Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things

Ashish Kumar [1] **Saurabh Goyal** [2] **Manik Varma** [1]

Abstract

This paper develops a novel tree-based algorithm, called Bonsai, for efficient prediction on IoT devices – such as those based on the Arduino Uno board having an 8 bit ATmega328P microcontroller operating at 16 MHz with no native floating point support, 2 KB RAM and 32 KB read-only flash. Bonsai maintains prediction accuracy while minimizing model size and prediction costs by: (a) developing a tree model which learns a single, shallow, sparse tree with powerful nodes; (b) sparsely projecting all data into a low-dimensional space in which the tree is learnt; and (c) jointly learning all tree and projection parameters. Experimental results on multiple benchmark datasets demonstrate that Bonsai can make predictions in milliseconds even on slow microcontrollers, can fit in KB of memory, has lower battery consumption than all other algorithms while achieving prediction accuracies that can be as much as 30% higher than state-of-the-art methods for resource-efficient machine learning. Bonsai is also shown to generalize to other resource constrained settings beyond IoT by generating significantly better search results as compared to Bing's L3 ranker when the model size is restricted to 300 bytes. Bonsai's code can be downloaded from (BonsaiCode).

1. Introduction

Objective: This paper develops a novel tree-based algorithm, called Bonsai, which can be trained on a laptop, or the cloud, and can then be shipped onto severely resource constrained Internet of Things (IoT) devices.

Resource constrained devices: The Arduino Uno board has an 8 bit ATmega328P microcontroller operating at 16 MHz with 2 KB SRAM and 32 KB read-only flash memory. The BBC Micro:Bit has a 32 bit ARM Cortex M0 microcontroller operating at 16 MHz with 16 KB SRAM and 256 KB read-only flash. Neither provides hardware support for floating point operations. Billions of such tiny IoT microcontrollers have been deployed in the world (Meunier et al., 2014). Before deployment, the OS and all application code and data are burnt onto flash, leaving only a few KB for storing the trained ML model, prediction code, feature extraction code and associated data and parameters. After deployment, the only writable memory available is the 2 KB (Uno) or 16 KB (Micro:Bit) of SRAM which might not be sufficient to hold even a single feature vector.

The Internet of Things: A number of applications have been developed for consumer, enterprise and societal IoT including predictive maintenance, intelligent healthcare, smart cities and housing, *etc*. The dominant paradigm for these applications, given the severe resource constraints of IoT devices, has been that the IoT device is dumb – it just senses its environment and transmits the sensor readings to the cloud where all the decision making happens.

Motivating scenarios: This paper proposes an alternative paradigm where the IoT device can make predictions locally without necessarily connecting to the cloud. This enables many scenarios, beyond the pale of the traditional paradigm, where it is not possible to transmit data to the cloud due to latency, bandwidth, privacy and energy concerns. For instance, consider a microcontroller implanted in the brain which warns patients about impending seizures so that they can call for help, pull over if they are driving, *etc*. Making predictions locally would allow the device to work everywhere irrespective of cloud connectivity. Furthermore, alerts could be raised more quickly with local predictions than if all the sensor readings had to be first transmitted to the cloud. In addition, since the energy required for executing an instruction might be much lower than the energy required to transmit a byte, making predictions locally would extend battery life significantly thereby avoiding repeated brain surgery and might also prevent brain tissue damage due to excess heat dissipation from the communicating radio. Finally, people might not be willing to transmit such sensitive data to the cloud. These characteristics are shared by many other scenarios including implants in the heart, precision agriculture on disconnected farms, smart spectacles for the visually impaired, *etc*.

[1]Microsoft Research, Bangalore, India [2]CSE Department, IIT Delhi, India. Correspondence to: <manik@microsoft.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

Tree algorithms: Tree algorithms are general and can be used for classification, regression, ranking and other problems commonly found in the IoT setting. Even more importantly, they are ideally suited to IoT applications as they can achieve good prediction accuracies with prediction times and energies that are logarithmic in the number of training points. Unfortunately, they do not directly fit on tiny IoT devices as their space complexity is linear rather than logarithmic. In particular, learning shallow trees, or aggressively pruning deep trees or large ensembles, to fit in just a few KB often leads to poor prediction accuracy.

Bonsai: This paper develops a novel tree learner, called Bonsai, designed specifically for severely resource constrained IoT devices based on the following contributions. First, Bonsai learns a single, shallow, sparse tree so as to reduce model size but with powerful nodes for accurate prediction. Second, both internal and leaf nodes in Bonsai make non-linear predictions. Bonsai's overall prediction for a point is the sum of the individual node predictions along the path traversed by the point. Path based prediction allows Bonsai to accurately learn non-linear decision boundaries while sharing parameters along paths to further reduce model size. Third, Bonsai learns a sparse matrix which projects all data points into a low-dimensional space in which the tree is learnt. This allows Bonsai to fit in a few KB of flash. Furthermore, the sparse projection is implemented in a streaming fashion thereby allowing Bonsai to tackle IoT applications where even a single feature vector might not fit in 2 KB of RAM. Fourth, rather than learning the Bonsai tree node by node in a greedy fashion, all nodes are learnt jointly, along with the sparse projection matrix, so as to optimally allocate memory budgets to each node while maximising prediction accuracy.

Implementation: Another contribution is an efficient implementation of Bonsai which reduces its prediction costs on the Arduino and Micro:Bit to be even lower than that of an unoptimized linear classifier. This allows Bonsai to enjoy the prediction accuracy of a non-linear classifier while paying less than linear costs. This paper does not focus on the system and implementation details due to space limitations but the interested reader is referred to the publically available source code (BonsaiCode).

Results: These contributions allow Bonsai to make predictions in milliseconds even on slow microcontrollers, fit in a few KB of flash and extend battery life beyond all other algorithms. Furthermore, it is demonstrated on multiple benchmark datasets that Bonsai's prediction accuracies can approach those of uncompressed kNN classifiers, RBF-SVMs, single hidden layer neural networks and gradient boosted decision tree ensembles whose models might take many MB of RAM. It is also demonstrated that Bonsai's prediction accuracies for a given model size can be as much as 30% higher than state-of-the-art methods for resource-efficient machine learning. Finally, Bonsai is shown to generalize to other resource constrained settings beyond IoT by producing significantly better search results than Bing's L3 ranker when the model size is restricted to 300 bytes.

2. Related Work

The literature on resource-efficient machine learning is vast and specialized solutions have been developed for reducing the prediction costs of kNN algorithms (Kusner et al., 2014b; Wang et al., 2016), SVMs (Hsieh et al., 2014; Jose et al., 2013; Le et al., 2013; Li et al., 2016), deep learning (Iandola et al., 2016; Han et al., 2016; Yang et al., 2015; Denton et al., 2014; Wu et al., 2016; Rastegari et al., 2016; Hubara et al., 2016; Shankar et al., 2016; Ioannou et al., 2016a), model compression (Bucilua et al., 2006; Ba & Caruana, 2014), feature selection (Kusner et al., 2014a; Xu et al., 2013; 2012; Nan et al., 2015; Wang et al., 2015) and applications such as face detection (Viola & Jones, 2004).

Resource-efficient tree classifiers are particularly germane to this paper. The standard approach is to greedily grow the decision tree ensemble node by node until the prediction budget is exhausted. A popular alternative is to first learn the random forest or gradient boosted decision tree ensemble to maximize prediction accuracy and then use pruning techniques to meet the budget constraints (Duda et al., 2002; Dekel et al., 2016; Nan et al., 2016; Li, 2001; Breiman et al., 1984; Zhang & Huei-chuen, 2005; Sherali et al., 2009; Kulkarni & Sinha, 2012; Rokach & Maimon, 2014; Joly et al., 2012). Unfortunately, such techniques are fundamentally limited as they attempt to approximate complex non-linear decision boundaries using a small number of axis-aligned hyperplanes. This can lead to poor prediction accuracies as observed in Section 5.

Tree models have also been developed to learn more complex decision boundaries by moving away from learning axis-aligned hyperplanes at internal nodes and constant predictors at the leaves. For instance, (Breiman, 2001; Murthy et al., 1994; Kontschieder et al., 2015) learnt more powerful branching functions at internal nodes based on oblique cuts and full hyperplanes while (Utgoff, 1989; Hsieh et al., 2014) learnt more powerful leaf node predictors based on linear classifiers, kernelized SVMs, *etc*. Bonsai achieves better budget utilization than such models by learning shorter trees, typically depth 4 or lower, and by sharing the parameters between leaf node predictors.

The models closest to Bonsai are Decision Jungles (Shotton et al., 2013) and LDKL (Jose et al., 2013). Bonsai improves upon LDKL by learning its tree in a low-dimensional space, learning sparse branching functions and predictors and generalizing the model to multi-class classi-

fication, ranking, *etc*. Decision Jungles are similar to Bonsai in that they share node parameters using a DAG structure. Unfortunately, Decision Jungles need to learn deep tree ensembles with many nodes as they use weak constant classifiers as leaf node predictors. Bonsai can have lower model size and higher accuracy as it learns a single, shallow tree in a low-dimensional space with non-linear predictors.

Note that while tree based cost-sensitive feature selection methods are not directly relevant, their performance is nevertheless empirically compared to Bonsai's in Section 5.

3. The Bonsai Model for Efficient Prediction

Overview: Bonsai learns a single, shallow sparse tree whose predictions for a point $\mathbf{x}$ are given by

$$\mathbf{y}(\mathbf{x}) = \sum_{k} I_k(\mathbf{x}) \mathbf{W}_k^\top \mathbf{Z}\mathbf{x} \circ \tanh(\sigma \mathbf{V}_k^\top \mathbf{Z}\mathbf{x}) \qquad (1)$$

where $\circ$ denotes the elementwise Hadamard product, σ is a user tunable hyper-parameter, $\mathbf{Z}$ is a sparse projection matrix and Bonsai's tree is parameterized by I_k, $\mathbf{W}_k$ and $\mathbf{V}_k$ where $I_k(\mathbf{x})$ is an indicator function taking the value 1 if node k lies along the path traversed by $\mathbf{x}$ and 0 otherwise and $\mathbf{W}_k$ and $\mathbf{V}_k$ are sparse predictors learnt at node k. The prediction function is designed to minimize the model size, prediction time and prediction energy, while maintaining prediction accuracy, even at the expense of increased training costs. The function is also designed to minimize the working memory required as the Uno provides only 2 KB of writeable memory for storing the feature vector, programme variables and intermediate computations.

Streaming sparse projection: Bonsai projects each D-dimensional input feature vector $\mathbf{x}$ into a low $\hat{D}$-dimensional space using a learnt sparse projection matrix $\mathbf{Z}_{\hat{D} \times D}$. Bonsai uses fixed point arithmetic for all math computation, including $\mathbf{Z}\mathbf{x}$, when implemented on the IoT device so as to avoid floating point overheads. Note that $\hat{D}$ could be as low as 5 for many binary classification applications. This has the following advantages. First, it reduces Bonsai's model size as all tree parameters are now learnt in the low-dimensional space. Second, when $\hat{D}$ is small, $\mathbf{Z}\mathbf{x}$ could be stored directly in the microcontroller's registers thereby reducing prediction time and energy. Third, learning the projection matrix jointly with the tree parameters improves prediction accuracy. Fourth, since $\mathbf{Z}\mathbf{x}$ can be computed in a streaming fashion, this allows Bonsai to tackle IoT applications where even a single feature vector cannot fit in 2 KB of SRAM. This is critical since standard tree implementations are unable to handle a streaming feature vector – the entire feature vector needs to be streamed for the root node to determine whether to pass the point down to the left or right child and therefore the vector is unavailable for processing at subsequent nodes. Some implementations work around this limitation by simultaneously evaluating the branching function at all nodes as the vector is streamed but this increases the prediction costs from logarithmic to linear which might not be acceptable.

Branching function at internal nodes: Bonsai computes I_k by learning a sparse vector $\boldsymbol{\theta}$ at each internal node such that the sign of $\boldsymbol{\theta}^\top \mathbf{Z}\mathbf{x}$ determines whether point $\mathbf{x}$ should be branched to the node's left or right child. Using more powerful branching functions than the axis-aligned hyperplanes in standard decision trees allows Bonsai to learn shallow trees which can fit in a few KB. Of course, this is not a novel idea, and is insufficient in itself to allow a single, shallow decision tree to make accurate predictions.

Node predictors: Decision trees, random forests and boosted tree ensembles are limited to making constant predictions at just the leaf nodes. This restricts their prediction accuracy when there are very few leaves. In contrast, for a multi-class, multi-label or regression problem with L targets, Bonsai learns matrices $\mathbf{W}_{\hat{D} \times L}$ and $\mathbf{V}_{\hat{D} \times L}$ at both leaf and internal nodes so that each node predicts the vector $\mathbf{W}^\top \mathbf{Z}\mathbf{x} \circ \tanh(\sigma \mathbf{V}^\top \mathbf{Z}\mathbf{x})$. Note that the functional form of the node predictor was chosen as it was found to work well empirically (other forms could be chosen if found to be more appropriate). Further note that $\mathbf{W}$ and $\mathbf{V}$ will reduce to vectors for binary classification, ranking and single-target regression. Bonsai's overall predicted vector is given by (1) and is the sum of the individual vectors predicted by the nodes lying along the path traversed by $\mathbf{x}$. This allows Bonsai to accurately learn non-linear decision boundaries using shallow trees with just a few nodes. Furthermore, path based prediction allows parameter sharing and therefore reduces model size as compared to putting independent classifiers of at least equal complexity in the leaf nodes alone. For instance, a depth 4 Bonsai tree with 15 internal and 16 leaf nodes stores 31 $\mathbf{W}$ and 31 $\mathbf{V}$ matrices with overall predictions being the sum of 4 terms depending on the path taken. If parameters were not shared and each leaf node independently learnt 4 $\mathbf{W}$ and 4 $\mathbf{V}$ matrices to make predictions of at least equal complexity, then a total of $16 \times 4 = 64$ $\mathbf{W}$ and 64 $\mathbf{V}$ matrices would need to be stored thereby exceeding the memory budget. As an implementation detail, note that Bonsai uses the approximation $\tanh(x) \approx x$ if $|x| < 1$ and $\text{signum}(x)$ otherwise in order to avoid floating point computation.

4. Training Bonsai

Notation: Bonsai learns a balanced tree of user specified height h with $2^h - 1$ internal nodes and 2^h leaf nodes. The parameters that need to be learnt include: (a) $\mathbf{Z}$: the sparse projection matrix; (b) $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{2^h-1}]$: the parameters of the branching function at each internal node; and (c) $\mathbf{W} = [\mathbf{W}_1, \ldots, \mathbf{W}_{2^{h+1}-1}]$ and $\mathbf{V} = [\mathbf{V}_1, \ldots, \mathbf{V}_{2^{h+1}-1}]$:

the predictor parameters at each node. Let $\mathbf{\Theta} = [\boldsymbol{\theta}, \mathbf{W}, \mathbf{V}]$ denote a matrix obtained by stacking all the parameters together except for $\mathbf{Z}$. Finally, it is assumed that N training points $\{(\mathbf{x}_i, \mathrm{y}_i)_{i=1}^N\}$ have been provided and that budget constraints $B_{\mathbf{Z}}$ and $B_{\mathbf{\Theta}}$ on the projection matrix and tree parameters have been specified depending on the flash memory available on the IoT device.

Optimization problem: Bonsai's parameters are learnt as

$$\begin{aligned} \min_{\mathbf{Z},\mathbf{\Theta}} \quad & \mathcal{J}(\mathbf{Z},\mathbf{\Theta}) = \frac{\lambda_{\boldsymbol{\theta}}}{2}\mathrm{Tr}(\boldsymbol{\theta}^\top\boldsymbol{\theta}) + \frac{\lambda_{\mathbf{W}}}{2}\mathrm{Tr}(\mathbf{W}^\top\mathbf{W}) \\ & + \frac{\lambda_{\mathbf{V}}}{2}\mathrm{Tr}(\mathbf{V}^\top\mathbf{V}) + \frac{\lambda_{\mathbf{Z}}}{2}\mathrm{Tr}(\mathbf{Z}\mathbf{Z}^\top) \\ & + \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\mathbf{x}_i, \mathbf{y}_i, \mathbf{y}(\mathbf{x}_i); \mathbf{Z}, \mathbf{\Theta}) \\ \text{s. t.} \quad & \|\mathbf{Z}\|_0 \leq B_{\mathbf{Z}}, \|\mathbf{\Theta}\|_0 \leq B_{\mathbf{\Theta}} \end{aligned} \tag{2}$$

where $\mathbf{y}(\mathbf{x}_i)$ is Bonsai's prediction for point $\mathbf{x}_i$ as given in (1) and $\mathcal{L}$ is an appropriately chosen loss function for classification, regression, ranking, *etc*. For instance, $\mathcal{L} = \max(0, 1 - y_i y(\mathbf{x}_i))$ with $y_i \in \{-1, +1\}$ for binary classification and $L = \max_{\mathbf{y}\in\mathbb{Y}}((\mathbf{y}_i - \mathbf{y})^\top\mathbf{y}(\mathbf{x}_i) + 1 - \mathbf{y}_i^\top\mathbf{y})$ with $\mathbb{Y} = \{\mathbf{y}|\mathbf{y} \in \{0,1\}^L, \mathbf{1}^\top\mathbf{y} = 1\}$ and $\mathbf{y}_i \in \mathbb{Y}$ for multi-class classification. It is worth emphasizing that the optimization problem is formulated such that all parameters are learnt jointly subject to the budget constraints. This leads to significantly higher prediction accuracies than if $\mathbf{Z}$ were first learnt independently, say using sparse PCA, and then $\mathbf{\Theta}$ was learnt afterwards (see Section 5).

Algorithm: Optimizing (2) over the space of all balanced trees of height h is a hard, non-convex problem. Tree growing algorithms typically optimize such problems by greedily growing the tree a node at a time starting from the root. Unfortunately, this leads to a suboptimal utilization of the memory budget in Bonsai's case as it is not clear *a priori* how much budget to allocate to each node. For instance, it is not apparent whether the budget should be distributed equally between all nodes or whether the root node should be allocated more budget and, if so, by how much.

Algorithm - Joint learning of nodes: Bonsai therefore learns all node parameters jointly with the memory budget for each node being determined automatically as part of the optimization. The difficulty with joint learning is that a node's ancestors need to be learnt before it can be determined which training points will reach the node. Furthermore, the path traversed by a training point is a sharply discontinuous function of $\boldsymbol{\theta}$ and $\mathbf{Z}$ thereby rendering gradient based techniques ineffective. Various approaches have been proposed in the literature for tackling these difficulties (Jose et al., 2013; Kontschieder et al., 2015; Norouzi et al., 2015; Xu et al., 2013; Ioannou et al., 2016b). Bonsai follows the approach of (Jose et al., 2013) and smooths the objective function by initially allowing points to traverse multiple paths in the tree. In particular, the indicator function $I_k(\mathbf{x})$ is relaxed to $I_{k>1}(\mathbf{x}) = \frac{1}{2}I_j(\mathbf{x})\left(1 + (-1)^{k-2j}\tanh(\sigma_I\boldsymbol{\theta}_j^\top\mathbf{Z}\mathbf{x})\right)$ where $j = \lfloor\frac{k}{2}\rfloor$ is k's parent node in a balanced tree, $I_1(\mathbf{x}) = 1$ and the parameter σ_I controls the fidelity of the approximation. Gradients can now be computed as

$$\nabla_{\boldsymbol{\theta}_l} I_k(\mathbf{x}) = \sigma_I I_k(\mathbf{x}) P_k^l(\mathbf{x})\mathbf{Z}\mathbf{x} \tag{3}$$

$$\nabla_{\mathbf{Z}} I_k(\mathbf{x}) = \sum_l \sigma_I I_k(\mathbf{x}) P_k^l(\mathbf{x})\boldsymbol{\theta}_l\mathbf{x}^\top \tag{4}$$

where $P_k^l(\mathbf{x}) = \delta_k^l((-1)^{C_k(l)} - \tanh(\sigma_I\boldsymbol{\theta}_l^\top\mathbf{Z}\mathbf{x}))$, $\delta_k^l = 1$ if node l is an ancestor of node k and 0 otherwise and $C_k(l) = 1$ if node k is in the right subtree of node l and 0 otherwise. Of course, allowing a point to traverse multiple paths increases prediction costs. Some approaches therefore allow multiple paths during training but select a single path during prediction (Xu et al., 2013; Ioannou et al., 2016b). At each node, a point $\mathbf{x}$ is greedily branched to the child node having the greatest $I_k(\mathbf{x})$. Unfortunately, this can lead to a drop in accuracy as the model learnt during training is different from the one used for prediction.

Bonsai therefore follows an alternative strategy where σ_I is tuned during training to ensure that points gradually start traversing at most a single path as optimization progresses. In particular, σ_I is initialized to a small value, such as 0.01, so as to ensure that $\tanh$ values are not saturated. As optimization progresses, σ_I is gradually increased so that $\tanh$ tends to the signum function and $I_k(\mathbf{x})$ goes back to being an indicator function by the time convergence is reached. This allows Bonsai to directly use the learnt model for prediction and was found to empirically lead to good results.

Algorithm - Gradient descent with iterative hard thresholding: Various gradient based approaches, including those based on alternating minimization, were tried for optimizing (2). A gradient descent based algorithm with iterative hard thresholding (IHT) was empirically found to yield the best solutions. Gradient descent was chosen over stochastic gradient descent as it removed the burden of step size tuning, led to slightly better prediction accuracies while keeping training time acceptable. For instance, training times range from 2 minutes for USPS-2 to 15 minutes for MNIST-2 on a single core of a laptop with an Intel Core i7-3667U processor at 2 GHz with 8 GB of RAM. Stochastic gradient descent could be utilized for larger datasets or if training costs also needed to be minimized. The algorithm proceeds iteratively based on the following gradient and IHT steps in each iteration.

Algorithm - Gradient step: Given feasible $\mathbf{Z}^t$ and $\mathbf{\Theta}^t$ with a feasible allocation of the memory budget to various nodes at time step t, Bonsai applies M updates of gradient descent keeping the support of $\mathbf{Z}$ and $\mathbf{\Theta}$ fixed so that

the budget allocations to nodes remain unchanged and the memory constraints are never violated. The update equations at each time step are

$$\mathbf{Z}^{t+1} = \mathbf{Z}^{t} - \eta_{\mathbf{Z}}^{t} \nabla_{\mathbf{Z}} \mathcal{J}(\mathbf{Z}^{t}, \mathbf{\Theta}^{t})|_{\text{supp}(\mathbf{Z}^{t})} \quad (5)$$

$$\mathbf{\Theta}^{t+1} = \mathbf{\Theta}^{t} - \eta_{\mathbf{\Theta}}^{t} \nabla_{\mathbf{\Theta}} \mathcal{J}(\mathbf{Z}^{t}, \mathbf{\Theta}^{t})|_{\text{supp}(\mathbf{\Theta}^{t})} \quad (6)$$

with step sizes $\eta_{\mathbf{Z}}$ and $\eta_{\mathbf{\Theta}}$ being chosen according to the Armijo rule and $|_{\text{supp}}$ indicating that the gradient was being computed only for the non-zero entries. $M = 5$ and $M = 15$ iterations were found to work well for binary and multi-class classification respectively. This allows Bonsai to decrease the objective function value without changing the budget allocation of various nodes.

Algorithm - IHT step: In order to improve the budget allocation, Bonsai performs a single gradient update with unrestricted support. This violates the memory constraints and Bonsai therefore projects the solution onto the feasible set by retaining the parameters with the largest magnitudes

$$\mathbf{Z}^{t+M+1} = \mathbf{T}_{B_{\mathbf{Z}}}(\mathbf{Z}^{t+M} - \eta_{\mathbf{Z}}^{t+M} \nabla_{\mathbf{Z}} \mathcal{J}(\mathbf{Z}^{t+M}, \mathbf{\Theta}^{t+M}))$$

$$\mathbf{\Theta}^{t+M+1} = \mathbf{T}_{B_{\mathbf{\Theta}}}(\mathbf{\Theta}^{t+M} - \eta_{\mathbf{\Theta}}^{t+M} \nabla_{\mathbf{\Theta}} \mathcal{J}(\mathbf{Z}^{t+M}, \mathbf{\Theta}^{t+M}))$$

where $\mathbf{T}_k$ is an operator returning k of its arguments which have the largest magnitudes while setting the rest to 0. This allows Bonsai to move to another feasible solution with even lower objective function value by improving the memory budget distribution across nodes.

Algorithm - Convergence: In general, projected gradient descent based algorithms might oscillate for non-convex problems. However, (Blumensath & Davies, 2008) prove that for smooth objective functions, gradient descent algorithms with IHT do indeed converge to a saddle point solution. Furthermore, if the objective function satisfies the Restricted Strong Convexity (RSC) property in a local region, then projected gradient descent with IHT will converge to the local minimum in that region (Jain et al., 2014). In practice, it was observed that the algorithm generally converged to a good solution soon and therefore was terminated after $T = 300$ iterations were reached.

Algorithm - Initialization & re-training: $\mathbf{Z}^0$ and $\mathbf{\Theta}^0$ could be set randomly. Prediction accuracy gains of up to 1.5% could be observed if Bonsai was initialized by taking T steps of gradient descent without any budget constraints followed by a hard thresholding step. Further gains of 1.5% could be observed by taking another T steps of gradient descent with fixed support after termination. This helped in fine-tuning Bonsai's parameters once the memory budget allocation had been finalized across the tree nodes.

More details about the optimization can be found in the supplementary material by clicking here.

5. Experiments

Datasets: Experiments were carried out on a number of publically available binary and multi-class datasets including Chars4K (Campos et al., 2009), CIFAR10 (Krizhevsky, 2009), MNIST (LeCun et al., 1998), WARD (Yang et al., 2009), USPS (Hull, 1994), Eye (Kasprowski & Ober, 2004), RTWhale (RTW), and CUReT (Varma & Zisserman, 2005). Binary versions of these datasets were downloaded from (Jose et al., 2013). Bing's L3 Ranking is a proprietary dataset where ground truth annotations specifying the relevance of query-document pairs have been provided on a scale of 0-4. Table 1 lists these datasets' statistics.

Baseline algorithms: Bonsai was compared to state-of-the-art algorithms for resource-efficient ML spanning tree, kNN, SVM and single hidden layer neural network algorithms including Decision Jungles (Shotton et al., 2013; Pohlen), Feature Budgeted Random Forests (BudgetRF) (Nan et al., 2015), Gradient Boosted Decision Tree Ensemble Pruning (Tree Pruning) (Dekel et al., 2016), Pruned Random Forests (BudgetPrune) (Nan et al., 2016), Local Deep Kernel Learning (LDKL) (Jose et al., 2013), Neural Network Pruning (NeuralNet Pruning) (Han et al., 2016) and Stochastic Neighbor Compression (SNC) (Kusner et al., 2014b). The differences between some of these algorithms and Bonsai is briefly discussed in Section 2. Publically available implementations of all algorithms were used taking care to ensure that published results could be reproduced thereby verifying the code and hyper-parameter settings. Note that Bonsai is not compared to deep convolutional neural networks as they have not yet been demonstrated to fit on such tiny IoT devices. In particular, convolutions are computationally expensive, drain batteries and produce intermediate results which do not fit in 2 KB RAM. Implementing them on tiny microcontrollers is still

Table 1: Dataset statistics - the number after the dataset name represents the number of classes so as to distinguish between the binary and multi-class versions of the dataset.

Dataset	# Train	# Test	# Features
L3 Ranking	704,841	123,268	50
Chars4K-2	4,397	1,886	400
CIFAR10-2	50,000	10,000	400
WARD-2	4,503	1,931	1,000
USPS-2	7,291	2,007	256
MNIST-2	60,000	10,000	784
Eye-2	456	196	8,192
RTWhale-2	5,265	5,264	2,000
CUReT-61	4,204	1,403	610
MNIST-10	60,000	10,000	784
Chars4K-62	4,397	1,886	400

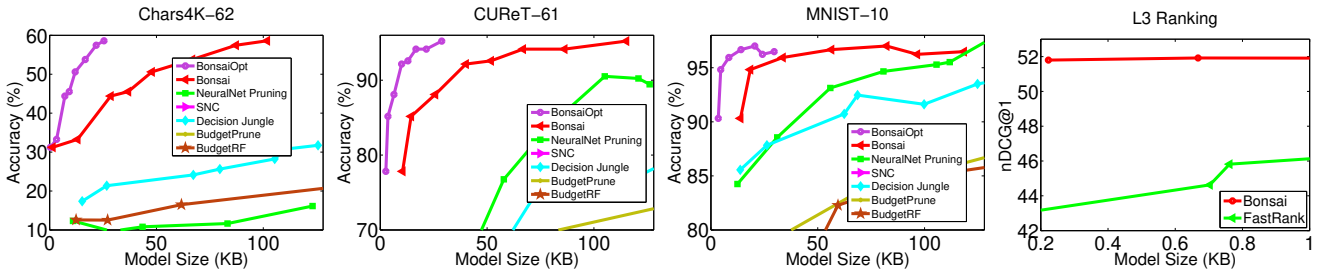

Figure 1: Multi-class & Ranking Datasets - Bonsai dominates over the state-of-the-art resource-efficient ML algorithms by as much as 30.7% on Chars4K-62 and 28.9% on CUReT-61. BonsaiOpt's gains are even higher. Some methods do not appear on the graphs as their accuracies were not high enough to fall within the y-axis limits. Bonsai also dominates Bing's FastRank L3 ranker. Figure best viewed magnified.

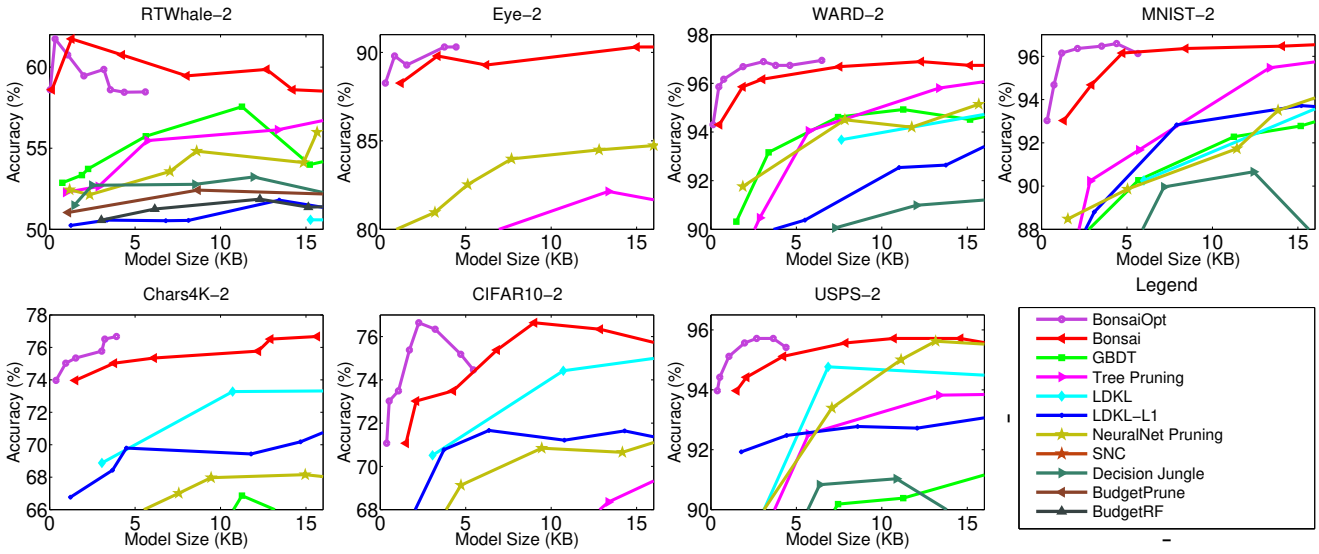

Figure 2: Binary Datasets - Bonsai dominates over state-of-the-art resource-efficient ML algorithms with gains of 8.6% on RTWhale-2 and 8.2% on Eye-2 in the 0-2 KB range. BonsaiOpt's gains are even higher. Figure best viewed magnified.

an open research problem. Bonsai's performance was however compared to that of uncompressed single hidden layer neural networks without convolutions, Gradient Boosted Decision Trees (GBDT), kNN classifiers and RBF-SVMs.

Hyper-parameters: The publically provided training set for each dataset was subdivided into 80% for training and 20% for validation. The hyper-parameters of all algorithms were tuned on the validation set. Once the hyper-parameters had been fixed, the algorithms were trained on the full training set and results were reported on the publically available test set.

Evaluation: IoT applications would like to maximize their prediction accuracies using the best model that might fit within the available flash memory while minimizing their prediction times and energies. Accuracies of all algorithms are therefore presented for a range of model sizes. Some of the algorithms were implemented on the Uno and their prediction times and energies were compared to Bonsai's.

Implementation: Results are presented throughout for an unoptimized implementation of Bonsai for a fair comparison with the other methods. For instance, 4 bytes were used to store floating point numbers for all algorithms, all floating point operations were simulated in software, *etc*. However, results are also presented for an optimized implementation of Bonsai, called BonsaiOpt, where numbers were stored in a 1 byte fixed point format, $\tanh$ was approximated, all floating point operations were avoided, *etc*.

Comparison to uncompressed methods: The results in Tables 2 and 3 demonstrate that Bonsai's prediction accuracies could compete with those of uncompressed kNN, GBDT, RBF-SVM and neural network classifiers with significantly larger model sizes. On RTWhale-2, Chars4K-62

Table 2: Binary Datasets - Bonsai can sometimes outperform uncompressed methods with significantly larger models.

Dataset	Bonsai (%)		GBDT (%)	kNN (%)	RBF-SVM (%)	NeuralNet (%)
	2KB	16KB				
RTWhale-2	61.74	61.74	56.95 (1172 KB)	51.75 (41143 KB)	53.82 (39905 KB)	52.26 (3910 KB)
Chars4K-2	74.28	76.67	72.38 (625 KB)	67.28 (6870 KB)	75.60 (6062 KB)	72.53 (314 KB)
Eye-2	88.26	90.31	83.16 (234 KB)	76.02 (14592 KB)	93.88 (7937 KB)	90.31 (6402 KB)
WARD-2	95.85	96.89	97.77 (1172 KB)	94.98 (17590 KB)	96.42 (7222 KB)	92.75 (3914 KB)
CIFAR10-2	73.02	76.64	77.19 (1562 KB)	73.70 (78125 KB)	81.68 (63934 KB)	75.90 (314 KB)
USPS-2	94.42	95.72	95.91 (234 KB)	96.70 (7291 KB)	96.86 (1660 KB)	95.86 (504 KB)
MNIST-2	94.38	96.47	98.36 (1172 KB)	96.90 (183750 KB)	98.08 (35159 KB)	98.33 (3070 KB)

Table 3: Multi-class Datasets - Bonsai can sometimes outperform uncompressed methods with significantly larger models.

Dataset	Bonsai (%)	GBDT (%)	kNN (%)	RBF-SVM (%)	NeuralNet (%)
Chars4K-62	58.59 (101 KB)	43.34 (9687 KB)	39.32 (6833 KB)	48.04 (7738 KB)	55.35 (1266 KB)
CUReT-61	95.23 (115 KB)	90.81 (2383 KB)	89.81 (10037 KB)	97.43 (8941 KB)	95.51 (1310 KB)
MNIST-10	97.01 (84 KB)	97.90 (5859 KB)	94.34 (183984 KB)	97.30 (39084 KB)	98.44 (4652 KB)

Table 4: The effect of Bonsai's components - Performing sparse PCA independently before training is not as effective as Bonsai's joint optimization of the projection matrix.

Method	Accuracy (%)	Model size (KB)
Bonsai with random initialization and without re-training	74.12	16
Bonsai without re-training	75.19	16
Bonsai	76.67	16
Bonsai with sparse PCA	58.32	16
Tree Pruning with sparse PCA	63.57	16
Decision Jungle with sparse PCA	61.67	16
RBF-SVM with sparse PCA	71.10	136

and Chars4K-2, Bonsai's accuracies were higher than all other methods by 4.8%, 3.2% and 1.1% while its model size was lower by 977x, 13x and 157x respectively. Bonsai's accuracies were lower by 1.0% - 5.0% on the other datasets with model size gains varying from 55x to 3996x. Note that, while BonsaiOpt's accuracies were similar to Bonsai's, its model sizes would be even lower.

Comparison to resource-efficient ML algorithms: The results in Figures 2 and 3 demonstrate that Bonsai's prediction accuracies dominate those of state-of-the-art resource-efficient ML algorithms for all model sizes. In fact, Bonsai could outperform all other algorithms, including tree algorithms by as much as 30.7% on Char4K-62 and 28.9% on CUReT-61 for a given model size. For binary datasets, the largest gains were observed in the 0-2 KB regime including 8.6% on RTWhale-2 and 8.2% on Eye-2. Of course, BonsaiOpt's gains were even higher on both binary and multi-class datasets. These results validate Bonsai's model, showing it to be accurate and compact and demonstrate that Bonsai's optimization algorithm yields good solutions.

L3 ranking: Bonsai was shown to generalise to other resource-constrained scenarios beyond IoT by ranking documents in response to queries on Bing. Bonsai was trained by replacing the classification gradients with rank-sensitive gradients approximating nDCG (Burges, 2010). As can be seen in Figure 1, using a 300 byte model, Bonsai could outperform Bing's FastRank L3 ranker by 8.3%. In fact, Bonsai could achieve almost the same ranking accuracy as FastRank but with a 660x smaller model.

Prediction on the Arduino Uno: Table 5 presents the prediction costs per test point for the highest accuracy models with size less than 2 KB for a few methods that were implemented on the Arduino Uno. The BonsaiOpt model was a more efficient implementation of the chosen Bonsai model. The results indicate that BonsaiOpt could be significantly more accurate, faster and energy-efficient as compared to other algorithms including an unoptimized linear classifier. Transmitting the test feature vector to the cloud, whenever possible, and running uncompressed GBDT might sometimes yield higher accuracies but would also consume 47x-497x more energy which might not be feasible.

Bonsai's components: The contribution of Bonsai's components on the Chars4K-2 dataset is presented in Table 4. Modest reductions in accuracy were observed without proper initialization or re-training. Learning a projection matrix independently via sparse PCA before training reduced accuracy significantly as compared to Bonsai's joint training of the projection matrix and tree parameters. Other tree and uncompressed methods also did not benefit much by training in the sparse PCA space.

Table 5: Prediction costs per test point on the Arduino Uno with the highest accuracy model of size less than 2 KB – The BonsaiOpt model was a more efficient implementation of the chosen Bonsai model. BonsaiOpt was significantly more accurate, faster and energy-efficient than all other methods. Transmitting the test feature vector to the cloud, whenever possible, and running uncompressed GBDT might sometimes yield higher accuracies but would also consume 47x-497x more energy which might not be feasible in many IoT applications.

Dataset		BonsaiOpt	Bonsai	Linear	LDKL	NeuralNet Pruning	Cloud GBDT
Eye-2	Model Size (KB)	0.30	1.20	2.00	1.88	1.96	234.00
	Accuracy (%)	88.78	88.26	80.10	66.33	80.45	83.16
	Prediction Time (ms)	10.75	12.26	15.13	15.80	15.48	2186.59
	Prediction Energy (mJ)	2.64	3.01	3.72	3.89	3.81	1311.95
RTWhale-2	Model Size (KB)	0.33	1.32	0.86	1.00	1.17	1172.00
	Accuracy (%)	60.94	61.74	50.76	50.24	52.44	56.95
	Prediction Time (ms)	5.24	7.11	4.68	6.16	8.86	521.27
	Prediction Energy (mJ)	1.29	1.75	1.15	1.52	2.18	312.76
Chars4K-2	Model Size (KB)	0.50	2.00	1.56	1.95	1.96	625.00
	Accuracy (%)	74.71	74.28	51.06	67.29	63.90	72.38
	Prediction Time (ms)	4.21	8.55	7.39	8.61	14.09	160.40
	Prediction Energy (mJ)	1.03	2.10	1.81	2.13	3.48	63.52
WARD-2	Model Size (KB)	0.47	1.86	1.99	1.99	1.91	1172.00
	Accuracy (%)	95.70	95.86	87.57	89.64	91.76	97.77
	Prediction Time (ms)	4.85	8.13	7.48	9.99	14.22	293.13
	Prediction Energy (mJ)	1.18	1.99	1.84	2.47	3.49	116.08
CIFAR10-2	Model Size (KB)	0.50	1.98	1.56	1.88	1.96	1562.00
	Accuracy (%)	73.05	73.02	69.11	67.54	67.01	77.19
	Prediction Time (ms)	4.55	8.16	7.73	8.12	13.87	160.40
	Prediction Energy (mJ)	1.11	2.01	1.90	2.00	3.43	63.52
USPS-2	Model Size (KB)	0.50	2.00	1.02	1.87	2.00	234.00
	Accuracy (%)	94.42	94.42	83.11	91.96	88.68	95.91
	Prediction Time (ms)	2.93	5.57	4.15	5.59	9.51	83.45
	Prediction Energy (mJ)	0.71	1.37	1.02	1.37	2.33	33.05
MNIST-2	Model Size (KB)	0.49	1.96	1.93	1.87	1.90	1172.00
	Accuracy (%)	94.28	94.38	86.16	87.01	88.65	98.36
	Prediction Time (ms)	5.17	8.90	6.72	8.72	14.67	264.96
	Prediction Energy (mJ)	1.27	2.18	1.65	2.16	3.59	104.92

6. Conclusions

This paper proposed an alternative IoT paradigm, centric to the device rather than the cloud, where ML models run on tiny IoT devices without necessarily connecting to the cloud thereby engendering local decision making capabilities. The Bonsai tree learner was developed towards this end and demonstrated to be fast, accurate, compact and energy-efficient at prediction time. Bonsai was deployed on the Arduino Uno board as it could fit in a few KB of flash, required only 70 bytes of writable memory for binary classification and 500 bytes for a 62 class problem, handled streaming features and made predictions in milliseconds taking only milliJoules of energy. Bonsai's prediction accuracies could be as much as 30% higher as compared to state-of-the-art resource-efficient ML algorithms for a fixed model size and could even approach and outperform those of uncompressed models taking many MB of RAM. Bonsai achieved these gains by developing a novel model based on a single, shallow, sparse tree learnt in a low-dimensional space. Predictions made by both internal and leaf nodes and the sharing of parameters along paths allowed Bonsai to learn complex non-linear decision boundaries using a compact representation. Bonsai's code is available from (BonsaiCode) and is part of Microsoft's ELL machine learning compiler for IoT devices.

Acknowledgements

We are grateful to Yeshwanth Cherapanamjeri, Ofer Dekel, Chirag Gupta, Prateek Jain, Ajay Manchepalli, Nagarajan Natarajan, Praneeth Netrapalli, Bhargavi Paranjape, Suresh Parthasarathy, Vivek Seshadri, Rahul Sharma, Harsha Vardhan Simhadri, Manish Singh and Raghavendra Udupa for many helpful discussions and feedback.

References

The right whale dataset. `https://www.kaggle.com/c/whale-detection-challenge/data`.

Ba, J. and Caruana, R. Do deep nets really need to be deep? In *NIPS*, 2014.

Blumensath, T. and Davies, M. E. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14(5-6), 2008.

BonsaiCode. Code for Bonsai. `http://www.manikvarma.org/code/Bonsai/download.html`.

Breiman, L. Random forests. *ML*, 2001.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. Classification and regression trees. In *CRC press*, 1984.

Bucilua, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

Burges, C. J. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581), 2010.

Campos, T. E., Babu, B. R., and Varma, M. Character recognition in natural images. In *VISAPP(2)*, 2009.

Dekel, O., Jacobbs, C., and Xiao, L. Pruning decision forests. In *Personal Communications*, 2016.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification*. John Wiley and Sons, 2nd edition, 2002.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

Hsieh, C. J., Si, S., and Dhillon, I. Fast prediction for large-scale kernel machines. In *NIPS*, 2014.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016.

Hull, J. J. A database for handwritten text recognition research. *IEEE PAMI*, 16, 1994.

Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, abs/1602.07360, 2016.

Ioannou, Y., Robertson, D., Cipolla, R., and Criminisi, A. Deep roots: Improving cnn efficiency with hierarchical filter groups. *arXiv preprint arXiv:1605.06489*, 2016a.

Ioannou, Y., Robertson, D., Kontschieder, D. Zikicand P., Shotton, J., Brown, M., and Criminisi, A. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016b.

Jain, P., Tewari, A., and Kar, P. On iterative hard thresholding methods for high-dimensional m-estimation. In *NIPS*, 2014.

Joly, A., Schnitzler, F., Geurts, P., and Wehenkel, L. L1-based compression of random forest models. In *ESANN*, 2012.

Jose, C., Goyal, P., Aggrwal, P., and Varma, M. Local deep kernel learning for efficient non-linear svm prediction. In *ICML*, 2013. `https://manikvarma.github.io/code/LDKL/download.html`.

Kasprowski, P. and Ober, J. Eye movement in biometrics. In *eccv*, 2004.

Kontschieder, P., Fiterau, M., Criminisi, A., and Bulo, S. R. Deep neural decision forests. In *ICCV*, 2015.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Kulkarni, V. Y. and Sinha, P. K. Pruning of random forest classifiers: A survey and future directions. In *ICDSE*, 2012.

Kusner, M. J., Chen, W., Zhou, Q., Xu, Z. E., Weinberger, K. Q., and Chen, Y. Feature-cost sensitive learning with submodular trees of classifiers. In *AAAI*, 2014a.

Kusner, M. J., Tyree, S., Weinberger, K. Q., and Agrawal, K. Stochastic neighbor compression. In *ICML*, 2014b. `http://mkusner.github.io/#code`.

Le, Q., Sarlós, T., and Smola, A. Fastfood-approximating kernel expansions in loglinear time. In *ICML*, 2013.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, S. Z. *Markov Random Field Modeling in Image Analysis*. Springer-Verlag, 2001.

Li, Z., Yang, T., Zhang, L., and Jin, R. Fast and accurate refined nyström-based kernel svm. In *AAAI*, 2016.

Meunier, F., Wood, A., Weiss, K., Huberty, K., Flannery, S., Moore, J., Hettenbach, C., and Lu, B. The internet of things is now connecting the real economy. Technical report, Morgan Stanley, 2014.

Murthy, S. K., Kasif, S., and Salzberg, S. A system for induction of oblique decision trees. *JAIR*, 2, 1994.

Nan, F., Wang, J., and Saligrama, V. Feature-budgeted random forest. In *ICML*, 2015. `http://sites.bu.edu/data/code-4/`.

Nan, F., Wang, J., and Saligrama, V. Pruning random forests for prediction on a budget. In *NIPS*, 2016.

Norouzi, M., Collins, M., Johnson, M. A., Fleet, D. J., and Kohli, P. Efficient non-greedy optimization of decision trees. In *NIPS*, 2015.

Pohlen, T. LibJungle - Decision Jungle Library. `https://bitbucket.org/geekStack/libjungle`.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

Rokach, L. and Maimon, O. *Data mining with decision trees: theory and applications*. World scientific, 2014.

Shankar, S., Robertson, D., Ioannou, Y., Criminisi, A., and Cipolla, R. Refining architectures of deep convolutional neural networks. In *CVPR*, 2016.

Sherali, H. D., Hobeika, A. G., and Jeenanunta, C. An optimal constrained pruning strategy for decision trees. *INFORMS Journal on Computing*, 21(1), 2009.

Shotton, J., Sharp, T., Kohli, P., Nowozin, S., Winn, J., and Criminisi, A. Decision jungles: Compact and rich models for classification. In *NIPS*, 2013.

Utgoff, P. E. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1(4), 1989.

Varma, M. and Zisserman, A. A statistical approach to texture classification from single images. *IJCV*, 62(1–2):61–81, 2005.

Viola, P. and Jones, M. J. Robust real-time face detection. *IJCV*, 57(2), 2004.

Wang, J., Trapeznikov, K., and Saligrama, V. Efficient learning by directed acyclic graph for resource constrained prediction. In *NIPS*, 2015.

Wang, W., Chen, C., Chen, W., Rai, P., and Carin, L. Deep distance metric learning with data summarization. In *ECML PKDD*, 2016.

Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016.

Xu, Z., Weinberger, K. Q., and Chapelle, O. The greedy miser: Learning under test-time budgets. In *ICML*, 2012.

Xu, Z. E., Kusner, M. J., Weinberger, K. Q., and Chen, M. Cost-sensitive tree of classifiers. In *ICML*, 2013.

Yang, J., Li, Y., Tian, Y., Duan, L., and Gao, W. Group-sensitive multiple kernel learning for object categorization. In *ICCV*, 2009.

Yang, Z., Moczulski, M., Denil, M., Freitas, N., Smola, A. J., Song, L., and Wang, Z. Deep fried convnets. *ICCV*, 2015.

Zhang, Y. and Huei-chuen, H. Decision tree pruning via integer programming. Technical report, 2005.