TERNARY HYBRID NEURAL-TREE NETWORKS FOR HIGHLY CONSTRAINED IOT APPLICATIONS

Dibakar Gope [1] **Ganesh Dasika** [1] **Matthew Mattina** [1]

ABSTRACT

Machine learning-based applications are increasingly prevalent in IoT devices. The power and storage constraints of these devices make it particularly challenging to run modern neural networks, limiting the number of new applications that can be deployed on an IoT system. A number of compression techniques have been proposed, each with its own trade-offs. We propose a hybrid network which combines the strengths of current neural- and tree-based learning techniques in conjunction with ternary quantization, and show a detailed analysis of the associated model design space. Using this hybrid model we obtained a 11.1% reduction in the number of computations, a 52.2% reduction in the model size, and a 30.6% reduction in the overall memory footprint over a state-of-the-art keyword-spotting neural network, with negligible loss in accuracy.

1 INTRODUCTION

Machine learning algorithms, and neural networks (NNs) in particular, are increasingly deployed in Internet-of-Things (IoT) devices. Popular applications include speech interfaces in smart-home devices, predictive maintenance for commercial and industrial machines, health-monitoring in wearables, etc. However, due to the energy, power, storage, and compute limitations of highly-constrained IoT devices, they are frequently limited to simplistic tasks, while more sophisticated requests are off-loaded to a more capable device or to a server. In addition to being computationally constrained, IoT devices frequently have very little available SRAM, tend to be "always-on", and are often connected to constrained power sources. Because of these constraints, reducing the computation and storage required by ML models for IoT applications is of paramount importance in order to ensure a longer battery life.

To enable this computation and size compression of NN models, one particularly effective technique has been the use of depthwise-separable (DS) convolutional layers. We see these layers being used in a large, image classification application (Howard et al., 2017) and also on a ubiquitous keyword-spotting application (Zhang et al., 2017), showing state-of-the-art or near-state-of-the-art accuracy in both cases.

While DS convolutional layers have been transformative, even further compression is still valuable in order to target the most constrained microcontrollers or to make a wider range of applications available on IoT devices. Recent work has shown that this might be possible through the use of binary- and ternary-weight networks (Rastegari et al., 2016; Alemdar et al., 2016; Li & Liu, 2016; Tschannen et al., 2018). In such networks, multiplications are replaced with additions, relying on binary (-1,1) or ternary (-1,0,1) weight matrices. This enables more energy-efficient and faster network architectures with fewer expensive multiplications but at the cost of modest to significant drop in prediction accuracy when compared to their full-precision counterparts. Recent work on StrassenNets (Tschannen et al., 2018) presents a more mathematically profound way to approximate matrix multiplication computation (and, in turn, convolutions) using mostly ternary weights and a few full-precision weights. It demonstrates no loss in predictive performance when compared to full-precision models. The effectiveness of StrassenNets is quite variable, however, depending on the neural network architecture. We observe, for example, that while *strassenifying* is effective in reducing the model size of DS convolutional layers, this might come with a prohibitive increase in the number of addition operations, reducing the energy efficiency of neural network inference.

The interest in reducing complexity has also expanded beyond neural networks. Recent research around tree-based learning algorithms has shown immense potential to perform classification and regression in the IoT setting with a significantly lower computation and storage budget than their neural counterparts, while maintaining acceptable model accuracy. More specifically, Bonsai decision trees (Kumar et al., 2017) make this possible by learning a single, shal-

[1]Arm ML Research Lab. Correspondence to: Dibakar Gope <dibakar.gope@arm.com>.

Proceedings of the 2^{nd} SysML Conference, Palo Alto, CA, USA, 2019.

low and sparse tree to reduce model size but with powerful nodes for accurate prediction. It uses more powerful branching functions than the axis-aligned hyperplanes in standard decision trees. This is coupled with non-linear predictions made by internal and leaf nodes on a single, shallow decision tree learned in a low-dimensional space. Combining these ideas allows Bonsai trees to learn complex non-linear decision boundaries using a compact representation. While the results in (Kumar et al., 2017) show promising results for smaller applications, our observations were that the techniques do not scale when extended to a more complex use-case, showing poor prediction accuracy even with a large model footprint.

We now, therefore, have two ways of compressing models, each with its own advantages and limitations:

- StrassenNets is effective at reducing the size of a neural network model but at a potentially significant cost of more addition operations.
- Bonsai tree is effective at reducing the number of operations for simple models but cannot easily be extended to larger models.

Motivated by these observations, we propose a hybrid network architecture capable of giving start-of-the-art accuracy levels, while requiring a fraction of the model parameters and considerably fewer operations per inference. The hybrid architecture makes this possible by leveraging a few neural DS convolutional layers for feature extraction and then relying on a compute-efficient, shallow Bonsai decision tree to perform the classification. It then applies StrassenNets over the overall neural-tree network to reduce its memory footprint significantly thus enabling a compact compute-efficient architecture. We apply this hybrid architecture to a representative IoT application keyword-spotting. The hybrid network achieves a 98.89% reduction in multiplications, a 12.22% reduction in additions (overall 11.1% reduction in number of operations), a 52.2% reduction in model size, and a 30.6% reduction in overall memory footprint over a state-of-the-art keyword-spotting neural model. The hybrid network accomplishes this with a very minimal loss in accuracy of 0.27%. The final network is well within the constrained compute budget of typical microcontrollers.

The remainder of the paper is organized as follows. Section 2 elaborates on the incentives behind this hybrid network architecture for microcontrollers and provides a brief overview of the neural and tree-based learning algorithms that it attempts to hybridize along with our observations of applying them to the keyword-spotting application. Failing to find a good balance between accuracy and computation costs shifts our focus towards designing a hybrid neural-tree network. Section 3 describes our hybrid network. Section 4 presents results. Section 5 compares our hybrid network against prior works and and Section 6 concludes the paper.

2 Model Compression Limitations for an IoT Application

2.1 StrassenNets

Given two 2×2 matrices, Strassen's matrix multiplication algorithm computes their product using 7 multiplications instead of the 8 required with a naïve implementation of matrix multiplication. It essentially converts the matrix multiplication operation to a 2-layer sum-product network (SPN) computation as shown below:

$$vec(C) = W_c[(W_b vec(B)) \odot (W_a vec(A))] \quad (1)$$

W_a, $W_b \in K^{r \times n^2}$ and $W_c \in K^{n^2 \times r}$ are ternary matrices with $K \in \{-1, 0, 1\}$, $vec(A)$ and $vec(B)$ are the vectorization of the two input square matrices A, $B \in R^{n \times n}$; and $vec(C)$ represents the vectorized form of the product $A \times B$. $\odot$ denotes the element-wise product. The $(W_b vec(B))$ and $(W_a vec(A))$ of the SPN compute r intermediate factors each from additions, and/or subtractions of elements of A and B realized by the two associated ternary matrices W_a and W_b respectively. The two generated r-length intermediate factors are then element-wise multiplied to produce the r-length $(W_b vec(B)) \odot (W_a vec(A))$. The outmost ternary matrix W_c later combines the r elements of the product $(W_b vec(B)) \odot (W_a vec(A))$ in different ways to generate the vectorized form of product matrix C. Therefore, the width of the hidden layer of the SPN r decides the number of multiplications required for the Strassen's matrix multiplication algorithm. For example, given two 2×2 matrices, ternary matrices W_a and W_b with sizes of 7×4 can multiply them using 7 multiplications and 36 additions. It is important to note that Strasssen's algorithm requires a hidden layer with 7 units here to compute the exact product matrix that a naïve matrix multiplication algorithm can obtain using 8 multiplications.

Building on top of Strassen's matrix multiplication algorithm, the StrassenNets work (Tschannen et al., 2018) instead realizes approximate matrix multiplications in DNN layers using fewer hidden layer units compared to the standard Strassen's algorithm to achieve the exact product matrix. StrassenNets makes this possible by training a SPN-based DNN framework end-to-end to learn the ternary weight matrices from the training data. The learned ternary weight matrices can then approximate the otherwise exact matrix multiplications of the DNN layers with significantly fewer multiplications than Strassen's algorithm. The approximate transforms realized by the SPNs, adapted to the DNN architecture and application data under consideration, can

Table 1. Test accuracy along with the number of multiplications, additions, operations and model size for state-of-the-art DS-CNN and strassenified DS-CNN (ST-DS-CNN) on KWS. r is the hidden layer width of a strassenified convolution layer, c_{out} is the number of output channels of the corresponding convolution layer.

NETWORK	ACC. (%)	MULS, ADDS	MACS	OPS	MODEL SIZE
DS-CNN	94.4	-	2.7M	2.7M	22.07KB
ST-DS-CNN ($r = 0.5c_{out}$)	93.18	0.05M, 2.85M	-	2.9M	16.23KB
ST-DS-CNN ($r = 0.75c_{out}$)	94.09	0.06M, 4.09M	-	4.15M	19.26KB
ST-DS-CNN ($r = c_{out}$)	94.03	0.07M, 5.32M	-	5.39M	22.29KB
ST-DS-CNN ($r = 2c_{out}$)	94.74	0.11M, 10.25M	-	10.36M	34.42KB

enable precise control over the number of multiplications and additions required per inference, creating an opportunity to tune DNN models to strike an optimal balance between accuracy and computational complexity. The success of StrassenNets in achieving significant compression for 3×3 convolutions (Tschannen et al., 2018) and increasing visibility of DS convolutions in resource-constrained IoT networks (Zhang et al., 2017) inspired us to apply StrassenNets over already compute-efficient IoT networks dominated with DS layers to reduce their computational costs and model size even further. Further compression of DS layers will not only enable more energy-efficient IoT networks leading to longer lasting batteries, but also will open up the opportunities for more complex IoT use-cases to fit in the limited memory budget of tiny microcontrollers.

As a representative benchmark for exploring different compression algorithms, we have chosen a keyword spotting (KWS) model from (Zhang et al., 2017). The DS convolution-based model (DS-CNN) shown in (Zhang et al., 2017) has state-of-the-art accuracy on the realistic Google speech commands dataset (Warden, 2018). Furthermore, when compared to traditional CNN or other RNN approaches, the model size is smaller and the number of operations required per inference is fewer as well.

2.1.1 *StrassenNets for KWS*

We observe that although strassenifying DS convolution layers reduces multiplications significantly as expected, it increases additions considerably in order to achieve an accuracy comparable to that of the state-of-the-art DS-CNN. Table 1 captures our observation with strassenifying DS layers of the uncompressed DS-CNN KWS model. Multiply, addition, and multiply-accumulate (MAC) operations typically incur similar execution latencies in modern microprocessors, but different models have different ratios of these operations. They are, therefore, counted individually and aggregated in the "Ops" column. The strassenified network with the $r = 0.75c_{out}$ configuration incurs a negligible loss in accuracy of 0.31% while reducing multiplications by 97.7% but increasing additions by 51.4% (2.7M MACs of DS-CNN vs. 0.06M multiplications and 4.09M additions of ST-DS-CNN with $r = 0.75c_{out}$). That means the strassenified network with $r = 0.75c_{out}$ configuration actually increases the number of total operations to 4.15M when compared to 2.7M operations in the uncompressed DS-CNN network. As shown in Table 1, a number of potential values for the hidden layer width (r) were explored and a value of at least $0.75c_{out}$ was needed to achieve a comparable accuracy to that of the full-precision DS-CNN model. Using fewer hidden units ($r = 0.5c_{out}$) than this incurs an accuracy loss of 1.22%, whereas wider strassenified hidden layers ($r = 2c_{out}$) recover the negligible accuracy loss of the $r = 0.75c_{out}$ configuration. For sufficiently large r values, the strassenified network can even out-perform the uncompressed DS-CNN model in accuracy, albeit with a significant increase (about 280% for $r = 2c_{out}$) in the number of additions than the DS-CNN model.

2.1.2 *Compute inefficiency of StrassenNets for models with DS convolutions*

It is important to note here that although the number of additions does increase marginally with strassenifying standard 3×3 or 5×5 convolutional layers (Tschannen et al., 2018), that trend does not hold true with strassenifying DS layers. This stems from the fact that 1×1 pointwise convolutions dominate the compute bandwidth of a neural network with DS layers (Zhang et al., 2017; Howard et al., 2017) and strassenifying a 1×1 pointwise convolution requires executing two equal-sized (for $r = c_{out}$) 1×1 convolution operations (with ternary weight filters) in place of the standard 1×1 convolution. This results in a significant increase in additions in comparison to the execution of the standard 1×1 convolution. In contrast to that, a 3×3 strassenified convolution with $r = c_{out}$ instead requires executing a 3×3 convolution and a 1×1 convolution with ternary weight filters, causing a marginal increase in additions compared to the execution of the standard 3×3 convolution (Tschannen et al., 2018). This overhead of addition operations with strassenified DS convolutions increases in proportion to the width of the strassenified hidden layers, i.e. to the size of the ternary convolution operations, as observed in Table 1. As a result, a strassenified DS convolution layer may incur enough overhead to offset the benefit of strassenifying a DS convolution layer.

While (Tschannen et al., 2018) demonstrates better tradeoffs when strassenifying ResNet-18 architecture, this is not likely to continue once a larger network dominated with DS convolutions (e.g. MobileNets (Howard et al., 2017)) is strassenified. (Tschannen et al., 2018) observes the ResNet-18 architecture with strassenified 3×3 convolutions to

achieve comparable accuracy to that of the uncompressed ResNet-18 on the ImageNet dataset with $r = 2c_{out}$ configuration while requiring a modest (29.63%) increase in additions. A strassenified MobileNets with $r = 2c_{out}$ configuration for the DS layers will give rise to about a 300% increase in additions over the uncompressed MobileNets architecture. This increase in computational costs associated with strassenified DS convolutions in conjunction with the high accuracy and low latency requirements of IoT applications call for a model architecture exploration that can leverage the compute efficiency of DS layers and model size reduction of strassenified convolutions owing to their ternary weights while maintaining acceptable or no increase in additions. As tree-based learning techniques from recent work (Kumar et al., 2017) exhibit accuracy on par with neural models while requiring significantly fewer MAC operations, this motivates us to explore the model accuracy and compute-efficiency of these tree-based techniques for representative IoT applications.

2.2 Bonsai Decision Trees

Piece-wise axis-aligned decision boundaries coupled with constant predictions at just the leaf nodes restrict the prediction accuracy of typical tree models when compared to that of their neural counterparts. Tree ensembles are commonly used to improve the accuracy, but they can occupy too large a memory footprint for typical microcontrollers. Recent work on tree models attempt to learn more complex decision boundaries by moving away from learning axis-aligned hyperplanes at internal nodes and constant predictors at the leaves. Bonsai decision trees (Kumar et al., 2017) fall into this paradigm. Using more powerful branching functions than the axis-aligned hyperplanes of standard decision trees in conjunction with non-linear prediction scores in both internal and leaf nodes allows Bonsai to learn a single, shallow tree that can achieve accuracy on par with small neural-based models. For a multi-class classification problem with L targets, Bonsai learns matrices $W_{\hat{D}\times L}$ and $V_{\hat{D}\times L}$ at both leaf and internal nodes so that each node now predicts a non-linear prediction score $W^\top Zx \circ tanh(\sigma V^\top Zx)$. Bonsai reduces model size by projecting each D-dimensional input feature vector x into a low $\hat{D}$-dimensional space using a projection matrix $Z_{\hat{D}\times D}$ in which the tree is learned. Once an input feature is projected to a low-dimensional space, Bonsai adds the individual node predictions along the path traversed by the projected input to derive the overall prediction. Owing to a single, shallow tree with powerful nodes and branching functions learned in a low-dimensional space, Bonsai can achieve impressive computation reduction over a typical DNN, while preserving DNN-level accuracy for very small models.

Table 2. Test accuracies for DS-CNN and Bonsai tree variants on KWS. $\hat{D}$ = projected dimension, T = depth of tree.

NETWORK	ACC. (%)	MACS	OPS	MODEL SIZE
DS-CNN	94.4	2.7M	2.7M	22.07KB
BONSAI ($\hat{D}$=64, T=2)	80.20	0.02M	0.02M	140.75KB
BONSAI ($\hat{D}$=64, T=4)	82.92	0.04M	0.04M	287.75KB
BONSAI ($\hat{D}$=128, T=2)	81.56	0.04M	0.04M	281.5KB
BONSAI ($\hat{D}$=128, T=4)	84.38	0.07M	0.07M	575.5KB

2.2.1 *Bonsai tree for KWS*

When applied to the KWS application, Bonsai shows poor prediction accuracy even with a significantly large tree with many internal and leaf nodes. As shown in Table 2, Bonsai trees achieve poor accuracies, saturating at about 84%, even when the tree architecture is scaled up with wider projection layers, more tree nodes and trained for longer[1]. Furthermore, a major fraction of the model size (e.g. 69.63% of Bonsai tree with $\hat{D}$=64 and T=2) is attributed to the fully-connected (FC) layer used in projecting the incoming input data to low-dimensional space. Clearly, weight quantization[2] and aggressive pruning will reduce the model size further, as described in (Kumar et al., 2017), however it will not be able to recover the significant accuracy loss of Bonsai trees when compared to that of the neural models for KWS (e.g. DS-CNN). It is worth emphasizing that although Bonsai occupies a large memory footprint, its computational costs are very low in comparison to those neural models.

2.2.2 *The limitations of Bonsai trees for KWS*

While (Kumar et al., 2017) shows the effectiveness of Bonsai trees for the applications they considered, our results show that for more complex applications, there might be a fundamental limitation in the expressiveness of Bonsai trees. More specifically, the simple projection matrix that is made of a FC layer in a Bonsai tree is likely not effective in compressing KWS's initial speech inputs to extract rich useful features. This observation is further corroborated by prior works (Zhang et al., 2017; Arik et al., 2017; Sainath & Parada, 2015) on designing state-of-the-art neural networks for small-footprint KWS that leverages convolutional layers instead to compress complex speech inputs of KWS applications to extract a few rich, meaningful features.

Based on these results, we can conclude that StrassenNets and Bonsai trees, while effective at reducing model complexity for some models, have limitations when applied to a representative IoT application. This motivates the use

[1] Bonsai trees in Table 2 are trained significantly longer than the other networks in this work. The learning rate is initially chosen as 0.001, and later gradually reduced after every 100 epochs.

[2] Each Bonsai tree weight in Table 2 requires 4 bytes to store.

of a potential hybrid model - one that can use the feature extraction capabilities of a convolutional network while also reducing the amount of compute required for subsequent classification of these features. This hybrid model proposed in this paper exploits Bonsai tree's strength as a compute-efficient classifier given rich features and couples that with StrassenNets to achieve significant reduction in model size.

3 Hybrid Neural-Tree Architecture

We propose a hybrid neural-tree architecture that can leverage a few convolutional layers to extract the minimal set of necessary local features, and then can rely on powerful branching functions and non-linear Bonsai tree nodes to find global correlation between features and to perform the required classification. As the tree section of the hybrid network is comparatively compute-efficient in terms of MAC operations, use of it to find the global interaction between local features and classifying the voice commands should result in an overall reduction of computational costs compared to a neural-only state-of-the-art network for KWS without compromising its accuracy. DS convolutional layers are used in particular for feature extraction in the hybrid network. Additionally, the matrix multiplications associated with the entire hybrid network are strassenified to reduce multiplications and the overall memory footprint to enable a more compact network.

Architecture. Figure 1 shows the hybrid neural-tree architecture optimized for KWS application with the corresponding parameters. The raw time-domain speech signal is converted to 2-D MFCC (Mel-frequency cepstral coefficients) inputs for succinct representation and efficient training. Speech features are first extracted from the MFCC inputs by one standard convolutional layer followed by two DS convolutional layers which greatly reduce dimensionality of the original speech signal. The low-dimensional compressed speech features are then fed to a single depth 2 Bonsai tree with 3 internal and 4 leaf nodes to provide global interaction and to identify the appropriate keyword in the detected voice command.

Note that the branching functions of the tree's internal nodes output a probability to influence whether the low-dimensional speech sample should be branched to a node's left or right child. During inference, non-linear prediction scores are computed for all tree nodes regardless of the most probable tree path traversed by the compressed sample. The tree nodes from the least probable paths contribute insignificantly to the overall prediction score. Computing prediction scores for all nodes certainly increases prediction costs. As the hybrid network for KWS has a shallow depth 2 tree, the incremental costs from computing prediction scores for all nodes is marginal in comparison to the computational

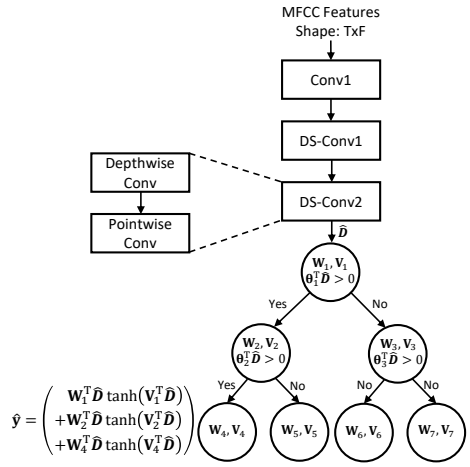

Figure 1. Hybrid neural-tree architecture.

costs of the overall hybrid network. However, evaluating the entire tree ensures that the tree computation does not incur any control-flow overhead in the processor from unpredictable branching in internal nodes. This, in turn, results in a more resource-efficient, data-parallel computation pattern and a more efficient utilization of any available SIMD units. For example, in Figure 1, a low-dimensional speech feature sample $\hat{D}$ finds the leftmost tree path most *probable* to traverse and as a result the three tree nodes along the leftmost path in the tree contribute the most to the overall prediction score, even though the prediction scores for all tree nodes are computed during inference.

End-to-end training. The three convolutional layers along with all the tree nodes of the hybrid network are trained jointly so as to maximize accuracy. A gradient-descent (GD) based training algorithm is used to train the hybrid network end-to-end. Note that the path traversed by a training point in a standard decision tree is a sharply discontinuous function of parameters of internal branching nodes thereby making gradient based techniques ineffective. In order to make effective use of GD algorithm with a differentiable loss function, the training of a Bonsai tree begins with smooth activation functions of internal branching nodes to allow points to traverse multiple paths in the tree. As training progresses, the activation functions of internal branching nodes are tuned to ensure that points gradually start traversing at most a single path.

Strassenified hybrid network. Finally, in order to reduce the hybrid network's memory footprint, we apply strassenified matrix multiplications to its convolution layers and tree section to create the strassenified hybrid network. Note that each of the tree nodes of the hybrid network learns two matrices W and V to compute a non-linear prediction score. Computation of this prediction scores at the tree nodes involves matrix multiplications, which are strassenified as

well. The hidden layer width (r) of the strassenified hybrid network is set to $0.75c_{out}$ for convolution layers(c_{out} is the number of output channels of a convolution layer), whereas r for the tree nodes is set to the number of targets (L) of the multi-class KWS classification problem.

Training a network with strassenified matrix multiplications essentially involves learning ternary W_a, W_b, and W_c matrices for each strassenified network layer. We employ the training procedure described in the StrassenNets (Tschannen et al., 2018) work to train the strassenified hybrid network. Training of the strassenified hybrid network begins with full precision W_a, W_b, and W_c. Once the network is sufficiently trained with full-precision W_a, W_b, and W_c, the elements of these three strassen matrices are quantized to ensure ternary-valued weights in them and the training continues. Quantization converts the full-precision W_b to a ternary-valued W_b^t along with a scaling factor (W_b = scaling factor * W_b^t). W_a and W_c are quantized the same way. Once the training recovers the accuracy loss with quantized strassen matrices, the three strassen matrices are fixed and the scaling factors associated with them are absorbed by full-precision $vec(A)$ portion of strassenified matrix multiplication. Note that in the context of strassenified matrix multiplications of a network layer, A is associated with the weights or filters of the layer and B is associated with the corresponding activations or feature maps. As a result, after training, W_a and $vec(A)$ can be collapsed into a vector $\hat{a} = W_a vec(A)$, as they are both fixed during inference. We follow the weight quantization procedure described in the StrassenNets (Tschannen et al., 2018) work for quantizing all strassen matrices of the hybrid network.

Furthermore, in order to recover any accuracy loss of the hybrid network compressed with strassenified matrix computations, knowledge distillation (KD) is exploited during training, as described in (Tschannen et al., 2018)[3]. Using KD, an uncompressed teacher network can transfer its prediction ability to a compressed student network by navigating its training. We use the uncompressed hybrid network as the teacher network and the compressed strassenified network as the student network in this work.

In short, the strassenified hybrid neural-tree architecture essentially combines the strengths of DS convolutions, Bonsai trees, and strassenified matrix computations, with additional strategies applied during training to improve the overall performance, while keeping a small-footprint size.

[3]We apply KD while training the strassenified DS-CNN networks in Section 2 as well. The results reported for strassenified networks in Table 1 are obtained using KD.

4 Experiments and Results

Datasets. We evaluate the hybrid neural-tree architecture on the Google speech commands dataset (Warden, 2018) and compare it against the state-of-the-art DS-CNN (Zhang et al., 2017), BinaryCmd (Fernndez-Marqus et al., 2018) and other baseline network architectures for KWS from literature (Arik et al., 2017; Sun et al., 2016; Sainath & Parada, 2015; Chen et al., 2014) analysed in (Zhang et al., 2017). The entire dataset consists of 65K different samples of 1-second long audio clips of 30 keywords, collected from thousands of different people. The length of each audio clip is 1 second, which is sufficiently long to capture one keyword. The corresponding 40 MFCC features are obtained from a speech frame of length 40ms with a stride of 20ms, yielding an input dimensionality of 49×10 features for 1 second of audio. The different network architectures are trained to classify the incoming audio into one of the 10 keywords - "Yes" "No" "Up" "Down" "Left" "Right" "On" "Off" "Stop" "Go" along with "silence" (i.e. no word spoken) and "unknown" word, which is the remaining 20 keywords from the dataset. The dataset is split into 80% for training, 10% for validation and 10% for testing. Training samples are augmented by applying background noise and random timing jitter to provide robustness against noise and alignment errors. We follow the input data processing procedure described in (Zhang et al., 2017) for training the baseline and hybrid networks presented here.

Hybrid network training. We use the Adam optimization algorithm to train the networks in the Tensorflow framework (Abadi et al., 2016). We use multi-class hinge loss to train the hybrid network[4]. The Adam optimizer with hinge loss achieves marginally better accuracy for the hybrid network than with cross-entropy loss. The network architectures are trained on the full training set and evaluated based on the classification accuracy on the test set. With a batch size of 20, the hybrid network is trained for 135 epochs with initial learning rate of 0.001 and progressively smaller learning rates after every 45 epochs. The training time for the hybrid network is restricted to 135 epochs to match against the epochs required in training the baseline DS-CNN network.

Hybrid network evaluation. The resulting testing accuracy along with the model size and the number of multiplications, and additions in the matrix-multiplication operations of the hybrid network is shown in Table 3 and compared against prior works. The hybrid network achieves an accuracy of 94.54% when compared to DS-CNN's accuracy of 94.4%, while reducing the number of operations by 44.4%. The reduction in operations of the hybrid network stems

[4]We use hinge loss to train the baseline Bonsai trees in Section 2.2.1, whereas we use standard cross-entropy loss to train the strassenified DS-CNN networks in Section 2.1.1.

Table 3. Comparison of hybrid neural-tree network (HybridNet) against DS-CNN, the current state-of-the-art for KWS application, and other baselines presented in (Zhang et al., 2017).

NETWORK	ACC. (%)	MACS	OPS	MODEL SIZE
DS-CNN	94.4	2.7M	2.7M	22.07KB
CRNN	94.0	1.5M	1.5M	73.7KB
GRU	93.5	1.9M	1.9M	76.3KB
LSTM	92.9	1.95M	1.95M	76.8KB
BASIC LSTM	92.0	2.95M	2.95M	60.9KB
CNN	91.6	2.5M	2.5M	67.6KB
DNN	84.6	0.08M	0.08M	77.8KB
HYBRIDNET	94.54	1.5M	1.5M	94.25KB

from its compute-efficient tree portion. Note that all the baseline networks in Table 3 require 1 byte to store their weights as opposed to the 4 bytes required to store the weights of the uncompressed hybrid network. Consequently, the uncompressed hybrid network requires a larger model size of 94.25KB when compared to other baselines in Table 3. Clearly, straightforward quantization of the weights of the hybrid network to low-precision values will result in reducing its model size. However, as discussed in this work, we instead apply StrassenNets over the entire hybrid network to reduce its full-precision (4 bytes) parameters and resultant model size.

Strassenified hybrid network training. We begin by training the strassenified hybrid network (ST-HybridNet) with full-precision strassen matrices (W_a, W_b, and W_c) for 135 epochs. The learning rate is initially chosen as 0.001, and later gradually reduced after every 45 epochs. We then activate quantization for these strassen matrices and the training continues. Finally, we fix the strassen matrices to their learned ternary values and continue training for another 135 epochs to ensure that the scaling factors associated with these matrices can be absorbed by full-precision $vec(A)$ portion of strassenified matrix multiplication.

Strassenified hybrid network evaluation. The testing accuracy of the ST-HybridNet is shown in Table 4, along with the reduction in the number of operations, and the model size in comparison to the uncompressed hybrid network. The ST-HybridNet achieves similar accuracy to that of the uncompressed hybrid network and the baseline DS-CNN while reducing the number of multiplications and additions by 98.89% and 12.22%, respectively, over the baseline DS-CNN network. Of particular note is that it reduces the number of additions to about 2.37M when compared to 4.09M additions of strassenified DS-CNN network described in Section 2. This, in turn, results in fewer overall operations, 2.4M, for the ST-HybridNet when compared to 2.7M operations of the baseline DS-CNN and 4.15M operations of the strassenified DS-CNN. This reduction in operations is primarily attributed to strassenifying a few (three) convolutional layers and a compute-efficient tree as opposed to strassenifying all of the five convolutional layers

Table 4. Comparison of the strassenified hybrid neural-tree network (ST-HybridNet) against the uncompressed hybrid network, DS-CNN, and strassenified DS-CNN network (ST-DS-CNN) presented in Section 2.

NETWORK	ACC. (%)	MULS, ADDS	MACS	OPS	MODEL SIZE
DS-CNN	94.4	-	2.7M	2.7M	22.07KB
ST-DS-CNN ($r = 0.75c_{out}$)	94.09	0.06M, 4.09M	-	4.15M	19.26KB
HYBRIDNET	94.54	-	1.5M	1.5M	94.25KB
ST-HYBRIDNET (WITHOUT KD)	94.51	0.03M, 2.37M	-	2.4M	14.99KB
ST-HYBRIDNET (WITH KD)	94.41	0.03M, 2.37M	-	2.4M	14.99KB

found in the baseline DS-CNN model. Owing to the ternary weights matrices, the ST-HybridNet reduces the model size to 14.99KB when compared to 22.07KB of the baseline DS-CNN network thus enabling a 32.1% savings in model size for KWS[5]. Furthermore, our ST-HybridNet does not incur any accuracy loss over the baseline DS-CNN while achieving reduction in computational costs and model size. The use of KD in training the ST-HybridNet does not result in any tangible change in accuracy.

We perform exhaustive search of feature extraction hyperparameters and model hyperparameters to develop ST-HybridNet. Table 5 summarizes the hyperparameters of different configurations of ST-HybridNet along with their impact on accuracy and computational complexity. A ST-HybridNet with two convolutional layers (one standard convolutional layer followed by one DS convolutional layer) and a single depth 2 Bonsai tree with 7 nodes (3 internal and 4 leaf nodes) reduces computational requirements of a KWS model but at the cost of more than 3% accuracy loss in comparison to the baseline DS-CNN network. Even a ST-HybridNet with three convolutional layers and a depth 1 Bonsai tree with 3 nodes (1 internal and 2 leaf nodes) cannot preserve the baseline accuracy. This hyperparameter search subsequently results in designing the ST-HybridNet with three convolutional layers and a depth 2 Bonsai tree for KWS application in this work.

As ternary W_a and full-precision $vec(A)$ weights of the ST-HybridNet are both fixed during inference, they are learned jointly as collapsed full-precision $\hat{a}$ ($W_a vec(A)$) from scratch, and these full-precision $\hat{a}$ weights along with the bias parameters occupy 7.34KB out of 14.99KB of the ST-HybridNet. The limited memory of microcontroller systems motivates further reducing the numerical precision of the entire network model, including the inputs, outputs, activations, and remaining full-precision weights, to minimize

[5]During inference, the batch normalization parameters (beta, moving mean, and moving variance) are folded either into the full-precision bias parameters of the preceding convolution layers and/or into the full-precision $vec(A)$ parameters of the ST-HybridNet.

Table 5. Different network hyperparameters of ST-HybridNet and their impact on accuracy and number of operations. D = depth of tree, N = number of tree nodes.

NETWORK	MODEL HYPERPARAMETERS	ACC. (%)	OPS
ST-HYBRIDNET	2 CONVOLUTIONAL LAYERS, D=2, N=7	91.1	1.53M
ST-HYBRIDNET	3 CONVOLUTIONAL LAYERS, D=1, N=3	93.15	2.39M
ST-HYBRIDNET	3 CONVOLUTIONAL LAYERS, D=2, N=7	94.51	2.4M

Table 6. Model quality and memory footprint after quantizing weights and activations of pre-trained ST-HybridNet. Memory footprint denotes the total memory required for storing weights and activations of a network during inference. 1KB = 1024 bytes.

NETWORK	ACC. (%)	OPS	MODEL SIZE	TOTAL MEMORY FOOTPRINT
DS-CNN	94.4	2.7M	22.07KB	37.7KB
ST-HYBRIDNET QUANTIZED (FULLY 8B ACTIVATIONS)	94.13	2.4M	10.54KB	26.17KB
ST-HYBRIDNET QUANTIZED (MIXED 8B/16B ACTIVATIONS)	94.71	2.4M	10.54KB	41.8KB

the overall memory footprint during inference.

Quantization of activations and remaining full-precision weights of strassenified hybrid network. Quantization can convert these high-precision floating-point weights and activations of ST-HybridNet to a low-precision fixed-point format more amenable for deployment in resource-constrained microcontrollers. This can also ensure faster inference through the use of fixed-point integer operations rather than floating-point operations.

We follow the quantization procedure described in (Qiu et al., 2016; Zhang et al., 2017) for quantizing the remaining full-precision weights and activations of the pre-trained ST-HybridNet. The full-precision weights and activations of the pre-trained ST-HybridNet are quantized progressively, one layer at a time, by finding the optimal min/max range for each layer that minimizes the loss in accuracy because of quantization. Table 6 captures the accuracy, model size and total memory required for storing the weights and activations of the quantized ST-HybridNet model during inference. We assume that the memory for activations is reused across different layers and, hence, the memory requirement for the activations uses the maximum of two consecutive layers (output activations from a preceding layer and input activations to the following layer). As shown in Table 6, quantizing activations of our ST-HybridNet to 8 bits reduces the model size to 10.54KB and the total memory footprint to 26.17KB, albeit with a very small loss in accuracy of 0.27%. This is primarily attributed to the intermediate activations (activations produced post-convolution with strassen matrix W_b) of the strassenified depthwise convolutions of two DS layers that require 16 bits to represent their range precisely and preserve baseline accuracy. Quantizing the $\hat{a}$ weights and the intermediate activations of the depthwise convolution layers to 16 bits and the remaining full-precision weights and activations to 8 bits in our ST-HybridNet not only recovers the small accuracy loss of the quantized ST-HybridNet with fully 8 bits activations, but also achieves marginally better accuracy than the baseline quantized DS-CNN network, possibly owing to better regularization because of quantization. The quantized ST-HybridNet with mixed $8/16$ bits activations (16 bits for strassenified depthwise convolution layers) reduces model size to 10.54KB and requires an overall memory footprint of 41.8KB. Out of 41.8KB of total footprint, 31.25KB of memory is primarily attributed to the storage of 16 bits intermediate activations of strassenified depthwise layers. Remaining layers of ST-HybridNet require at most 15.63KB of memory during inference for storing activations of two consecutive layers.

In summary, the quantized ST-HybridNet reduces model size by 52.2% and overall memory footprint by 30.6% while incurring a negligible loss in accuracy when compared to the baseline quantized DS-CNN. It is important to note that Table 6 captures accuracy results with quantizing weights and activations of the *pre-trained* ST-HybridNet. In other words, the ST-HybridNet here is not retrained post quantization. We believe this 0.27% drop in accuracy with quantizing activations to 8 bits can be recovered via integrating the quantization process into the training procedure of ST-HybridNet. We leave this exploration for future work.

5 COMPARATIVE ANALYSIS

In recent years, numerous research efforts have been devoted to compressing neural networks for deployment in resource-constrained environments through the use of model pruning, quantization, low-rank matrix factorization, compact network architecture design, etc. ST-HybridNet falls into the category of compact architecture design for IoT applications. In order to demonstrate the efficacy of ST-HybridNet over other model compression techniques, we apply state-of-the-art pruning and quantization techniques to the baseline DS-CNN network and present their performance in this section.

Model pruning. Pruning away unimportant connections induces sparsity in a neural network, thereby reducing the number of nonzero-valued parameters in the model. Recent works (Han et al., 2015; Narang et al., 2017; Zhu & Gupta, 2017) on model pruning have shown that common networks have significant redundancy and can be pruned dramatically during training with marginal to no degradation in the model accuracy. By reducing nonzero parameters of a network, model pruning attempts to reap improvements

Table 7. Model size and accuracy tradeoff for DS-CNN, the current state-of-the-art for KWS application.

SPARSITY	NONZERO PARAMETERS	ACC. (%)
0%	23.18K	94.4
50%	11.59K	94.03
75%	5.79K	92.37
90%	2.31K	87.41

in inference time and energy-efficiency. In addition to the storage for the nonzero model elements, a pruned model requires to store auxiliary data structures for indexing these elements resulting in additional storage overhead. On top of that the specialized routines involved with sparse matrix computations of a pruned model require considerable sparsity in associated matrices to realize any benefit in runtime owing to their irregular computation pattern and under utilization of any available SIMD units. Typically, a sparsity level of 70% or above is required in order for a sparse matrix computation to observe any benefit in runtime than the corresponding dense matrix computation.

We follow the gradual pruning technique proposed in (Zhu & Gupta, 2017) to prune the parameters of the baseline DS-CNN network. (Zhu & Gupta, 2017) gradually prunes the small magnitude weights to achieve a preset level of network sparsity. Table 7 compares the performance of sparse DS-CNN models pruned to varying extents. As shown in Table 7, although a 50% sparse DS-CNN model causes marginal loss in accuracy, it will be hard for the DS-CNN model to realize any benefit with this sparsity either in runtime, due to the sparse matrix computation, or in model size, due to the overhead from storing indices when compared to ST-HybridNet. Nevertheless, as different model pruning techniques (Guo et al., 2016; Aghasi et al., 2017; Wen et al., 2016; He et al., 2017; Luo et al., 2017; Yang et al., 2018; Gordon et al., 2018) are orthogonal to our compression scheme, they can be used in conjunction with ST-HybridNet to further reduce model size.

Model quantization. As mentioned previously, the baseline DS-CNN network in Table 3 uses an 8-bit fixed-point quantized format to represent weights. In order to observe the impact of binary/ternary quantization (Courbariaux et al., 2015; Rastegari et al., 2016; Lin et al., 2017; Cai et al., 2017; Li & Liu, 2016; Zhu et al., 2016), we apply ternary weight quantization (Li & Liu, 2016) over the baseline DS-CNN network. Ternary quantization of the weights of DS-CNN reduces the model size to 9.92KB but drops prediction accuracy significantly (by 2.27%). Any increase in the size of the DS-CNN network to recover the accuracy loss while using ternary quantization will lead to an increase in the number of MAC operations. Recent work on BinaryCmd (Fernndez-Marqus et al., 2018) achieves significant reduction in KWS model size but at the cost of 3.4% accuracy loss compared to the baseline DS-CNN network.

Low-rank matrix factorization. Besides pruning and quantization, low-rank matrix factorization techniques (Jaderberg et al., 2014; Tai et al., 2015; Wen et al., 2017) exploit parameter redundancy to obtain low-rank approximations of weight matrices without compromising model accuracy. Strassen matrices of our ST-HybridNet can adopt these prior proposals to further reduce model size and computational complexity.

Compact network architectures. Much research has been done in recent years on developing compact architectures (Chen et al., 2014; Sainath & Parada, 2015; Sun et al., 2016; Arik et al., 2017; Zhang et al., 2017; Li et al., 2017; Fernndez-Marqus et al., 2018; Myer & Tomar, 2018; Coucke et al., 2018) for keyword spotting on resource-constrained environments. Recent work on EdgeSpeechNets (Lin et al., 2018) produces good results albeit with significantly higher computational complexity. It is targeted for mobile processors (Arm Cortex-A53) as it requires at least $10x$ more MAC operations than our baselines and proposed ST-HybridNet, all of which are are primarily targeted for microcontrollers. EdgeSpeechNet is well beyond the constrained compute and storage budget of typical microcontrollers described in (Zhang et al., 2017).

6 CONCLUSION AND FUTURE WORK

We have presented a hybrid network architecture for a keyword spotting application capable of giving start-of-the-art accuracy levels while requiring a fraction of the model parameters and considerably fewer operations per inference pass. The hybrid architecture makes this possible by leveraging a few neural DS layers to extract features from the audio input and feeding those features to a shallow Bonsai decision tree to perform the classification. Furthermore, StrassenNets is used to significantly reduce the model size. The reduction in computation from the Bonsai tree, the parameter-efficiency of the DS convolutional layers, and the model footprint reduction provided by StrassenNets all combine to make the KWS model much more amenable to run on a highly constrained IoT device.

In the next iterations of this work, we will explore different algorithmic ways to constrain the number of additions in a strassenified network dominated with DS layers or specifically pointwise convolutions (e.g. MobileNets architecture) and develop architectures or specialized hardware suitable for such changes. This will not only enable a more homogeneous network architecture, but also will pave the way for incorporating StrassenNets into the next generation microcontrollers while maintaining acceptable computational costs and model size. We leave this exploration for future work.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

Aghasi, A., Abdi, A., Nguyen, N., and Romberg, J. Netrim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 3180–3189, 2017.

Alemdar, H., Caldwell, N., Leroy, V., Prost-Boucle, A., and Pétrot, F. Ternary neural networks for resource-efficient AI applications. *CoRR*, abs/1609.00222, 2016.

Arik, S. Ö., Kliegl, M., Child, R., Hestness, J., Gibiansky, A., Fougner, C., Prenger, R., and Coates, A. Convolutional recurrent neural networks for small-footprint keyword spotting. In *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, pp. 1606–1610, 2017.

Cai, Z., He, X., Sun, J., and Vasconcelos, N. Deep learning with low precision by half-wave gaussian quantization. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5406–5414, 2017. doi: 10.1109/CVPR.2017.574.

Chen, G., Parada, C., and Heigold, G. Small-footprint keyword spotting using deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*, pp. 4087–4091, 2014. doi: 10.1109/ICASSP.2014.6854370.

Coucke, A., Chlieh, M., Gisselbrecht, T., Leroy, D., Poumeyrol, M., and Lavril, T. Efficient keyword spotting using dilated convolutions and gating. *CoRR*, abs/1811.07684, 2018.

Courbariaux, M., Bengio, Y., and David, J. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 3123–3131, 2015.

Fernndez-Marqus, J., W.-S. Tseng, V., Bhattacharya, S., and D. Lane, N. Binarycmd: Keyword spotting with deterministic binary basis. In *1st Conference on Systems and Machine Learning*, 2018.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T., and Choi, E. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 1586–1595, 2018. doi: 10.1109/CVPR.2018.00171.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 1387–1395, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 1398–1406, 2017. doi: 10.1109/ICCV.2017.155.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*, 2014.

Kumar, A., Goyal, S., and Varma, M. Resource-efficient machine learning in 2 KB RAM for the internet of things. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1935–1944, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Li, B., Sainath, T. N., Narayanan, A., Caroselli, J., Bacchiani, M., Misra, A., Shafran, I., Sak, H., Pundak, G., Chin, K. K., Sim, K. C., Weiss, R. J., Wilson, K. W., Variani, E., Kim, C., Siohan, O., Weintraub, M., McDermott, E., Rose, R., and Shannon, M. Acoustic modeling for google home. In *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, pp. 399–403, 2017.

Li, F. and Liu, B. Ternary weight networks. *CoRR*, abs/1605.04711, 2016.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 344–352, 2017.

Lin, Z. Q., Chung, A. G., and Wong, A. Edgespeechnets: Highly efficient deep neural networks for speech recognition on the edge. *CoRR*, abs/1810.08559, 2018.

Luo, J., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 5068–5076, 2017. doi: 10.1109/ICCV.2017.541.

Myer, S. and Tomar, V. S. Efficient keyword spotting using time delay neural networks. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018.*, pp. 1264–1268, 2018. doi: 10.21437/Interspeech.2018-1979.

Narang, S., Diamos, G. F., Sengupta, S., and Elsen, E. Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119, 2017.

Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S., Wang, Y., and Yang, H. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pp. 26–35, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3856-1. doi: 10.1145/2847263.2847265.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pp. 525–542, 2016. doi: 10.1007/978-3-319-46493-0_32.

Sainath, T. N. and Parada, C. Convolutional neural networks for small-footprint keyword spotting. In *INTERSPEECH*, pp. 1478–1482. ISCA, 2015.

Sun, M., Raju, A., Tucker, G., Panchapagesan, S., Fu, G., Mandal, A., Matsoukas, S., Strom, N., and Vitaladevuni, S. Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting. In *SLT*, pp. 474–480. IEEE, 2016.

Tai, C., Xiao, T., Wang, X., and E, W. Convolutional neural networks with low-rank regularization. *CoRR*, abs/1511.06067, 2015.

Tschannen, M., Khanna, A., and Anandkumar, A. StrassenNets: Deep learning with a multiplication budget. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4985–4994, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR*, abs/1804.03209, 2018.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 2082–2090, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9.

Wen, W., Xu, C., Wu, C., Wang, Y., Chen, Y., and Li, H. Coordinating filters for faster deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 658–666, 2017. doi: 10.1109/ICCV.2017.78.

Yang, T., Howard, A. G., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X*, pp. 289–304, 2018. doi: 10.1007/978-3-030-01249-6_18.

Zhang, Y., Suda, N., Lai, L., and Chandra, V. Hello edge: Keyword spotting on microcontrollers. *CoRR*, abs/1711.07128, 2017.

Zhu, C., Han, S., Mao, H., and Dally, W. J. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *CoRR*, abs/1710.01878, 2017.