures Most of these can be seen as variations of the hierarchical mixtures of experts Jordan & Jacobs (1994), which are "soft" trees. That is, the input instance is propagated through every path down to every expert, with a certain positive probability, and their outputs are averaged correspondingly. An important advantage is that such architectures can be readily trained using gradient- or EM-based algorithms. However, training is very slow because each instance affects each node—there is no conditional computation—and likewise inference is very slow, which would defeat the purpose of compression. Also, "hardening" a soft tree generally increases the prediction error significantly. The main motivation for this line of work with respect to neural networks is to improve generalization by replacing the final layer by a soft decision tree Wan et al. (2021), a forest of soft trees Rota Buló & Kontschieder (2014); Kontschieder et al. (2015), or a mixture of smaller neural nets Ahmed et al. (2016). Another line of work includes building a neural network with a tree structure, so that every root to leaf path is a proper neural network Murdock et al. (2016); Tanno et al. (2019); Wang et al. (2018). To the best of our knowledge we are unaware of any works using hard trees for network compression.

Teacher-student approaches The earliest teacher-student approaches sought to extract rules from a neural net by training a decision tree to mimic it, using the predictions of the neural net for the training set or other dataset Craven & Shavlik (1994; 1996); Domingos (1998); Andrews et al. (1995). Their success was strongly limited because they tried to mimic the entire neural net, which is hard enough, and using a single, axis-aligned tree, which is generally too inaccurate. A recent version of this is Frosst & Hinton (2017), who use a soft tree, but this defeats the purpose of fast inference, since the input instance must travel through each path of the tree.

The teacher-student approach has also been used explicitly for compression, often by having the teacher and the student be both neural nets of the same architecture, but forcing the student to learn the teacher's outputs with some compression constraint such as quantization Polino et al. (2018) or pruning Aflalo et al. (2020).

Pretraining and transfer learning A common paradigm for transfer learning is to extract features from early layers of a neural net ("pretrained" features) in the hope that they are transferable to another task. Then, one fine-tunes a new classifier (which could be a forest) on those features. This effectively results in a hybrid architecture consisting of some neural layers and a forest classifier. However, this does not consitute compression of the original neural net for the original task.

Decision trees Although different types of decision trees have been proposed since the 1950s, the form that has become established consists of axis-aligned trees, where a decision node tests a single

input feature vs a threshold, and a leaf outputs a prediction (say, class label). Likewise, most decision forests use that type of trees with a bagging approach, such as random forests Breiman (2001), or boosting, such as AdaBoost Freund & Schapire (1997) or gradient boosting Friedman (2001). They are among the most powerful machine learning models, and very efficient implementations have made them widely used in applications Chen & Guestrin (2016). In all cases, each individual tree is learned using a recursive partitioning approach that grows the tree top-down one node at a time, greedily fixing its parameters (feature and threshold) based on a heuristic that estimates a good, "pure" split (such as the Gini index in CART Breiman et al. (1984) or the entropy in C4.5 Quinlan (1993)). This is optionally followed by a pruning step that helps to reduce overfitting. Oblique trees, having a hyperplane decision, are obviously more powerful, but have proven harder to learn than axis-aligned trees Hastie et al. (2009). The TAO algorithm Carreira-Perpiñán & Tavallali (2018), described below, is able to learn oblique trees with a much better accuracy, and this has been shown to produce more accurate forests as well Zharmagambetov & Carreira-Perpiñán (2020) (although these works did not use them for neural net compression).

3 ALTERNATING OPTIMIZATION TO LEARN SPARSE OBLIQUE TREES

The TAO algorithm was originally proposed in Carreira-Perpiñán & Tavallali (2018) to learn sparse oblique trees with constant-label leaves, and later generalized to handle forests for classification and regression Zharmagambetov & Carreira-Perpiñán (2020). We describe the form of the algorithm we use and refer the reader to those papers for further details. Consider a decision tree with decision nodes in set $\mathcal{D}$ and leaves in set $\mathcal{L}$. Each decision node $i \in \mathcal{D}$ sends an input instance $\mathbf{x}$ to its right child if $\boldsymbol{\theta}_i^T \mathbf{x} + \theta_{i0} \geq 0$ and to its left child otherwise (hyperplane, or oblique, decision nodes). For classification, each leaf $i \in \mathcal{L}$ outputs a single class $\theta_i \in \{1, \dots, K\}$ (constant-label leaves). The tree predictive function $T(\mathbf{x}; \boldsymbol{\Theta})$ (with nodes' parameters written jointly as $\boldsymbol{\Theta}$) routes $\mathbf{x}$ to a leaf which outputs its class. TAO works in a way that is very different from recursive partitioning (as in CART or C4.5), and much closer to how most other machine learning models (such as neural nets) are learned. TAO iteratively optimizes the regularized empirical risk of a parametric model, jointly over all the nodes' parameters:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(y_n, T(\mathbf{x}_n; \boldsymbol{\Theta})) + \lambda \sum_{i \in \mathcal{D}} \|\boldsymbol{\theta}_i\|_1 \tag{1}$$

where $L(\cdot, \cdot)$ is a loss function (such as the 0/1 loss) and $\lambda \geq 0$ a regularization hyperparameter. That is, unlike with the traditional algorithms, with TAO we can choose a specific loss and regularization term. One first fixes the model structure (say, a complete tree of depth Δ with random node parameters) just as one would with a neural net. Then, each iteration updates the tree parameters (here, the decision node hyperplanes and leaf classes) so E decreases monotonically. Where in a neural net one would use a gradient-based algorithm, with trees TAO uses alternating optimization over sets of non-descendant nodes. Why this works is due to two theorems. Define the *reduced set* $\mathcal{R}_i \subset \{1, \dots, N\}$ *of decision node or leaf* i as the training instances that reach i under the current tree. Then we have (see Carreira-Perpiñán & Tavallali (2018) for proofs):

Separability condition If nodes i and j are not descendants of each other (e.g. all nodes at the same depth), then E can be written equivalently as a separable function of the parameters of i and j. (This follows from the fact that $\mathcal{R}_i \cap \mathcal{R}_j = \varnothing$, because the tree makes hard, not soft, decisions, by sending an instance down exactly one child at each decision node.)

Reduced problem The problem of optimizing E over the parameters $\boldsymbol{\theta}_i$ of node i simplifies as follows. For a leaf i, it is equivalent to training its model parameter θ_i on $\mathcal{R}_i$ to optimize E. In our case, θ_i is simply the majority class on $\mathcal{R}_i$.

For a decision node i, it is equivalent to a weighted 0/1 loss binary classification problem over the node's decision function on its reduced set. Each instance is "pseudolabeled" as the child (left or right) that leads to a lower value of E under the current tree. (This follows from the fact that all a decision node can do with an instance is send it down its left or right child, and the ideal choice is the one that results in the best prediction downstream from that node.) The regularization term $\|\boldsymbol{\theta}_i\|_1$ carries over to the reduced problem. While optimizing the 0/1 loss in general is NP-hard, we can approximate it by a surrogate loss; we use ℓ_1-regularized logistic regression and solve it with LIBLINEAR Fan et al. (2008).

The separability condition allows us to optimize E in parallel over the parameters of any set of nodes that are not descendants of each other (fixing the parameters of the remaining nodes). We process the

nodes in reverse breadth-first search order. The reduced problem theorem implies that optimizing E over a decision node's parameters reduces to optimizing a binary linear classifier with the 0/1 loss with certain pseudolabels. Overall, then, what TAO does is simple: at each iteration, it trains a binary linear classifier at each decision node and a majority-vote K-class classifier at each leaf. This monotonically decreases the objective function E. Note that after each iteration the reduced set on which each classifier or leaf predictor is trained changes.

Finally, if one uses an ℓ_1 regularization term (as in eq. (1)), each node's weight vector will be sparse, or even entirely zero, which makes the node redundant (since it directs all input instances to the same child) and can be removed in a postprocessing step at the end of the training. This means that TAO also learns the tree structure (subject to being a subset of the initial tree's structure). The size of the tree (and the sparsity of the node weight vectors) is controlled by λ: larger values result in smaller trees, eventually a single-leaf tree. This type of trees were called *sparse oblique trees* in Carreira-Perpiñán & Tavallali (2018) and can be seen as a tree form of a Lasso.

3.1 Sparse oblique forest: construction and hyperparameters

A sparse oblique forest can be naturally learned via any standard ensembling mechanism, such as bagging or boosting, but where each individual tree is learned with TAO. Here we consider bagging Breiman (1996), where each tree is trained independently on a bootstrap sample; and SAMME Zhu et al. (2009), a version of AdaBoost, where trees are trained sequentially on the entire, but adaptively reweighted, dataset.

A sparse oblique forest is controlled by 3 hyperparameters: Δ, the depth of the initial, complete tree; λ, the sparsity hyperparameter (the same for all the trees in the forest); and T, the number of trees. The structure and size of the learned tree is determined by λ, so Δ should simply be taken large enough so it does not interfere. (Note that Zharmagambetov & Carreira-Perpiñán (2020) used a fixed λ value and varied Δ and T instead.)

To find a forest that strikes a good tradeoff between accuracy and compression, we want to generate a collection of forests, each for a choice (λ_i, T_i) where $\lambda_i \in \{\lambda_1, \lambda_2, \dots, \lambda_Q\}$ and $T_i \in \{1, \dots, T\}$. To learn them it is useful to use warm-start, i.e., to compute the regularization path starting from λ_1 and ending in λ_Q so that the tree for λ_i is initialized from that of λ_{i-1}. This has the advantage over using an arbitrary initial tree with random parameters that training is faster (the tree for λ_{i-1} should be close to that of λ_i) and the result is smoother (we reduce erratic convergence to disparate local optima). For bagging, repeating this independently (i.e., we run T regularization paths) for each $T_i \in \{1, \dots, T\}$ gives the Q forests. For boosting, we run a single regularization path, which gives the first tree for each forest; then, the remaining trees in each forest are learnt by boosting as usual.

4 Teacher-student compression of neural net with forests

In our setting, the teacher model is the part of the network that is being replaced, and the student model is a forest of oblique decision trees trained using the TAO algorithm (sec. 3). Formally, let us denote the input to the network as $\mathbf{x}$, the corresponding true label as $\mathbf{y}$, and the output after the kth layer as $\mathbf{f}_k(\mathbf{x})$ with $k = 1, \dots, K$. Using such a definition, the output of the entire network is $\mathbf{f}_K(\mathbf{x})$, and typically the teacher model is trained to have $\mathbf{f}_K(\mathbf{x}) \approx \mathbf{y}$ by minimizing some loss function (e.g., cross entropy in case of multi class classification). Now, assume we want to replace all K' to K layers using a forest $\mathcal{F}$: that is, the output of the preceding layer, $\mathbf{f}_{K'-1}(\mathbf{x})$, will be fed as an input to our forest $\mathcal{F}$ and the forest's prediction $\mathcal{F}(\mathbf{f}_{K'-1}(\mathbf{x}))$ will be used instead of the network's output $\mathbf{f}_K(\mathbf{x})$. In the knowledge distillation version of the teacher-student approach, the student model is trained to match the teaching model's output (i.e., $\mathbf{f}_K(\mathbf{x})$), however, we use the true label $\mathbf{y}$ as the teaching signal. See an illustration in Fig. 6 of appendix. To train our forests we collect the input/target pairs $(\mathbf{f}_{K-1}(\mathbf{x}), \mathbf{y})$ computed on the training data.

The compression of the replaced part of the network is achieved in two ways. First, since we are using hard decision trees, the prediction through a tree requires computing inference along a single root to leaf path of depth Δ (a balanced tree), which involves Δ vector-vector products, while prediction through a neural network requires computing $K - K' + 1$ matrix-vector products of various sizes. While it is still possible to design a large tree that requires more computation than the replaced part of the neural network, we experimentally find that this is not the case: a relatively small and shallow trees can approximate large chunks of networks well. A second factor contributing to

the compression is the sparsity of the decision nodes controlled by a hyperparameter λ, see eq. (1). With sparse decision nodes, each evaluation becomes much faster and requires fewer parameters (for a sufficiently sparse weights). The sandwiching of sparse decision trees and forest on top of the neural network can lead to a further compression as sparse decision tree will act as feature selector. The inputs that are not used by a tree, which correspond to the output neurons of the feeding layer, can be safely removed.

5 EXPERIMENTS

We quantify how well sparse oblique forests and axis-aligned forests are able to replace specific portions of several well-known deep net architectures. In summary, we find that oblique forests (with suitable hyperparameter choices) can comfortably retain the accuracy of all the fully-connected layers and even some convolutional layers while speeding up the inference very considerably, and also using fewer parameters. Random forests and XGBoost forests are not able to replace so many layers, and are also much slower. Next, we describe how we measure the number of parameters and inference time, and describe our experimental results. We relegate details of the training to the appendix B. In this section, and throughout the text, unless specifically noted, *we report the compression ratios wrt replaced parts of the neural networks*.

Number of parameters We define the number of parameters as the total count of nonzero weights in the model. We opt to report the number of parameters instead of the required storage bits when saved to disk because: first, the actual disk storage will depend on the chosen format for the sparse weights, and it is often possible to manipulate the final storage size by simply changing the format or by chaining it with other compression forms (which constitutes a large research field on its own); second, the number of nonzero parameters can be compared directly across different models and papers, while the storage size depends on many (hidden) factors which will hinder future comparisons.

Inference time and FLOPs We report an estimate of floating-point operations (FLOPs) and the required inference times through the forests we have trained. Unlike the neural networks, where given a constant sized input, the FLOPs count remain the same, the inference FLOPs through a decision tree is instance conditioned and, depending on the taken branch (and its sparsity), might differ across the examples. Additionally, we find that the FLOPs model is a poor approximation for the inference speed of decision trees.

We define a FLOPs count as the average number of fused additions and multiplications *involving nonzero weights* as counted during the inference of a single training point and then averaged across the dataset. We use fused FLOPs count to be consistent with neural network compression literature: for example, $1 \times 2 + 3$ will be reported as a single FLOP (instead of two FLOPs).

The inference time is defined in a similar to FLOPs manner: it is the average number of seconds required to compute a prediction for a single input (averaged over training dataset). To measure the inference times we execute all models (including neural nets) in a single core (single thread) on Intel Xeon CPU (E5-2699v3) clocked at 2.30GHz. While simple in definition, in practice, the measurements are complicated due to the quality of the implementations. For example, neural networks enjoy highly optimized execution due to structured matrix-vector products (which induce fewer number of cache misses and easier to vectorize), and have efficient back-ends (we use ONNX Runtime in our measurements). On the other hand, while XGBoost library is highly optimized for training and inference of decision trees, we find that single input inference exhibits unexpectedly large inference times for forests of small sizes (e.g., $T = 1$) when compared to the inference times of larger XGBoost forests (e.g., $T = 100$). At the same time, the random forest implementation through scikit-learn was the slowest, possibly due to being written in python.

To alleviate the implementation effects of XGBoost and random forests, we used TreeLite library Cho & Li (2018) that optimize the decision tree's computational graph and result in very fast trees by compiling them into annotated C code that provides hints for compiler optimization. Unfortunately, TreeLite does not support oblique trees, which puts our TAO forests at a disadvantage: while our TAO tree inference is reasonably implemented, we strongly believe that it would benefit from additional optimization. For these reasons, we expect our inference times will vary depending on the system. However, since our oblique trees have most room for additional system optimizations, we would expect their performance gap over the other models to be even larger. Finally, we did not consider parallel processing; all models here will profit from it, again depending on the system.

Table 1: Compression of the softmax layer of VGG16 trained on CIFAR10. Inference time measured using default implementation (inf.) and using TreeLite (inf.$^+$)

Model	test error, %	params	FLOPs	inf. (ms)	inf.$^+$ (ms)
Reference softmax	6.46	5130	5130	0.01249	—
TAO, $T = 1$, $\lambda = 0.01$	**6.68**	3070	1305	**0.00156**	—
CART, $T = 1$	7.37	**19**	**6**	0.07140	0.01973
XGB, $T = 1 \times 10$	7.75	40	10	0.41064	0.02132

Gradient boosting is slow with multiclass problems First, we give some intuition why gradient boosting (GB) forests are expected to be large. Boosting works by greedily learning T learners. With K-class classification, each learner can be a single K-class tree with AdaBoost, but with GB it is a forest of K trees each outputting a scalar. Hence, T learners are T trees for AdaBoost (or for Random Forests, or for our bagged or boosted oblique trees) but a staggering TK trees for XGBoost.

Now consider replacing a K-class softmax layer having D input features. Its computation can be seen as K scalar products (of dimension D), to compute each class score. The XGBoost computation can be seen as K independent computations too, each a T-tree forest. Although the actual inference time depends on specifics (in particular the dimension D), this seems much heavier than one scalar product, however well optimized XGBoost is.

We evaluated this in compression of softmax layer of VGG16 trained on CIFAR10, which has a test error of 6.46% and 5130 softmax weights. We report our compression results in Table 1. Our single TAO tree achieves a marginally higher test error of 6.68% and has fastest inference time. Notice that FLOPs and parameters counts for XGBoost and random forest (RF) are much smaller than for a TAO tree, yet, it does not result in a faster inference even when using with TreeLite compilation (reported as inf.$^+$). In particular, even a single CART tree needs more time to finish the inference than softmax itself, whereas our single tree is 8$\times$ faster.

How many layers can we replace with a forest without accuracy loss? An important question we need to ask is how many layers we can replace with a forest without an accuracy loss. This is an empirical question and its answer will vary from case to case. To get an intuition, we compress progressively larger parts of 16 layer VGG network (13 convolutional and 3 fully-connected layers) trained on CIFAR10, starting from fully-connected layer 1 (i.e., FC1 $\rightarrow$ output). We then run a similar experiment using random forest compression and report the achieved test errors in Fig.1.

We observe that using TAO trees we can replace parts of the network up to 10th convolutional layer (conv10 $\rightarrow$ output) without any loss in predictive performance. With random forest compression, the accuracy degradation is quite significant and starts earlier than with TAO trees.

How much faster is a forest than the original neural net portion? Equipped with a strong empirical evidence that large portions of the neural networks can be precisely approximated by forests of decision trees, we now task ourselves with compressing some parts of the neural networks and analyze it from the compression perspective.

In the first experiment, we replace all (two) fully-connected layers of LeNet5 trained on MNIST. The original network has 431K parameters, 2.2M FLOPs, runs in 0.18 ms, and has the test error of 0.55%. The replaced part has 405K parameters, 405K FLOPs, and runs in 0.076 ms. The compression-error tradeoff curves for this experiment are given in Fig. 2 We observe that bagged and boosted TAO forests allow us to significantly reduce the required FLOPs and storage: with bagged forest of $T = 11$ trees ($\lambda = 10$) we obtain a replacement model with the test accuracy of 0.57%, 5841 FLOPs ($\downarrow 69\times$) and 118K params ($\downarrow 3.41\times$), which runs in 0.019 ms; similarly, using the boosted ensemble of $T = 5$ trees ($\lambda = 0.01$) we can compress the fully-connected layers to a model having 19K FLOPs ($\downarrow 21\times$), 110K parameters ($\downarrow 4\times$), and the inference time of 0.020 ms, which has a better test error of 0.52%.

In the second experiment, we replace the last seven (four convolutional and three fully-connected) layers of VGG16. The original reference model has 15.2M parameters, 313.7M FLOPs, and runs in 13.03 ms, and the test error of 6.46%. The replaced part has 9.9M parameters 66.5M FLOPs, and runs in 4.42 ms. For this network, we observe even higher compression ratios. For instance, an

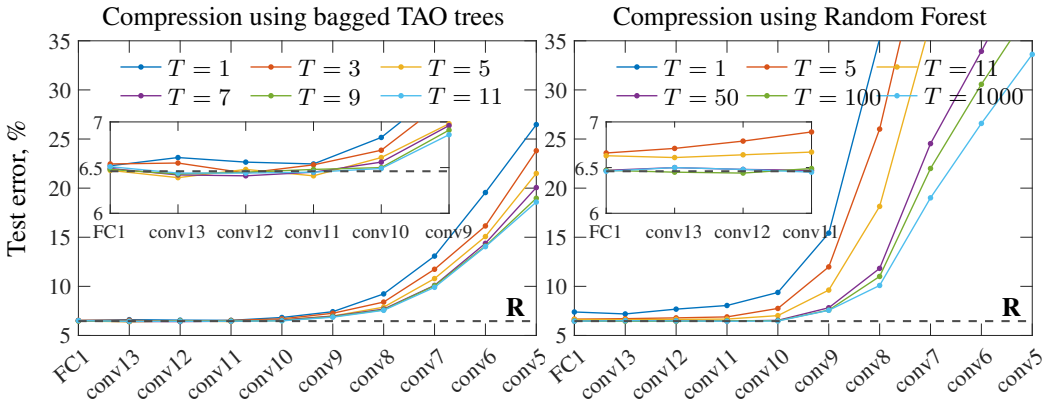

Figure 1: Test error as a function of layers when tree-based compression (TAO bagging and random forest) is applied to progressively larger parts of the network. For every layer X (FC1, conv13, etc.) we report the test accuracy when we replace the X $\rightarrow$ output part of the network with a forest of size T. Test error of the reference network is indicated with a horizontal dashed line labeled with **R**.

ensemble of $T = 9$ bagged trees ($\lambda = 0.01$) has the same error as the reference net (6.46%) but needs only 1.04M parameters ($\downarrow 9.6\times$), 0.13 MFLOPs ($\downarrow 516\times$), and runs in 0.16 ms ($\downarrow 28\times$). An ensemble of $T = 9$ boosted trees ($\lambda = 1$) has the test error of 6.46% with 10.5K FLOPs ($\downarrow 6289\times$) and 20K parameters ($\downarrow 488\times$), and runs in 0.052 ms ($\downarrow 86\times$)

In Figures 3 and 5 we put selected results in comparison to the baselines. Our TAO forests are orders of magnitudes faster than regular XGBoost and random forests, and achieve much smaller test errors. Our advantage in inference-error tradeoff space remains still even when we compare it to TreeLite optimized versions (dashed colored lines) of achieved XGBoost and random forests. Additional experiments on ResNets and VGG16 networks are available in appendices D, E and F.

Training time of a sparse oblique forest A major advantage of our compression approach is that we do not need to train any neural networks: we just need to train a forest on the selected neural net outputs. The TAO training itself, is parallelized on the level of decision node problems (see sec. 3), and on the level of trees in case of bagging. Additionally, our node-problem solver, LIBLINEAR, can utilize the sparsity in the input features to speed up the training; this comes handy because majority of the activations in the neural networks are ReLUs, which produce sparse outputs. Overall, using 16 cores of Intel Xeon E5-2699 clocked at 2.30 GHz, our longest TAO bagging experiment (VGG16, conv8 $\rightarrow$ output) with $T = 15$ finishes in 24 minutes, and the longest TAO boosting experiment (on the same dataset) finishes in 95 minutes. In average, any reported experiments finish within 40 minutes.

Combination with other compression techniques Our compression mechanism is orthogonal to other techniques like pruning, quantization, and low-rank compression; and in combination with those, further compression can be achieved. To demonstrate such a capability, we obtain low-rank compressed VGG16 model from Idelbayev & Carreira-Perpiñán (2021) trained on CIFAR10. This network has a test error of 6.49%, 1.96M parameters, 61.31M FLOPs, and runs in 2.20 ms; with respect to the reference VGG16, this numbers correspond to 7.74$\times$ reduction in parameters, 5.10$\times$ reduction in FLOPs, and 5.91$\times$ inference speed-up. We replace the last six layers of this low-rank network with a bagged forest of $T = 15$ trees and achieve a further compressed low-rank+tree model with a better test error of 6.44%, fewer parameters count of 1.55M ($\downarrow 9.81\times$) and fewer FLOPs count of 58.85M ($\downarrow 5.31\times$), which runs in 2.13($\downarrow 6.11\times$).

Our obtained low-rank + tree model not only improves over original low-rank model but also obtains a better tradeoff than filter pruning methods like HRank Lin et al. (2020) (which achieves 8.77% test error at 73.70M FLOPs and 1.78M parameters).

6 Discussion

Although we have motivated our work as neural net compression, effectively we learn a hybrid architecture consisting of the composition of neural and forest layers. Why not train the whole model jointly over all its parameters? At present, we do not have a reliable way to do this, because a decision tree is a nondifferentiable function. Also, separate training does have an important advantage:

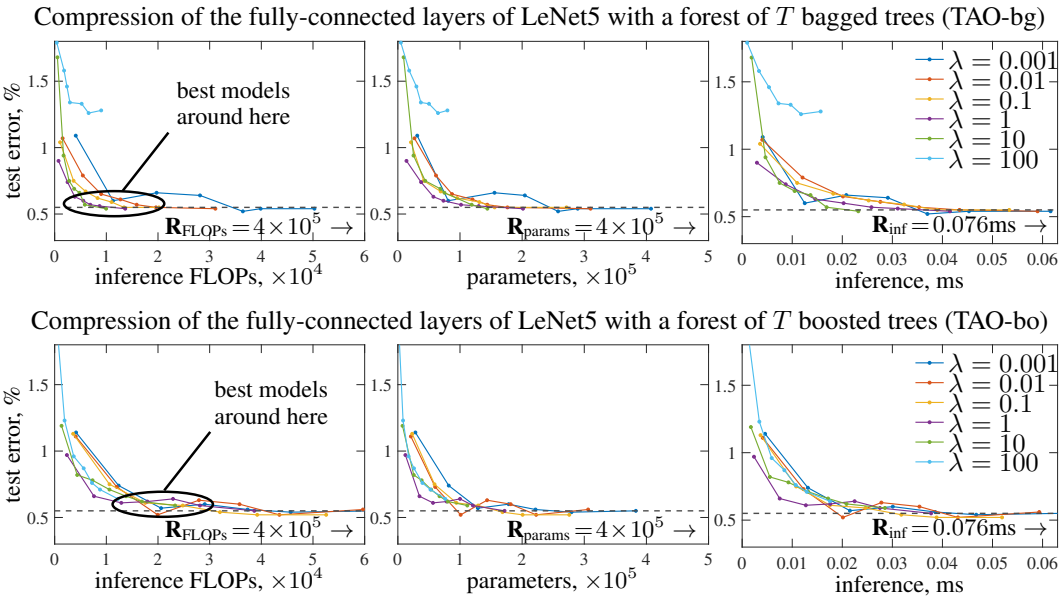

Figure 2: Tradeoff curves when compressing the last two fully-connected layers of LeNet5 (on MNIST). For a given value of λ we change the number of trees T and generate an entire curve when using bagging (top) or boosting (bottom). The test error of the reference network (along with its FLOPs, parameters, and inf. time) is denoted with a horizontal dashed line marked with **R**. The best models are as close as possible to the left bottom corner.

Selected results on LeNet5

Model	error, %	params.	FLOPs	inf.	inf.$^+$
reference FC layers	0.55	405K	405K	0.076	—
TAO, $T=1, \lambda=0.01$	1.11	20K	4K	**0.004**	—
TAO-bo, $T=5, \lambda=0.01$	**0.52**	101K	20K	0.020	—
TAO-bg, $T=11, \lambda=10$	0.57	118K	5.8K	0.017	—
CART, $T=1$	6.15	**2.8K**	**48**	0.043	0.022
RF, $T=5$	3.50	26K	159	0.446	0.026
RF, $T=7$	2.70	36K	240	0.592	0.027
RF, $T=100$	1.18	521K	3.4K	5.459	0.115
RF, $T=1000$	1.12	5.22M	34K	53.169	—
XGB, $T=1\times10$	6.19	2.7K	80	0.491	0.024
XGB, $T=10\times10$	1.50	32K	800	0.588	0.050
XGB, $T=100\times10$	0.84	55K	4.5K	0.760	0.148

Inference time vs test error

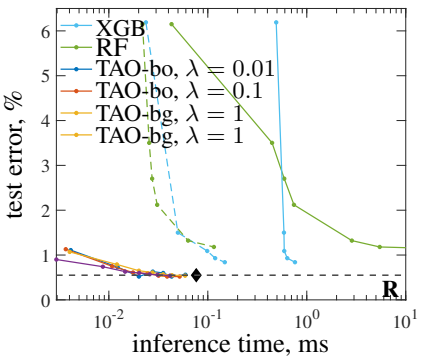

Figure 3: Left: selected comparisons to the baselines when compressing all (two) FC layers of LeNet5. Inference speed using regular baselines reported as inf. and TreeLite-compiled versions as inf.$^+$ Right: comparison to baselines along entire inference-error tradeoff curves. Our trees achieve of magnitudes better tradeoff than regular and highly optimized versions (using TreeLite, dashed lines) of these forests. The inference time (of the replaced part) is given with a diamond mark, ♦.

it is simple and much faster, since we need not train any neural net. That said, the remaining portions of the neural net can be compressed with other approaches (such as pruning or quantization), so the final model may use a combination of techniques in order to achieve the most compression.

Replacing the last layers of a neural net can also be seen as learning a decision forest on pretrained features (as is often done in transfer learning). However, our goal is to compress optimally a given neural net for a given task, and determining which layers to replace is part of the problem.

A limitation of our experiments is that our inference times may change depending on the hardware and software implementation of the different models. That said, since the least optimized implementation is that of the oblique forests, we expect their advantage to remain. Ultimately, however, the success of the approach—what portions of a neural net can be successfully replaced by an oblique forest—must be determined empirically in each case.

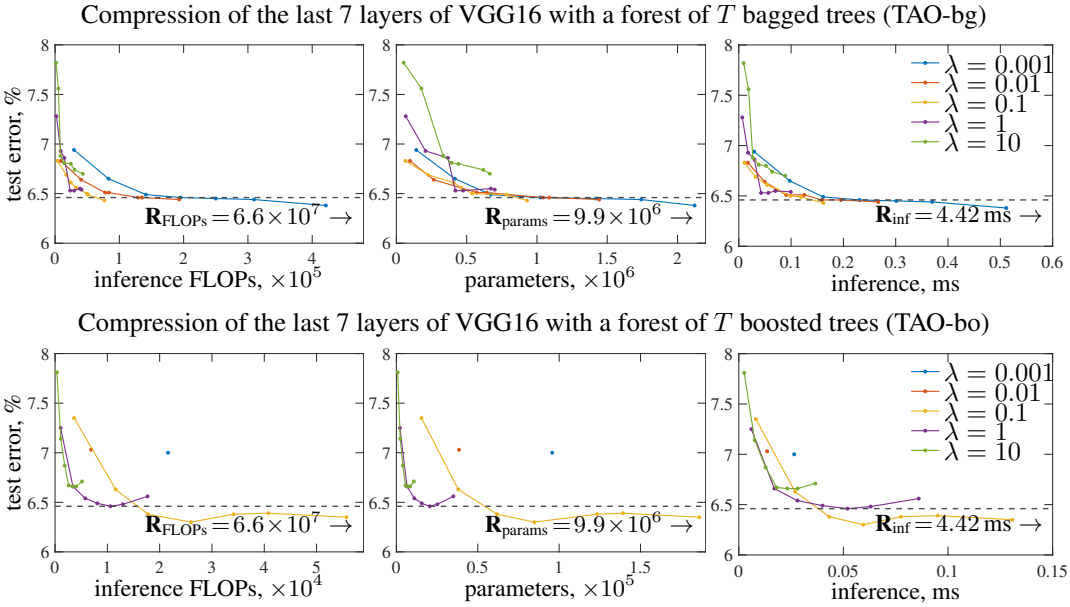

Figure 4: Similar curves as in Fig. 2 but now for the bagged ensemble of trees applied to compress the last 7 layers of the VGG16 on the CIFAR10: we jointly compress convolutional layers 10–13 and 3 fully-connected layers. Note that some boosting curves (e.g., $\lambda = 0.01$) have only a single reported result due to achieving 0 train error, thus no further boosting is possible. These 7 layers have 9.9M parameters and require 66.5 MFLOPs.

Selected results on VGG16

Model	error, %	param.	FLOPs	inf.	inf.$^+$
reference last 7 layers	6.46	10M	66M	4.419	—
TAO, $T = 1$, $\lambda = 0.01$	6.83	97K	8871	0.017	—
TAO-bg, $T = 9$, $\lambda = 0.01$	6.46	1M	129K	0.159	—
TAO-bo, $T = 9$, $\lambda = 1$	6.46	20K	10.5K	0.052	—
TAO-bo, $T = 7$, $\lambda = 0.1$	6.30	84K	26.0K	0.059	—
CART, $T = 1$	9.38	733	20	0.106	0.040
RF, $T = 5$	7.75	6.7K	145	1.725	0.042
RF, $T = 11$	7.02	13.7K	292	4.900	0.047
RF, $T = 1000$	6.47	1.4M	29K	109.58	0.612
XGB, $T = 1 \times 10$	9.84	829	60	2.368	0.044
XGB, $T = 10 \times 10$	6.86	12K	600	2.286	0.052
XGB, $T = 100 \times 10$	6.58	23K	1931	2.484	0.080

Inference time vs error

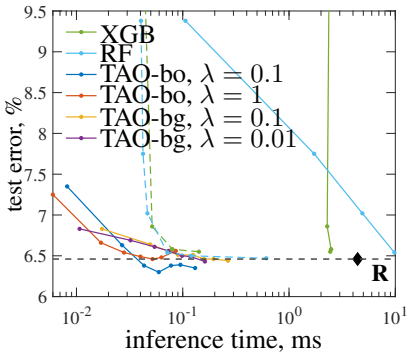

Figure 5: Similar comparisions as in Fig. 3 but now when compressing last 7 layers of the VGG16 on the CIFAR10, While our oblique trees have larger parameter and FLOPs count wrt regular CART/RF/XGB models, our models achieve significantly better tradeof in the inference-compression design space.

7 Conclusion

We have shown that a special type of decision forests, consisting of sparse oblique trees, can replace significant portions of a neural net—resulting in a neural net/forest hybrid—so that its prediction accuracy is retained but its inference is considerably accelerated. The same does not seem to hold as well for traditional forests of axis-aligned trees. The training time required to compress the neural net is also much faster than for compression approaches based on pruning, quantization or low-rank factorization that require neural net training; we just need to train the forest, which is much faster and highly parallelizable (within each individual tree, and across trees with bagging).

The success of this hybrid architecture opens up multiple directions for future research, such as replacing arbitrary portions of the neural net (rather than the last few layers), again with a teacher-student approach; using other types of trees or ensembling mechanisms; effectively training such an architecture end-to-end; and efficiently implementing oblique trees and forests in CPUs or GPUs.

REFERENCES

Yonathan Aflalo, Asaf Noy, Ming Lin, Itamar Friedman, and Lihi Zelnik-Manor. Knapsack pruning with inner distillation. arXiv:2002.08258, 2020.

Karim Ahmed, Mohammad Haris Baig, and Lorenzo Torresani. Network of experts for large-scale image categorization. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *Proc. 14th European Conf. Computer Vision (ECCV'16)*, pp. 516–532, Amsterdam, The Netherlands, October 11–14 2016.

Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, December 1995.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.

Leo J. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

Leo J. Breiman, Jerome H. Friedman, R. A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Calif., 1984.

Miguel Á. Carreira-Perpiñán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pp. 1211–1221. MIT Press, Cambridge, MA, 2018.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proc. of the 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2016)*, pp. 785–794, San Francisco, CA, August 13–17 2016.

Hyunsu Cho and Mu Li. Treelite: Toolbox for decision tree deployment. In *Proc. of the 1st Conf. Systems and Machine Learning (SysML 2018)*, Stanford, CA, February 15–16 2018.

Mark Craven and Jude W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Proc. of the 11th Int. Conf. Machine Learning (ICML'94)*, pp. 37–45, 1994.

Mark Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In David S. Touretzky, M. C. Mozer, and M. E. Hasselmo (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 8, pp. 24–30. MIT Press, Cambridge, MA, 1996.

Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. 108(4):485–532, April 2020.

Pedro Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2(1–4): 187–202, 1998.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *J. Machine Learning Research*, 9:1871–1874, August 2008.

Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Computer and System Sciences*, 55(1):119–139, 1997.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. arXiv:1711.09784, November 27 2017.

Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *Int. J. Computer Vision*, 129:1789–1819, June 2021.

Trevor J. Hastie, Robert J. Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009.

Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In Hal Daumé III and Aarti Singh (eds.), *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pp. 4138–4148, Online, July 13–18 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, pp. 770–778, Las Vegas, NV, June 26 – July 1 2016.

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán. A flexible, extensible software framework for model compression based on the LC algorithm. arXiv:2005.07786, May 15 2020.

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán. Optimal selection of matrix shape and decomposition scheme for neural network compression. In *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'21)*, pp. 3250–3254, Toronto, Canada, June 6–11 2021.

Yani Ioannou, Duncan P. Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. arXiv:1603.01250, March 3 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proc. of the 32nd Int. Conf. Machine Learning (ICML 2015)*, pp. 448–456, Lille, France, July 6–11 2015.

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, March 1994.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Buló. Deep neural decision forests. In *Proc. 15th Int. Conf. Computer Vision (ICCV'15)*, pp. 1467–1475, Santiago, Chile, December 11–18 2015.

Alexander Kozlov, Ivan Lazarevich, Vasily Shamporov, Nikolay Lyalyushkin, and Yury Gorbachev. Neural network compression framework for fast model inference. arXiv:2002.08679, February 20 2020.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, November 1998.

Tailin Lianga, John Glossnera, Lei Wanga, and Shaobo Shi. Pruning and quantization for deep neural network acceleration: A survey. arXiv:2101.09671, January 24 2021.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. HRank: Filter pruning using high-rank feature map. In *Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20)*, pp. 1526–1535, Seattle, WA, June 14–19 2020.

Calvin Murdock, Zhen Li, Howard Zhou, and Tom Duerig. Blockout: Dynamic model selection for hierarchical deep networks. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, pp. 2583–2591, Las Vegas, NV, June 26 – July 1 2016.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *J. Machine Learning Research*, 12:2825–2830, October 2011. Available online at `https://scikit-learn.org`.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018)*, Vancouver, Canada, April 30 – May 3 2018.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

Samuel Rota Buló and Peter Kontschieder. Neural decision forests for semantic image labelling. In *Proc. of the 2014 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'14)*, pp. 81–88, Columbus, OH, June 23–28 2014.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: the sparsely-gated mixture-of-experts layer. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, April 24–26 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.

Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*, pp. 6166–6175, Long Beach, CA, June 9–15 2019.

Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (eds.), *Proc. 15th European Conf. Computer Vision (ECCV'18)*, pp. 3–18, Munich, Germany, September 8–14 2018.

Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Suzanne Petryk, Sarah Adel Bargal, and Joseph E. Gonzalez. NBDT: Neural-backed decision tree. In *Proc. of the 9th Int. Conf. Learning Representations (ICLR 2021)*, Online, April 25–29 2021.

Peisong Wang, Qinghao Hu, Yifan Zhang, Chunjie Zhang, Yang Liu, and Jian Cheng. Two-step quantization for low-bit neural networks. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, pp. 4376–4384, Salt Lake City, UT, June 18–22 2018.

Jiaxiang Wu, Yao Zhang, Haoli Bai, Huasong Zhong, Jinlong Hou, Wei Liu, Wenbing Huang, and Junzhou Huang. PocketFlow: An automated framework for compressing and accelerating deep neural networks. In *NIPS Workshop on Compact Deep Neural Network Representation with Industrial Applications (CDNNRIA)*, 2018.

Arman Zharmagambetov and Miguel Á. Carreira-Perpiñán. Smaller, more accurate regression forests using tree alternating optimization. In Hal Daumé III and Aarti Singh (eds.), *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pp. 11398–11408, Online, July 13–18 2020.

Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. Multi-class AdaBoost. *Statistics and Its Interface*, 2(3):349–360, 2009.

Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. Neural network distiller: a Python package for DNN compression research. arXiv:1910.12232, October 27 2019.

APPENDICES

This appendix contains additional graphical illustrations explaining our approach (sec. A), full details of experimental setup (sec. B), details of LeNet experiments (sec. C), details of VGG experiments on CIFAR10 (sec. D) and CIFAR100 (sec. E), details of ResNet56 experiment (sec. F), and details on combining tree-based compression with other mechanisms (sec. G).

A ILLUSTRATION OF OUR APPROACH

Figures 6–7 illustrate pictorially our neural net compression approach, and the construction of the forests, respectively.

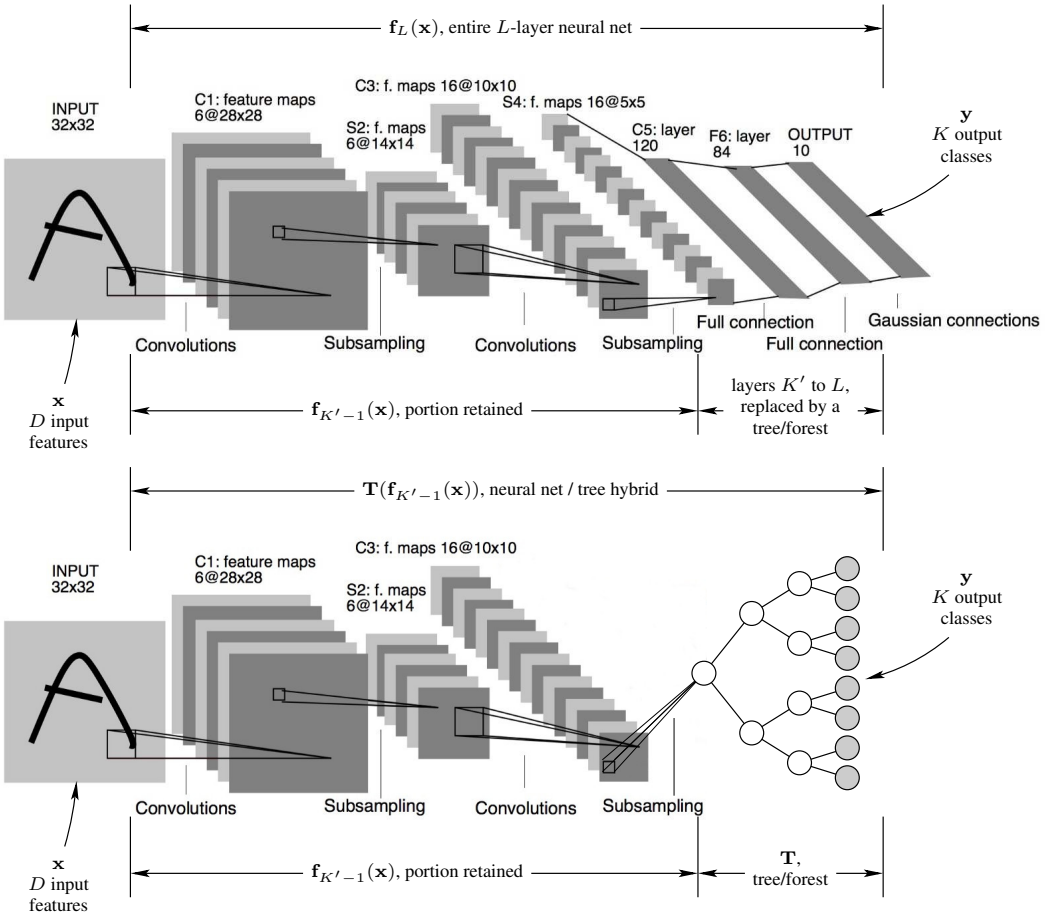

Figure 6: Illustration of our neural net compression approach. *Top*: a standard neural network architecture (the LeNet5 of (LeCun et al., 1998)), showing the portion retained and the portion replaced. *Bottom*: the resulting model. The parameters of the retained part are unchanged; the tree/forest is trained via the teacher-student approach. This diagram is only for illustration purposes, it does not necessarily correspond to any specific experiment in the paper.

B DETAILS OF EXPERIMENTAL SETUP

We give the details of the experimental setup for TAO (sec. B.1), the other baselines (sec. B.2, and describe our methodology in reporting various metrics (sec. B.3). Summary of hardware and software resources used in our experiments can be found in Table 2.

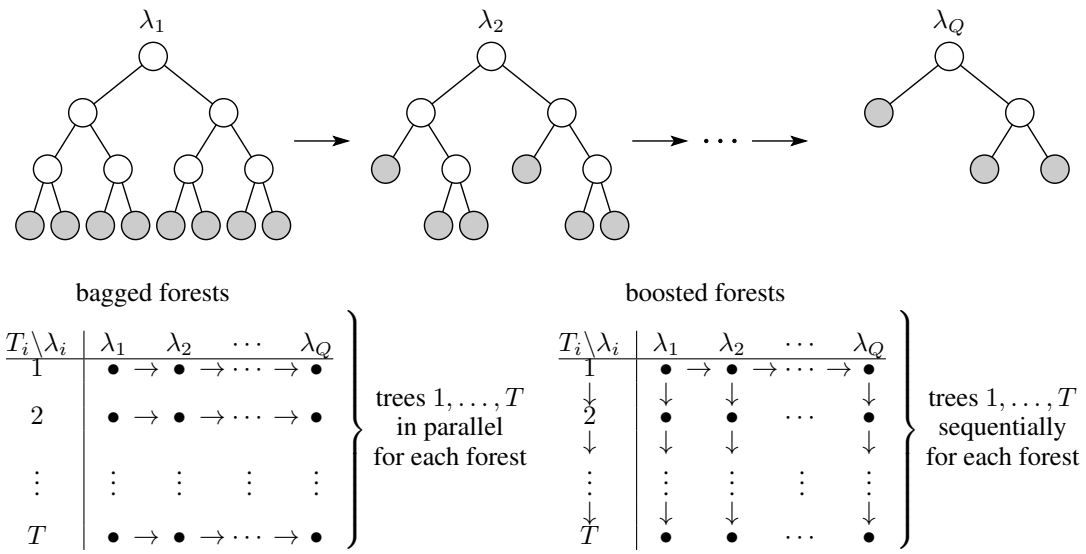

Figure 7: *Top*: illustration of the regularization path for $\lambda_1 < \lambda_2 < \cdots < \lambda_Q$. Each tree is trained with TAO using the same initial tree structure (a complete tree of depth Δ) but with warm-start, i.e., the tree for λ_i is initialized from the parameters of the tree for λ_{i-1}. As λ increases, so does the sparsity penalty, which encourages weight vectors at the decision nodes to become sparse and nodes to be pruned. Although it is not shown, the parameters at each tree (in the decision nodes and leaves) are different, since TAO optimizes jointly over all the parameters. *Bottom*: illustration of the procedure to construct a forest of T_i trees and sparsity hyperparameter λ_i, for bagging (left) and boosting (right), for $\lambda_i \in \{\lambda_1, \lambda_2, \ldots, \lambda_Q\}$ and $T_i \in \{1, \ldots, T\}$. Each black circle $\bullet$ represents one tree. The (T_i, λ_i) forest consists of all the trees in the column for λ_i, for rows 1–T_i. Hence, each column is a forest (T, λ_i). A horizontal arrow "$\rightarrow$" means the next tree is initialized from the previous one, to construct a regularization path (as in the top panel). A vertical arrow "$\downarrow$" means, for boosting, that the next tree $t+1$ depends on all the previous trees $1, \ldots, t$, to construct a greedy additive model. In bagging, the trees $1, \ldots, T$ are independent.

B.1 TAO EXPERIMENTS

We have re-implemented the TAO algorithm and the inference of the TAO-generated oblique decision trees in the C++. We use LIBLINEAR v2.30 as the node-problem solver within our library.

Hyperparameters There are three hyperparameters of interest: the amount of sparsity of the nodes in the ensemble (controlled by λ), the number of trees in the ensemble T, and the depth of the decision trees d. Out of these three hyperparameters, we set d to be as large as (reasonably) possible, and initialize the trees to be a balanced tree with $2^d + 1$ decision nodes. A typical range for d is 4 to 11. In our experiments we vary number of trees T, amount of sparsity λ, and generate entire error-compression tradeoffs. We set the number of maximum iterations of the TAO algorithm to be 20: here, one iteration is a single optimization pass over all decision nodes and leaves. Decision nodes require training ℓ_1 regularized logistic regressions, which were optimized using LIBLINEAR with default settings. The decision node parameters are initialized randomly by sampling from a standard Gaussian. For boosting experiments we use the learning rate of 0.1.

Training time and other resources The training of the forest using TAO can be efficiently parallelized on two levels: first, all non-descending nodes' problems are independent of each other, thus they can be trained in parallel; second, if trees are independent of each other (in case bagging) we have tree-level parallelism. In our experiments, we utilize both parallelization options. In general, the training of the forests is extremely fast. For example, using 16 cores of Intel Xeon E5-2699 clocked at 2.30 GHz, our longest TAO bagging experiment (VGG16, conv8 $\rightarrow$ output) with $T = 15$ finishes in 24 minutes, and the longest TAO boosting experiment (on the same dataset) finishes in 95 minutes. In average, any reported experiments finish within 40 minutes.

In terms of memory usage, the amount of used RAM is proportional to the level of parallelization and the size of dataset. The recorded largest RAM usage for a single experiment was 54.4 GB when training in parallel on 16 cores.

B.2 BASELINES

We compare TAO trained forests to random forest and boosted forest models. We use the scikit-learn Pedregosa et al. (2011) library to train the random forest models, and the XGBoost Chen & Guestrin (2016) library for boosted forests. To strengthen the inference speed measurements of these baselines, we further compiled the generated forests using TreeLite library Cho & Li (2018). In the paper we report the inference times for regular and TreeLite-optimized models.

Random forest hyperparameters When training the random forest ensembles we use the following settings: no limit on the depth of the trees (i.e., fully grown trees), no pruning, the size of the randomly selected features subset is $\sqrt{D}$ where D is the size of the training input. Other options were populated with the default scikit-learn settings for random forests.

XGBoost hyperparameters We set a predetermined number of rounds for boosting (which is equivalent to the number of trees T/C in the ensemble, where C is the number of classes). For every T value we are using, we find the optimal depth and learning rate using a grid search with the following grid options: tree depth in $\{4, 6, 8, 10\}$ and learning rate in $\{0.1, 0.3, 0.5\}$. The quality of the settings were validated on randomly selected hold-out subset of training points.

B.3 METRICS

As part of our evaluation, we report several metrics of interest: the number of floating-point operations (FLOPs), the number of the parameters, and the inference time. Here we discuss how we compute these values.

FLOPs We define a total FLOPs count as the number of fused additions and multiplications *involving nonzero weights* happening during the inference of a single input[1]. With such a definition, for instance, the expression $1 \times 2 + 0 \times 3$ will be counted as a single FLOP: although there are four operations (2 multiplications, 2 additions). While we can additionally leverage the sparsity in inputs themselves—with prevalent ReLU non-linearities, a significant portion of the between-layer activations are exactly zero—we opt not to involve it into FLOPs count to be consistent with the literature. For neural networks, the inference FLOPs is a constant and does not depend on input values. For trees, it is different: depending on which branch of the tree is being traversed, the number

[1]For the axis-aligned trees we extend this definition to include comparison operator (e.g., $x > 4$) as a single FLOP

Machine details

CPU	2× Xeon E5-2699 v3, 2.30GHz
RAM	256 GB ECC DDR3 at 2133 MHz
OS	Ubuntu 20.04.2 LTS
Kernel	Linux 5.4.0-70-generic
Storage	Hard drive

Software details

TAO	re-implementation in C++
scikit-learn	version 0.24.2
xgboost	version 1.3.3
TreeLite	version 1.3.0
LIBLINEAR	version 2.30
ONNXruntime	version 1.7.0
PyTorch	version 1.4.1

Table 2: Summary of the training machine (left) and the software (right) used in our experiments. Although our machine has 72 CPU cores, we used no more than 16 cores for any single experiment to share the resources with other machine users. The RAM usage for TAO was proportional to the size of the dataset: at most, we recorded using 54.4 GBs of RAM when parallelizing on 16 cores (bagging); for TAO boosting, the RAM usage was never higher than 10GB. Except for the TAO training code, which we have re-implemented internally in C++, we used standard software packages available online. Our TAO implementation uses LIBLINEAR v2.30 as the node-problem solver.

of floating-point operations might differ. Therefore, for decision trees we report an average FLOPs count per example when making an inference over the training dataset.

Parameters We define the number of parameters as the total count of non-zero weights in the model. We opt to report the number of parameters instead of the required storage bits (when model is saved to disk) for two reasons. First, the actual disk storage will depend on the chosen format for the sparse weights, and it is often possible to manipulate the final storage size by simply changing the format or by chaining it with other compression forms (e.g., quantization). Second, the number of nonzero parameters can be compared directly across different models and papers, while the storage size depends on many (hidden) factors which will hinder future comparisons.

Inference time We define the inference time in a similar manner as the FLOPs count: inference time is the amount of time required to complete an inference of a single input through a model. In case of trees, the inference time is averaged over the dataset. We measure the inference times on a single core of Intel Xeon CPU model E5-2699v3 clocked at 2.30GHz. To make the actual measurements consistent we measured all inference times within python, and use the following strategy: we record the start time by calling `time.time()`, we then run the inference through entire training data (calling appropriate backend) using batch size of 1, and then record the finishing time. Then, the reported inference time is computed as:

$$\text{inf. time} = \frac{\text{start time} - \text{end time}}{\text{total processed examples}}. \tag{2}$$

To measure the neural net times we use ONNXruntime v1.7 as the inference backend due to highly optimized CPU kernels.

C LeNet5 on MNIST

We trained Caffe version[2] of the LeNet5 network containing 431K parameters which requires 2.29M floating-point operations (FLOPs) to perform the inference. The network consists of two convolutional and two fully-connected (FC) layers, and we replace both FC layers with our proposed scheme. The two FC layers that are being compressed have 405K params and 405K FLOPs.

Training details The reference network was trained to the test error of 0.55% using the SGD with batch size of 256 images and learning rate of 0.05 which was decayed by 0.99 after every epoch. The total training epochs was 300. The input to the first FC layer is a vector of size 800. We applied TAO compression using boosting and bagging for $T \in \{1, 3, 5, 7, 9, 11, 15\}$ and $\lambda \in \{0.001, 0.01, 0.1, 1, 10, 100\}$. In all experiments we used trees of depth $d = 6$.

D VGG16 on CIFAR10

We adapt the ImageNet version of the VGG16 Simonyan & Zisserman (2015) for the CIFAR10 dataset. The network has the same number of convolutional filters and parameters and the differences are in the dimensions of fully-connected layers (the FC layers have 512 units instead of 4096 as in the ImageNet version) and the usage of batch-normalization Ioffe & Szegedy (2015) between layers. In total, the network has 13 convolutional and 3 fully-connected layers (hence the name VGG16), 15.2M parameters, and requires 313 MFLOPs of computation for the inference pass. The reference network were trained to the test error of 6.46%.

Training details The reference VGG16 network is trained using SGD with a learning rate of 0.035 which was decayed by 0.97716 after every epoch. The network was trained for 350 epochs in total. We used a standard data augmentation of: first zero padding the images from all sides with a row of 2 pixels, and then cropping out 32×32 sized part at a random location, followed by a random left-to-right flip. We applied TAO compression using bagging for $T \in \{1, 3, 5, 7, 9, 11\}$ and $\lambda \in \{0.001, 0.01, 0.1, 1, 10\}$. We set the following depth d for the trees: for conv10 $\rightarrow$ output compression (Figure 5 in the main paper) we had $d = 6$, for layerwise compression results (Figure 1 in the main paper) we varied d in range 6–10, and for compression of the FC layers we set $d = 4$.

[2] `https://github.com/BVLC/caffe/blob/master/examples/mnist/lenet.prototxt`

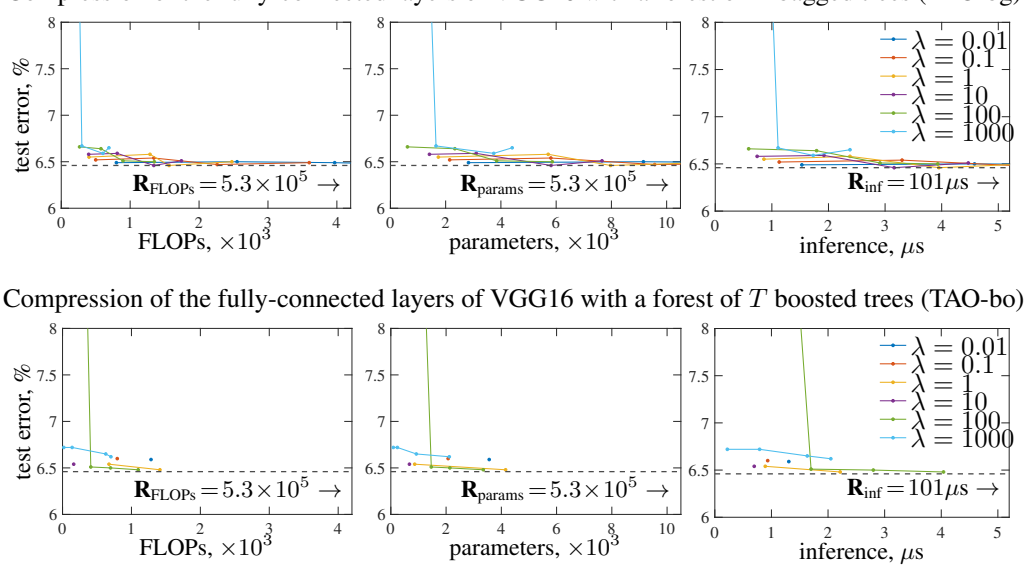

Figure 8: Similar curves as in Figures 2 and 4 of the main paper, but now applied to compress the last 3 fully-connected layers (params/FLOPs: 0.53M, inference: 101 μs) of the VGG16 on the CIFAR10. A single tree ($T = 1$) is already powerful enough to compress all FC layers with a minimal degradation: we can achieve the test error of 6.49% using only 2827 parameters ($\downarrow 187\times$), which requires 800 FLOPs ($\downarrow 663\times$) and runs in 1.5 μs ($\downarrow 66\times$) in average. Bagging (top) and boosting (bottom) the trees makes the accuracy identical to the NN's performance and still give competitive operating characteristics: a bagged ensemble of $T = 5$ trees has the same error as the reference net (6.46%) but needs only 5816 parameters ($\downarrow 91.2\times$), 1343 FLOPs ($\downarrow 395\times$), and runs in 3.1 μs ($\downarrow 32\times$).

Compression of the fully-connected FC layers of VGG16 (CIFAR10)

Selected results

Model	error, %	params.	FLOPs	inf.	inf.$^+$
reference FC layers	6.46	530K	530K	0.1013	—
TAO, $T = 1$, $\lambda = 0.01$	6.49	2827	800	**0.0015**	—
TAO-bg, $T = 5$, $\lambda = 10$	**6.46**	5816	1343	0.0031	—
TAO-bo, $T = 3$, $\lambda = 1$	6.48	4154	1413	0.0022	—
CART, $T = 1$	7.39	**19**	**5**	0.0937	0.0180
RF, $T = 5$	6.66	133	42	1.9418	0.0196
RF, $T = 7$	6.63	289	82	2.7108	0.0214
RF, $T = 100$	6.47	3K	843	17.574	0.0367
RF, $T = 1000$	6.46	29K	8K	200.00	0.1344
XGB, $T = 1 \times 10$	8.44	193	45	0.3647	0.0212
XGB, $T = 10 \times 10$	7.74	2K	408	0.3725	0.0249

Inference time vs error

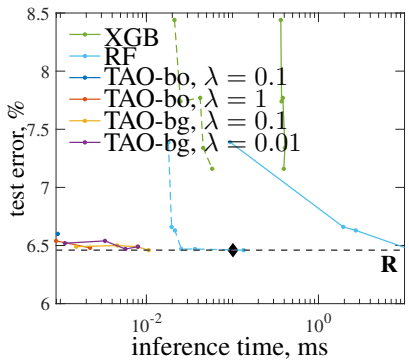

Figure 9: Selected comparisons when compressing all FC layers of the VGG16 trained on the CIFAR10. While our oblique trees have larger parameter and FLOPs count wrt regular CART/RF models, in the inference-compression tradeoff our time our models are the fastest thanks to shorter depth and extreme decision node sparsity. The TreeLite optimized inference times are given by inf.$^+$ and depicted using the dashed lines.

Additional experiments Along with the VGG16 experiments reported in the main paper, here we report additional experiments on compressing all (three) fully-connected layers of VGG16 (see Figures 8–9), and on compression of the final softmax layer (Figure 11, bottom).

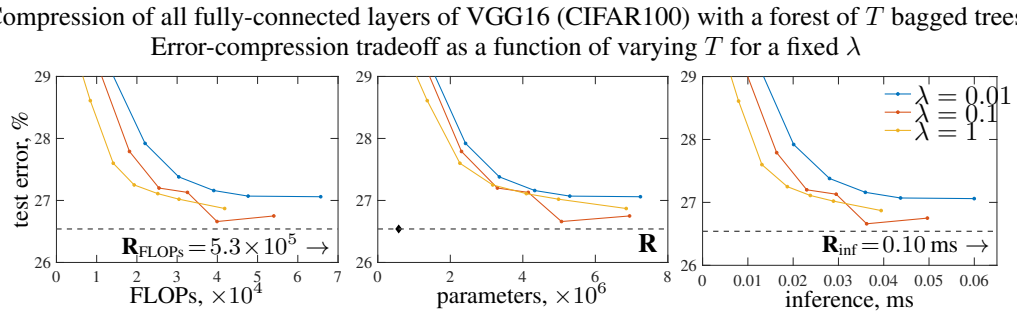

Figure 10: Similar curves as in Figure 8 but now for the bagged ensemble of trees applied to compress all fully-connected layers of the VGG16 on the CIFAR100. These layers have 576K parameters and require 576K FLOPs. The test error of the reference network is indicated by a horizontal dashed line with a label **R**, and the black diamond symbol (♦) along this dashed line indicates the tradeoff of the reference model; if no black diamond is shown, the reference model's operating point is outside of the axis on the right. Speeding-up the fully-connected layers of the CIFAR100 version of VGG16 is a harder task, however, a forest of $T = 11$ trees trained with $\lambda = 0.1$ can achieve a speed-up of $14.45\times$. Unfortunately, such a forest requires $5\times$ more parameters than the reference fully-connected layers on their own.

E VGG16 ON CIFAR100

We modified the original VGG16 to the CIFAR100 dataset in the same way as in sec. D with only difference of having 100 output classes instead of 10. The network still has 16 layers (13 convolutional and 3 fully-connected), but the FLOPs and parameters counts are slightly higher: 313M FLOPs and 15.3M parameters respectively. The reference network is trained to have a test error of 26.54%.

Training details The reference net was trained using the same settings as in sec. D). We applied TAO compression using bagging for $T \in \{1, 3, 5, 7, 9, 11, 15\}$ and $\lambda \in \{0.001, 0.01, 0.1, 1, 10, 100\}$. We used trees of depth $d = 9$ for softmax layer compression, and trees of depth $d = 11$ for compression of all fully-connected layers.

Results We report the results of compression of all (three) fully-connected layers using TAO-bagging (Figure 10), and the results of softmax layer compression using TAO-bagging as well (Figure 11, bottom)

F RESNET56 ON CIFAR10

We train a 56-layer version of ResNet He et al. (2016) designed for CIFAR10 dataset. The network has 125M FLOPs, 0.84M parameters, and achieve a test error of 6.58%. We apply our compression mechanism to replace final three layers (two convolutional and one fully-connected). These three layers have 74K parameters and 4.7M FLOPs.

Training details To train the reference model we use the settings (data-augmentation, learning-rate schedule) as recommended in the original paper He et al. (2016). We train TAO forests with depth $d = 7$ and $\lambda = 0.01$.

Results With a bagged forest of $T = 3$ trees, we can replace the last three layers of the ResNet56 to a model having 300K parameters, and 31K FLOPs that achieves a test error of 7.78%. This outperforms random forest and XGBoost based compression: even a large random forest model with $T = 1000$ trees, with 11M parameters and 37K FLOPs achieves much higher test error of 8.97%. The XGBoost forest with $T = 10 \times 100$ trees achieve a test error of 8.36% with 68K parameters and 700 FLOPs.

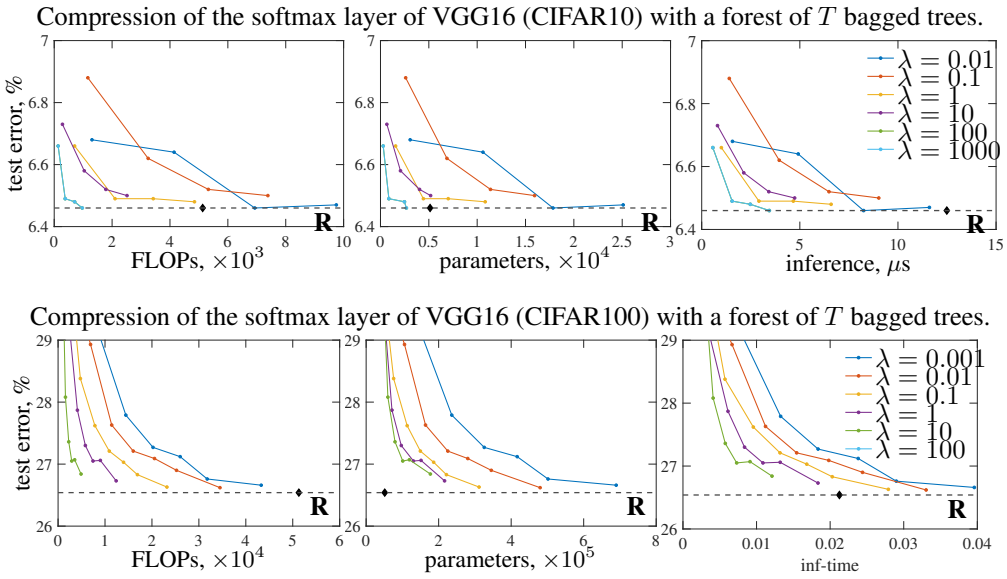

Figure 11: Compression of softmax layer of VGG16 on CIFAR10. The test error of the softmax layer is indicated by a horizontal dashed line with a label **R**, and the black diamond symbol (⧫) along this dashed line indicates the operating point of the reference model.

G COMBINATION WITH OTHER COMPRESSIONS

In the main paper, we reported the results of combining the tree-based compression with already downsized models obtained through low-rank compression. Here we report the hyperparameters of this experiment.

Training details The low-rank models were obtained following the training routine of the original paper Idelbayev & Carreira-Perpiñán (2021). For our compression, we used a bagged forest of $T = 5$ trees with $\lambda = 0.01$.