Ternary MobileNets via Per-Layer Hybrid Filter Banks

Dibakar Gope, Jesse Beu, Urmish Thakker, and Matthew Mattina
Arm ML Research Lab
{dibakar.gope, jesse.beu, urmish.thakker, matthew.mattina}@arm.com

Abstract

MobileNets family of computer vision neural networks have fueled tremendous progress in the design and organization of resource-efficient architectures in recent years. New applications with stringent real-time requirements on highly constrained devices require further compression of MobileNets-like compute-efficient networks. Model quantization is a widely used technique to compress and accelerate neural network inference and prior works have quantized MobileNets to $4-6$ *bits, albeit with a modest to significant drop in accuracy. While quantization to sub-byte values (i.e. precision* ≤ 8 *bits) has been valuable, even further quantization of MobileNets to binary or ternary values is necessary to realize significant energy savings and possibly runtime speedups on specialized hardware, such as ASICs and FPGAs. Under the key observation that convolutional filters at each layer of a deep neural network may respond differently to ternary quantization, we propose a novel quantization method that generates per-layer hybrid filter banks consisting of full-precision and ternary weight filters for MobileNets. Using this proposed quantization method, we quantize a substantial portion of weight filters of MobileNets to ternary values resulting in a* 27.98% *savings in energy, and a* 51.07% *reduction in the model size, while achieving comparable accuracy and no degradation in throughput on specialized hardware in comparison to the baseline full-precision MobileNets. Finally, we demonstrate the generalizability and effectiveness of hybrid filter banks to other neural network architectures.*

1. Introduction

Deeper and wider convolutional neural networks (CNNs) has led to outstanding predictive performance in many machine learning tasks, such as image classification, object detection, and semantic segmentation. However, the large model size and corresponding computational inefficiency of these networks often make it infeasible to run many real-time machine learning applications on resource-constrained mobile and embedded hardware, such as smartphones, AR/VR devices, etc. To enable this computation and size compression of CNN models, one particularly effective approach has been the use of resource-efficient MobileNets architecture. MobileNets introduces depthwise-separable (DS) convolution as an efficient alternative to the standard 3-D convolution operation.While MobileNets architecture has been transformative, even further compression of MobileNets is valuable in order to meet the stringent real-time requirements of new applications on highly constrained devices or to make a wider range of applications available on them [12].

Model quantization has been a popular technique to facilitate that. Quantizing the weights of MobileNets to binary $(-1,1)$ or ternary $(-1,0,1)$ values in particular has the potential to achieve significant improvement in energy savings and possibly overall throughput especially on custom hardware, such as ASICs and FPGAs while reducing the resultant model size considerably. This is attributed to the replacement of multiplications by additions in binary- and ternary-weight networks. Multipliers occupy considerably more area on chip than adders [25], and consume significantly more energy than addition operations [19, 3]. Specialized hardware can therefore trade off multiplications against additions and potentially accommodate considerably more adders than multipliers to achieve high throughput and significant savings in energy for binary- and ternary-weight networks.

However, prior approaches to binary and ternary quantization [32, 2, 25, 39] incur significant drop in prediction accuracy for MobileNets. Recent work on StrassenNets [39] shows the potential to approximate matrix multiplication (and, in turn, convolutions) of a network using mostly ternary weights and a few full-precision weights without dropping its predictive performance. For every DNN layer, StrassenNets essentially casts the (matrix) multiplication of weight matrix with activations as a 2-layer sum-product network (SPN). The number of hidden units in the SPNs determines the addition and multiplication budget of the corresponding DNN layers and in turn decides the approximation error of the corresponding matrix multiplication operations. While the results in [39] using StrassenNets

show no loss in predictive performance for a few networks in comparison to their full-precision models, the effectiveness of StrassenNets varies considerably, however, depending on the architecture of a neural network. Our observations are, for example, that while *strassenifying* DS convolutional layers reduces the model size and the number of multiplication operations significantly, this might come at the cost of a prohibitive increase in the number of addition operations. This in turn may degrade the throughput and energy efficiency of neural network inference using StrassenNets.

The exorbitant increase in additions primarily stems from the use of wide hidden layers for closely approximating each convolutional filter in a network layer. While this might be required for some of the convolutional filters in a layer, our observations indicate that not all filters require wide strassenified hidden layers. As different filters in a network layer tend to capture different features, some being more complicated than others, they respond differently to ternary quantization, and, in turn, to strassenified convolution at varied hidden layer widths. Furthermore, due to the hiddent unit reuse in the strassenified network, a group of filters with sub-filter similarities at a layer may respond more favorably to ternary quantization than outlier filters within the same layer extracting significantly different features.

Guided by these insights, we propose a layer-wise hybrid filter banks for the MobileNets architecture capable of giving start-of-the-art accuracy while requiring a fraction of the model size and considerably fewer MAC and multiplication operations per inference. The end-to-end learning of hybrid filter banks makes this possible by keeping precision critical convolutional filters in full-precision values and only strassenifying quantization tolerant filters to ternary values. *The filters that are most sensitive to quantization errors perform traditional convolutions with input feature maps, whereas ternary quantization tolerant filters can perform strassenified convolutions using narrow hidden layers.* We apply this proposed quantization scheme to the MobileNets-V1 architecture. The hybrid filter banks for MobileNets achieves a 46.4%, and a 51.07% reduction in multiplications and model size respectively while incurring modest increase in additions. This translates into a 27.98% savings in energy required per inference while ensuring no degradation in throughput on a DNN hardware accelerator consisting of both MAC and adders when compared to the execution of baseline MobileNets on a MAC-only hardware accelerator. The hybrid filter banks accomplishes this with a very minimal loss in accuracy of 0.51%. Hybrid filter banks applied to ResNet yields consistently better accuracy results than StrassenNets on CIFAR-10 dataset, demonstrating its generalizability to other neural network architectures. **To the best of our knowledge, the hybrid filter banks proposed in this work is a first step towards quantizing the already compute-efficient MobileNets architecture to ternary values on a large-scale dataset, such as ImageNet.**

The remainder of the paper is organized as follows. Section 2 elaborates on the incentives behind the development of per-layer hybrid filter banks. Section 3 describes our hybrid filter banks. Section 4 presents results. Section 5 compares hybrid filter banks against prior works. Section 6 concludes the paper.

2. Model Quantization Limitations for MobileNets

This section briefly reviews the important existing works on ternary quantization, which we focus on in this paper, and illustrates their limitations to motivate the development of per-layer hybrid filter banks.

2.1. Ternary Quantization of Weights

In order to observe the impact of ternary quantization [10, 32, 26, 5, 25, 49, 48], we apply the ternary weight quantization method from [25] over the baseline MobileNets-V1 architecture. It approximates a full-precision weight W^{fp} by a ternary-valued W^t and a scaling factor such that $W^{fp} \approx$ scaling factor $* W^t$. Ternary quantization of the weights of MobileNets achieves substantial reduction in model size but at the cost of significant drop (by 9.66%, see Table 1) in predictive performance when compared to the full-precision model. Any increase in the size of the MobileNets architecture to recover the accuracy loss while using ternary quantization will lead to a significant increase in the number of addition operations. Recent work on StrassenNets [39], which we describe next, has shown the potential to achieve near state-of-the-art accuracy for a number of deep CNNs while maintaining acceptable increase in addition operations.

2.2. StrassenNets

Given two 2×2 square matrices, Strassen's matrix multiplication algorithm requires 7 multiplications to compute the product matrix instead of the 8 required with a naïve implementation of matrix multiplication. It essentially casts the matrix multiplication as a 2-layer SPN computation.

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

W_a, $W_b \in K^{r \times n^2}$ and $W_c \in K^{n^2 \times r}$ represent ternary matrices with $K \in \{-1, 0, 1\}$. The $W_a vec(A)$ and $W_b vec(B)$ of the SPN combine the elements of A and B through additions, and/or subtractions by using the two associated ternary matrices W_a and W_b respectively to generate r intermediate terms each. The two generated r-length

intermediate terms are then elementwise multiplied to compute the r-length $(W_b vec(B)) \odot (W_a vec(A))$ vector. The outmost ternary matrix W_c later combines these intermediate r terms through additions, and/or subtractions to produce $vec(C)$. Hence, the number of multiplications and additions required for the Strassen's algorithm are decided by the width of the hidden layer of the SPN, r. Given two 2×2 matrices, for example, ternary matrices W_a, W_b, and W_c with sizes of 7×4, 7×4, and 4×7 respectively can multiply them using 7 multiplications instead of 8.

While Strasssen's algorithm requires a hidden layer with 7 units here to compute the exact product matrix, the StrassenNets work [39] instead realizes approximate matrix multiplications in DNN layers[1] using fewer hidden layer units. The learned ternary matrices can then use significantly fewer multiplications than Strassen's algorithm. This approximation can have impact on predictive performance of the DNN architecture. The significant compression achieved by StrassenNets for 3×3 convolutions [39] and increasing visibility of DS convolution layers in compute-efficient networks [21, 33, 47, 8] motivated us to apply StrassenNets over MobileNets architecture dominated with DS layers to reduce its computational complexity and model size even further. Among the various MobileNets architectures [21, 33, 20], in this work we extensively study the quantization of MobileNets-V1 [21]. MobileNets-V1 stacks one 3×3 and 13 DS convolutional layers. A DS convolution first convolves each channel in the input feature map with a separate 2-D filter (depthwise convolution) and then uses 1×1 pointwise convolutions to combine the outputs in the depth dimension.

2.2.1 StrassenNets for MobileNets

We observe that while strassenifying MobileNets is effective in reducing the number of multiplications and the model size significantly, it increases additions prohibitively to preserve the predictive performance of the baseline MobileNets with 16-bit floating-point weights. Table 1 captures our observation. The strassenified MobileNets with the $r = 2c_{out}$ configuration achieves a comparable accuracy to that of the full-precision MobileNets while reducing multiplications by 97.91% but increasing additions by 317.59% (149.49M MACs of MobileNets vs. 3.11M multiplications and 624.27M additions of ST-MobileNets with $r = 2c_{out}$). This in turn offers modest savings in energy required per inference but causes significant degradation in throughput (see Section 4 for details). The use of fewer hidden units (r) (e.g. $r = c_{out}$) incurs a significant accuracy loss of 3.4%.

[1] A convolutional operation in DNN layers can be reduced to a general matrix multiplication (GEMM). 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.

Table 1: Performance of MobileNets-V1 and strassenified MobileNets-V1 (ST-MobileNets) with the width multiplier of 0.5 on ImageNet dataset. r is the hidden layer width of a strassenified convolution layer, c_{out} is the number of output channels of the corresponding convolution layer. MAC = multiply-accumulate operation, E = Energy/inference (normalized), TP = Throughput (normalized).

Network	Acc. (%)	Muls, Adds (M)	MACs (M)	Model size (KB)	E	TP
MobileNets (float16)	65.2	-	149.49	2590.07	1	1
MobileNets (TWN)	55.54	-, 149.49	-	323.75	0.2	2
ST-MobileNets ($r = 0.5c_{out}$)	48.92	0.77, 158.54	8.69	522.33	0.27	1.69
ST-MobileNets ($r = 0.75c_{out}$)	56.95	1.16, 236.16	8.69	631.76	0.37	1.17
ST-MobileNets ($r = c_{out}$)	61.8	1.55, 313.78	8.69	741.19	0.48	0.9
ST-MobileNets ($r = 2c_{out}$)	65.14	3.11, 624.27	8.69	1178.92	0.9	0.46

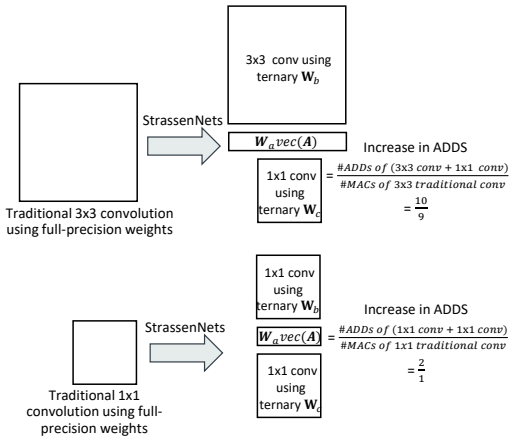


Figure 1: Application of StrassenNets to 3 x 3 and 1 x 1 convolution. The cost of elementwise multiplication with intermediate $W_a vec(A)$ is comparably negligible and hence is ignored in estimating the increase in additions.

2.2.2 Compute inefficiency of StrassenNets for MobileNets

Note that while strassenifying traditional 3×3 or 5×5 convolutional layers increases the addition operations marginally as observed in StrassenNets [39], that trend does not hold true when StrassenNets is applied over MobileNets dominated with DS layers. This is attributed to the fact that the computational cost of a neural network with DS layers is dominated by 1×1 pointwise convolutions [21]

and strassenifying a 1×1 convolution requires executing two equal-sized (for $r = c_{out}$) 1×1 convolutions with ternary weights along with few elementwise multiplications in place of the standard 1×1 convolution, as shown in Figure 1. This in turn causes a significant increase ($2 : 1$ or 100%) in additions when compared to the execution of the standard 1×1 pointwise convolution. On the other hand, as Figure 1 illustrates, a 3×3 strassenified convolution with $r = c_{out}$ instead requires executing a 3×3 convolution and a 1×1 convolution with ternary weights in conjunction with few elementwise multiplications. This in turn results in a marginal increase ($10 : 9$ or 11.1%) in additions in comparison to the execution of the standard 3×3 convolution. This overhead of addition operations with applying StrassenNets to DS convolution layers goes up in proportion to the width of the hidden layers, i.e. to the size of the ternary convolution operations, as observed in Table 1, reducing the throughput and energy-efficiency of neural network inference.

This also indicates that DS convolutions, being more efficient, are more prone to quantization error and this manifests when StrassenNets is applied. Considering the fact that MAC operations typically consume about five times more energy than addition operations for 16-bit floating-point values [19, 3], an about 317.59% increase in additions in place of about 98% saving on multiplications will result in diminishing or no returns in terms of energy savings and runtime speedups even on specialized hardware dominated with adders. The increase in computational costs of MobileNets with applying StrassenNets along with the high accuracy and stringent real-time requirements of new applications on highly constrained devices necessitate a model architecture exploration that can exploit the compute efficiency of DS layers and the model size reduction ability of StrassenNets while maintaining acceptable or no increase in additions.

2.2.3 Variance in the sensitivity of convolutional filters to ternary quantization

Although a strassenified MobileNets with $r = 2c_{out}$ recovers the accuracy loss of $r = c_{out}$, it makes a strong assumption that all filters require wider strassenified hidden layers to quantize to ternary values to preserve the representational power of the baseline full-precision network. While this might be true for some of the convolutional filters, not all filters need to be quantized using the $r = 2c_{out}$ configuration. This observation stems from the following two reasons:

(a) Different sensitivity of individual filters to StrassenNets. Different convolutional filters tend to extract different type of features, ranging from simple features (e.g. edge detection) to more complicated higher-level (e.g. facial shapes) or object specific features. As a result, different filters may respond differently to ternary quantization. That implies there are filters that are easy to quantize to ternary values using narrower hidden layers while still ensuring low L2 reconstruction error in output feature maps, and vice versa.

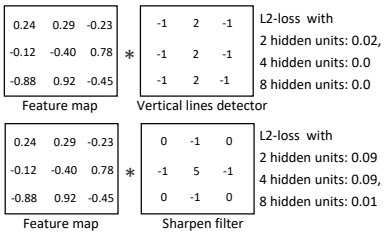
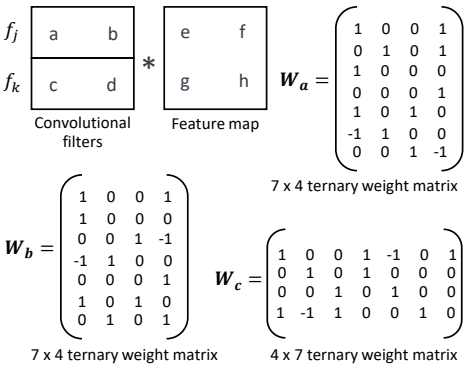


Figure 2: Variance in the sensitivity of individual convolutional filters to quantization.

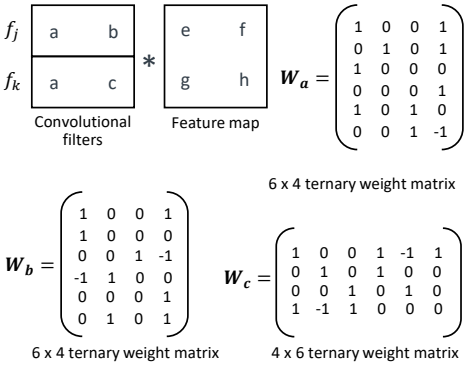
Given a feature map, Figure 2 presents a scenario where a strassenified vertical lines detector with fewer hidden layer units can closely approximate the output map (with low L2 reconstruction loss) produced otherwise using its full-precision counterpart. However a convolutional filter that sharpen images requires a wider hidden layer to ensure a low L2 loss (see Appendix A.1 for more details). Note that we only consider 2D filters for illustration purpose, whereas this difference in complexity should exist in 3D filters common to CNNs.

(b) Different sensitivity of group of filters to StrassenNets. Furthermore, there exists convolutional filters at each layer that tend to extract different features but can have numerical-structural similarities (e.g., a 3×3 vertical lines detector and a horizontal lines detector sharing common values at all the corners and at the center, see Appendix B for details). In addition to that, there exists filters that tend to extract fairly similar features with slightly different orientations (e.g. two filters attempting to detect edges rotated by few degrees). As a result, when these groups of convolutional filters are quantized to ternary values using StrassenNets, they may share many hidden layer elements. These groups of convolutional filters with similar value structure in turn are more amenable to quantization using fewer hidden layer units than filters with no common value structure. Given a constrained hidden layer budget for StrassenNets (e.g. $r = c_{out}$), these groups of convolutional filters may together respond well to ternary quantization while other dissimilar filters struggle to be strassenified alongside them with low quantization error, due to the restricted hidden layer bandwidth.

Figure 3(a) specifies a set of weight matrices that can perform exact convolution of the 2×2 filter bank comprising f_j and f_k with the feature map using 7 multiplications. Note that the two filters f_j and f_k do not have any common values. However, owing to the presence of common value



(a) Strassen's matrix multiplication for two filters f_j anf f_k having unique values.



(b) Strassen's matrix multiplication for two filters f_j anf f_k having some common values.

Figure 3: Understanding the sensitivity of group of filters to ternary quantization.

of a between f_j and f_k filters in Figure 3(b), Strassen's algorithm now can compute the exact product matrix using only 6 multiplications instead of the 7 required otherwise for unique filters lacking common value structure in Figure 3(a). A set of ternary weight matrices with fewer hidden units implementing an exact convolution in this case is shown in Figure 3(b).

Motivated by these observations, we propose per-layer hybrid filter banks.

3. Per-Layer Hybrid Filter Banks

The per-layer hybrid filter banks can quantize a substantial fraction of convolutional filters to ternary values at each layer while relying on few remaining full-precision filters to preserve the representational power of the original full-precision network. *As easy-to-quantize filters are quantized only using StrassenNets leaving the difficult-to-quantize filters in full-precision values, this should in turn require narrow hidden layers for quantizing them thus restricting the increase in additions and resulting in an overall reduction in multiplications, MAC operations and memory footprint while ensuring no loss in accuracy.*

Architecture. The proposed quantization method convolves the same input feature map with full precision weight filters and ternary weight filters in parallel, concatenating the feature maps from each convolutions into a unified feature map. This concatenated feature map is fed as input to the next network layer. At each layer, the combination of the two convolutions from full-precision and ternary filters ensures that they combine to form a output feature map of identical shape as in the baseline full-precision network. For instance, given an input feature map with c_{in} channels, the quantization technique applies traditional convolution with k full-precision weight filters W_{fp} of shape $c_{in} \times w_k \times h_k$ and strassen convolution with $c_{out} - k$ ternary weight filters W_t to produce a feature map of total c_{out} channels for a layer. Here c_{out} is the number of channels in the output volume of the corresponding convolution layer in the baseline full-precision network, and w_k, h_k are the kernel size. The fraction of channels generated in an output feature map from the full-precision weight filters, α (or in others words the channels generated from the ternary weight filters, $1 - \alpha$) is a hyperparameter in our quantization technique and it decides the representational power and computational costs of MobileNets with hybrid filter banks. Figure 4 shows the organization of the hybrid filter bank for a MobileNets layer. The depthwise convolutions of the depthwise-separable layers are not quantized using either StrassenNets or our hybrid filter banks. This is primarily due to the following reasons: (a) they do not dominate the compute bandwidth of MobileNets [21], (b) as per our observations, quantizing those to ternary values hurt the accuracy significantly without offering any significant savings in either model size or computational costs. The strassenified convolutions portion of hybrid filter banks at each layer are quantized using a number of r values, where r is the hidden layer width of a strassenified convolution layer. *The $r << 2c_{out}$ configuration in conjunction with an optimal non-zero α should offer substantial savings in model size and addition operations without compromising accuracy in comparison to a fully strassenified MobileNets architecture with $r = 2c_{out}$ configuration.* The presented quantization technique can also be applied to the fully-connected layer parameters, however, we only focus on convolution layers in this work. We compress the last fully-connected layer of MobileNets uniformly using StrassenNets.

End-to-end training. The full-precision filters along with the strassenified weight filters for each layer are trained jointly using a gradient-descent (GD) based training algorithm so as to maximize accuracy. Before the training begins, depending on the value of α, the top $\alpha * c_{out}$ channels of a feature map are configured to generate from full-precision traditional convolutions, and the remaining $1 - \alpha * c_{out}$ channels are forced to generate from ternary strassenified convolutions. Note that the order of the chan-

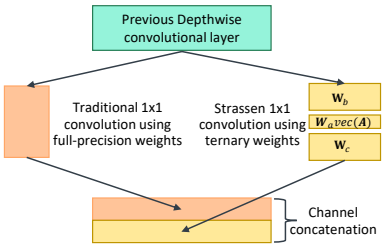


Figure 4: A MobileNets pointwise layer with hybrid filter bank.

nels generated in the output feature volume by either full-precision filters or ternary filters is not important, as the output feature map comprising all the channels generated forms the input of the subsequent layer and the weights in the subsequent layer can adjust to accommodate that. During the end-to-end training process, the organization of hybrid filter banks tend to influence the difficult-to-quantize filters (that require full-precision filters to extract features) to be trained using full-precision values, and the filters that are less susceptible to ternary quantizationto be trained using ternary values from strassenified convolutions. Furthermore, in order to recover any accuracy loss of the hybrid MobileNets compressed with strassenified matrix computations, we use knowledge distillation, as exploited in [39], during training. Knowledge distillation allows an uncompressed teacher network to transfer its prediction ability to a compressed student network by navigating its training. We use the uncompressed MobileNets with per-layer hybrid filter banks as the teacher network and the compressed network with ternary weight matrices as the student network.

4. Experiments and Results

Datasets and experimental setup. We evaluate the MobileNets-V1 architecture comprising proposed per-layer hybrid filter banks (Hybrid MobileNets) on the ImageNet (ILSVRC2012) dataset and compare it against the state-of-the-art MobileNets [21] with 16-bit floating-point weights. The baseline and other network architectures presented here use a width multiplier of 0.5[2] to stress more the impact quantization. We use MXNet [6] based GluonCV toolkit[3] to train the networks. In this work, the baseline MobileNets and the full-precision filters of the hybrid filter banks use 16-bit floating-point weights. We quantize the activations of the baseline and proposed architectures to 16-bit floating-point values. A 8-bit representation of weights and activations should not alter the conclusions made in this work. At the time of writing this paper, GluonCV toolkit does not support training with 8-bit weights and activations.

[2]Using a width multiplier of 0.5 halves the number of channels used in each layer of the original MobileNets architecture [21].

[3]GluonCV: a Deep Learning Toolkit for Computer Vision, https://gluon-cv.mxnet.io/index.html

Hybrid MobileNets architecture training. We use the Nesterov accelerated gradient (NAG) optimization algorithm and follow the other training hyperparameters described in the GluonCV framework for training the baseline full-precision MobileNets, strassenified MobileNets and our proposed Hybrid MobileNets. We begin by training the Hybrid MobileNets with full-precision strassen matrices (W_a, W_b, and W_c) for 200 epochs. With a mini-batch size per GPU of 128 on a 4 GPU system, the learning rate is initially chosen as 0.2, and later gradually reduced to zero following a cosine decay function as used in the GluonCV framework for training the baseline full-precision MobileNets (see Appendix A.2 for more details). We then activate quantization for these strassen matrices and the training continues for another 75 epochs with initial learning rate of 0.02 and progressively smaller learning rates. Quantization converts a full-precision strassen matrix to a ternary-valued matrix along with a scaling factor (e.g., W_b = scaling factor * W_b^t).

To evaluate our hypothesis that some full-precision filters are changing significantly to recover features lost due to quantization, we measured the L2 distance between their pre- and post-quantization weight vectors. We found the L2 distances fit a normal distribution: most filters experience low-to-moderate changes to their weight vectors while a few exceptional filters saw very significant movement. This supports our claim that the full-precision filters are preserving the overall representational power of the network. Finally, we fix the strassen matrices of the hybrid filter banks to their learned ternary values and continue training for another 25 epochs with initial learning rate of 0.002 and progressively smaller learning rates to ensure that the scaling factors associated with the ternary matrices can be absorbed by full-precision $vec(A)$ portion of strassenified matrix multiplication.

Energy and throughput modeling for hybrid filter banks. The proposed per-layer hybrid filter banks for MobileNets can be executed by existing DNN hardware accelerators, such as DaDianNao [7] and TPU [23] consisting of only MAC units. However, in order to achieve an energy- and runtime- efficient execution of hybrid filter banks dominated with additions, we propose a custom hardware accelerator, where a fraction of MAC units are replaced by low-cost adders within the same silicon area. A 16-bit floating-point MAC unit takes about twice the area of a 16-bit floating-point adder [30]. Given a fixed silicon area and a model configuration for Hybrid MobileNets, the ratio of MAC units to adders in the proposed hardware accelerator is decided in such a way that the maximum possible throughput can be achieved for the configuration. We use the energy consumption numbers of adder and MAC units reported in [19] to estimate the energy required per inference of baseline and proposed models.

Table 2: Top-1 accuracy along with the computational costs, model size, and energy per inference for baseline MobileNets-V1, ST-MobileNets, and Hybrid MobileNets on ImageNet dataset. α is the fraction of channels generated by the full-precision weight filters at each layer, c_{out} is the number of remaining channels generated by the ternary strassen filters at the corresponding convolutional layer, r is the hidden layer width of the strassenified convolutions. The last column shows the throughput of proposed models on an area-equivalent hardware accelerator comprising both MAC and adder units when compared to the throughput of baseline MobileNets with 16-bit floating-point weights on a MAC-only accelerator.

Network	Alpha (α)	r	Acc. (%)	Muls, Adds	MACs	Model size	Energy/inference (normalized)	Throughput (normalized)
MobileNets (float16)	-	-	65.2	-	149.49M	2590.07KB	1	1
ST-MobileNets	0	$2c_{out}$	65.14	3.11M, 624.27M	8.69M	1178.92KB	0.9	0.46
MobileNets (Hybrid filter banks)	0.25	c_{out}	63.62	1.16M, 204.63M	43.76M	1004.67KB	0.56	1.02
		$1.33c_{out}$	63.47	1.55M, 270.95M	43.76M	1097.07KB	0.65	0.83
		$2c_{out}$	64.84	2.33M, 405.59M	43.76M	1284.65KB	0.84	0.6
MobileNets (Hybrid filter banks)	0.375	c_{out}	64.13	0.97M, 157.84M	61.3M	1131.43KB	0.62	1.06
		$1.6c_{out}$	64.17	1.55M, 250.34M	61.3M	1260.44KB	0.74	0.8
		$2c_{out}$	65.2	1.94M, 312.01M	61.3M	1346.45KB	0.83	0.68
MobileNets (Hybrid filter banks)	0.5	c_{out}	**64.69**	**1.28M, 142.37M**	**78.83M**	**1267.13KB**	**0.72**	**1**
		$2c_{out}$	65.17	1.55M, 228.68M	78.83M	1327.88KB	0.83	0.77

Hybrid MobileNets architecture evaluation. One of the main focus of our evaluation is the study of how α impacts on the performance of our models. This parameter, that can be independently set for each convolutional layer in the network, is directly proportional to the number of learnable parameters in a given layer. In this work, we use identical value of α for all the layers of Hybrid MobileNets. We believe use of different values for different layers may result in better cost accuracy trade-offs. We leave this exploration for future work. Ideally small values of α and r are desired to achieve significant reduction in MAC along with addition operations while preserving the baseline accuracy.

We search the model hyperparameters space systematically to develop Hybrid MobileNets. Table 2 captures the top-1 accuracy of the Hybrid MobileNets for various configurations of α and hidden layer width r, along with their impact on computational costs, model size, energy required per inference, and throughput and and compares that against baseline full-precision MobileNets, and ST-MobileNets. As shown in Table 2, the ST-MobileNets and various configurations of Hybrid MobileNets offer comparable reduction (about 50%) in model size over the baseline full-precision Mobilenets. While the $r = 2c_{out}$ configurations for different values of α (0.25, 0.375, and 0.5) can preserve the baseline top-1 accuracy of 65.2% and offer modest savings in energy required per inference, that comes at the cost of large increase in additions. This in turn causes significant degradation in throughput on the proposed hardware accelerator when compared to the throughput of the baseline full-precision MobileNets on an existing DNN accelerator consisting of only MAC units. On the other end, the $c_{out} \leq r < 2c_{out}$ configurations with the α of 0.25 and 0.375 incur modest to significant drop in top-1 accuracy possibly owing to lack of enough full-precision weights filters at each hybrid filter bank to preserve the representational ability of the overall network. The $r < c_{out}$ configurations for different values of α leads to large drop in prediction accuracy and hence is not shown in Table 2.

The Hybrid MobileNets with the $\alpha = 0.5$ and $r = c_{out}$ configuration strikes an optimal balance between accuracy, computational costs, energy, and throughput. It achieves comparable accuracy to that of the baseline MobileNets, strassenified and Hybrid MobileNets with the $r = 2c_{out}$ configuration while reducing the number of MACs, and multiplications by 47.26%, and 46.4% respectively and requiring a modest (45.51%) increase in additions over the baseline MobileNets architecture. Of particular note is that it reduces the number of additions to about 142.37M when compared to 624.27M additions of ST-MobileNets described in Section 2. The significant reduction in MAC operations and modest increase in additions over the baseline full-precision MobileNets in turn translates into 27.98% savings in energy required per inference while ensuring no degradation in throughput in comparison to the execution of baseline MobileNets on a MAC-only hardware accelerator. This reduction in additions is primarily attributed to strassenifying easy-to-quantize filters using fewer hidden units ($r = c_{out}$) while relying on full-precision filters to generate 50% channels at each layer and preserve the representational ability of the overall MobileNets architecture. Owing to the substantial presence of ternary weights matrices, the Hybrid MobileNets with the

$\alpha = 0.5$ and $r = c_{out}$ configuration reduces the model size to 1267.13KB when compared to 2590.07KB of the baseline MobileNets network thus enabling a 51.07% savings in model size.

In summary, the Hybrid MobileNets reduces model size by 51.07% and energy required per inference by 27.98% while incurring a negligible loss in accuracy and no degradation in throughput when compared to the baseline full-precision MobileNets. Note that because of the large savings in model size, our Hybrid MobileNets will have significantly fewer accesses to the energy- and power-hungry DRAM. This in conjunction with skipping ineffectual computations of zero-valued weights in our proposed hardware accelerator (as exploited by [46]), owing to about $40 - 50\%$ of sparsity in the ternary weight matrices of strassenified layers as we observe, will improve the energy savings and run-time performance even further. Our current energy and throughput modeling does not take this into account. We leave this exploration for future work.

Generalizability of hybrid filter banks to other network architectures. We evaluate the ResNet-20 architecture comprising hybrid filter banks (Hybrid ResNet-20) on the CIFAR-10 dataset to demonstrate the efficacy of hybrid filter banks over other state-of-the-art ternary quantization techniques and its generalizability to other neural network architectures, especially to architectures dominated with 3×3 convolutional layers. ResNet-20 has 19 3×3 convolutional layers. The Hybrid ResNet-20 consistently achieves a better accuracy for different hidden layer widths in comparison to StrassenNets. For example, the Hybrid ResNet-20 with the $\alpha = 0.25$ and the $r = 0.75c_{out}$ configuration achieves an accuracy of 91.55% when compared to the accuracy of 90.62% observed by StrassenNets with $r = 0.75c_{out}$. The accuracy of full-precision ResNet-20 is 92.1%. All other configurations consistently outperform the state-of-the-art StrassenNets (see Appendix C), demonstrating the generalizability and effectiveness of hybrid filter banks to 3×3 convolutional layers.

Furthermore, we evaluate MobileNets-V2 [33] with hybrid filter banks on the ImageNet dataset. The initial performance results for MobileNets-V2 is promising, incurring very marginal loss (about 2%) in accuracy compared to the uncompressed MobilNets-V2 with only the first set of hyperparameters we chose. We believe the small accuracy drop can be bridged with more model hyperparameters exploration (e.g., appropriate division of output channels to be generated from either full-precision or ternary weight filters at each layer, appropriate value of hidden layer width for ternary weight filters, etc.) associated with hybrid filter banks approach and knowledge distillation (as exploited by StrasseNets baseline and MobileNets-V1 with hybrid filter banks and mentioned in Section 3). Knowledge distillation historically improves accuracy by another $1 - 2\%$.

5. Comparison against Prior Work

In recent years, numerous research efforts have been devoted to quantizing ResNet architecture to ternary values while preserving the accuracy of full-precision model [39, 28, 43, 11, 24, 52, 51, 34, 45, 15]. However, none of the recent works on binary and ternary quantization demonstrate their potential to quantize MobileNets on ImageNet or other datasets. There are recent works [40, 4, 9] that can quantize MobileNets with 4-6-bit weights (see Appendix D and Table 2 in [28] for more details). **To the best of our knowledge, the hybrid filter banks proposed is a first step towards quantizing the already compute-efficient MobileNets to ternary values on a large-scale dataset, such as ImageNet.** The hybrid filter banks quantizes a significant fraction of weight filters of MobileNets to ternary values while achieving comparable accuracy to that of baseline full-precision model on ImageNet. Nevertheless, the hybrid filter banks can benefit further by adopting these prior proposals. Furthermore, several approaches have been proposed in recent years on developing compressed neural networks through the use of weight pruning, tensor decomposition, compact network architecture design, etc.

Weight pruning. Recent work on channel pruning [17] demonstrates negligible drop in accuracy for MobileNets while achieving significant reduction in computational costs. As different channel pruning [17, 53, 18] and filter pruning techniques [16, 31, 50, 14, 1, 41, 29, 44, 13] are orthogonal to our compression scheme, they can be used in conjunction with hybrid filter banks to further reduce model size and computational complexity.

Tensor decomposition. Tensor decomposition techniques [22, 35, 42, 38, 36, 37] exploit parameter redundancy to obtain low-rank approximations of weight matrices. Full-precision weights filters and Strassen matrices of hybrid filter banks can adopt these prior proposals to further reduce model size and computational complexity.

Compact network architectures. While we show promising results for MobileNets-V1 and ResNet-20 here, the benefits of hybrid filter banks should scale when extended to other popular resource-efficient architectures dominated with either DS convolutions, such as ShuffleNet [47], and Xception [8] or 3×3 convolutions.

6. Conclusion and Future Work

We use per-layer hybrid filter banks to quantize already highly optimized CNNs, especially MobileNets to ternary weights. We use 16-bit to represent intermediate activations and traditional weight filters of hybrid filter banks. In future, we plan to explore the impact of quantizing them to 8-bit or less. In addition, it will be interesting to see how channel pruning [17, 53] assists in reducing the computational complexity of strassenified MobileNets.

References

[1] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: 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*, pages 3180–3189, 2017. 8

[2] Hande Alemdar, Nicholas Caldwell, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. Ternary neural networks for resource-efficient AI applications. *CoRR*, abs/1609.00222, 2016. 1

[3] R. Andri, L. Cavigelli, D. Rossi, and L. Benini. Yodann: An architecture for ultralow power binary-weight cnn acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):48–60, Jan 2018. 1, 4

[4] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *CoRR*, abs/1810.05723, 2018. 8, 14

[5] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 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*, pages 5406–5414, 2017. 2

[6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015. 6

[7] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 609–622, Washington, DC, USA, 2014. IEEE Computer Society. 6

[8] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 3, 8

[9] Yoni Choukroun, Eli Kravchik, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. *CoRR*, abs/1902.06822, 2019. 8, 14

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 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*, pages 3123–3131, 2015. 2

[11] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 8, 14

[12] Dibakar Gope, Ganesh Dasika, and Matthew Mattina. Ternary hybrid neural-tree networks for highly constrained iot applications. *CoRR*, abs/1903.01531, 2019. 1

[13] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. 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*, pages 1586–1595, 2018. 8

[14] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1387–1395, USA, 2016. Curran Associates Inc. 8

[15] Yiwen Guo, Anbang Yao, Hao Zhao, and Yurong Chen. Network sketching: Exploiting binary structure in deep cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4040–4048, 2017. 8

[16] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015. 8

[17] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: automl for model compression and acceleration on mobile devices. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, pages 815–832, 2018. 8

[18] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1398–1406, 2017. 8

[19] M. Horowitz. Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, Feb 2014. 1, 4, 6

[20] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019. 3

[21] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 3, 5, 6

[22] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*, 2014. 8

[23] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James

Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12, New York, NY, USA, 2017. ACM. 6

[24] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 8, 14

[25] Fengfu Li and Bin Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016. 1, 2

[26] Xiaofan Lin, Cong Zhao, and Wei Pan. 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*, pages 344–352, 2017. 2

[27] Zhi Gang Liu and Matthew Mattina. Learning low-precision neural networks without straight-through estimator (STE). In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 3066–3072, 2019. 14

[28] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. 8, 14

[29] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 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*, pages 5068–5076, 2017. 8

[30] David R. Lutz. Arm floating point 2019: Latency, area, power. In *IEEE Symposium on Computer Arithmetic*, 2019. 6

[31] Sharan Narang, Gregory F. Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119, 2017. 8

[32] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 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*, pages 525–542, 2016. 1, 2

[33] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. 3, 8

[34] Qigong Sun, Fanhua Shang, Kang Yang, Xiufang Li, Yan Ren, and Licheng Jiao. Multi-precision quantized neural networks via encoding decomposition of {-1, +1}. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pages 5024–5032, 2019. 8

[35] Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. *CoRR*, abs/1511.06067, 2015. 8

[36] Urmish Thakker, Jesse G. Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. Run-time efficient RNN compression for inference on edge devices. *CoRR*, abs/1906.04886, 2019. 8

[37] Urmish Thakker, Jesse G. Beu, Dibakar Gope, Chu Zhou, Igor Fedorov, Ganesh Dasika, and Matthew Mattina. Compressing rnns for iot devices by 15-38x using kronecker products. *CoRR*, abs/1906.02876, 2019. 8

[38] Urmish Thakker, Igor Fedorov, Jesse Beu, Dibakar Gope, Chu Zhou, Ganesh Dasika, and Matthew Mattina. Pushing the limits of rnn compression. *CoRR*, abs/1910.02558, 2019. 8

[39] Michael Tschannen, Aran Khanna, and Animashree Anandkumar. StrassenNets: Deep learning with a multiplication budget. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4985–4994, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 1, 2, 3, 6, 8, 13

[40] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 8, 14

[41] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2082–2090, USA, 2016. Curran Associates Inc. 8

[42] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 658–666, 2017. 8

[43] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 8, 14

[44] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 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*, pages 289–304, 2018. 8

[45] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate

and compact deep neural networks. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VIII*, pages 373–390, 2018. 8

[46] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-x: An accelerator for sparse neural networks. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-49, pages 20:1–20:12, Piscataway, NJ, USA, 2016. IEEE Press. 8

[47] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 3, 8

[48] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016. 2

[49] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016. 2

[50] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *CoRR*, abs/1710.01878, 2017. 8

[51] Shilin Zhu, Xin Dong, and Hao Su. Binary ensemble neural network: More bits per network or more networks per bit? In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 8

[52] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Structured binary neural networks for accurate image classification and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 8

[53] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jin-Hui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 883–894, 2018. 8

A. Training Details

A.1. Sensitivity of Convolutional Filters to StrassenNets

We generate a training set containing 100k pairs (A_i, B_i) with values i.i.d. uniform on $[-1, 1]$ in A_i, and values of a given convolutional filter in B_i. The SPN is then trained using different number of hidden units. We begin training with full-precision weights (initialized i.i.d. uniform on $[-1, 1]$) for one epoch with SGD (learning rate 0.1, momentum 0.9, mini-batch size 4), activate quantization, and train for few epochs with initial learning rate of 0.01 and progressively smaller learning rates. Once the training converges after activation of the quantization, we collect the L2-loss.

A.2. Hyperparameters Settings for Training Hybrid MobileNets

Table 3: Hyperparameters for training Hybrid MobileNets

.

Training phase	Hyperparameters
Train using full-precision strassen matrices	Batch size per GPU: 128 Number of GPUs used: 4 Optimizer: Nesterov accelerated gradient (NAG) (Momentum: 0.9, Weight decay: 0.0001) Number of epochs: 200 Weight initialization: Xavier Initial, final learning rate: 0.2, 0.0 Learning rate schedule: cosine decay Number of warmup epochs: 5 Starting warmup learning rate: 0.0 Size of the input image: 224 x 224 x 3
Activate quantization for strassen matrices	Batch size per GPU: 128 Number of GPUs used: 4 Optimizer: Nesterov accelerated gradient (NAG) (Momentum: 0.9, Weight decay: 0.0001) Number of epochs: 75 Initial, final learning rate: 0.02, 0.0 Learning rate schedule: cosine decay
Freeze strassen matrices to ternary values	Batch size per GPU: 128 Number of GPUs used: 4 Optimizer: Nesterov accelerated gradient (NAG) (Momentum: 0.9, Weight decay: 0.0001) Number of epochs: 25 Initial, final learning rate: 0.002, 0.0 Learning rate schedule: cosine decay

The training images from ImageNet are preprocessed by using mean and standard deviation. These images are resized such that the shorter side has length of 256 and are then randomly cropped to 224×224 pixels. Random horizontal flips are applied for data augmentation. The center 224×224 crop of the images are used for evaluation.

Table 3 shows the hyperparameters values used for training Hybrid MobileNets. Similar hyperparameters values are used for training baseline full-precision MobileNets and ST-MobileNets also. The learning rate scheduling involves a 'warm up' period in which the learning rate is annealed from zero to 0.2 over the first 5 epochs, after which it is gradually reduced following a cosine decay function.

B. Group of Filters with Sub-Filter Similarities

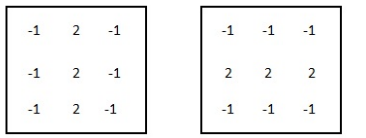


Figure 5: A 3×3 vertical lines detector and a horizontal lines detector sharing common values at all the corners and at the center.

C. Performance of Per-Layer Hybrid Filter Banks on the ResNet-20 Architecture

Table 4: Accuracy along with the computational costs, model size, and energy per inference for baseline ResNet-20, ST-ResNet-20, and ResNet-20 with hybrid filter banks on the CIFAR-10 dataset. α is the fraction of channels generated by the full-precision weight filters at each layer, c_{out} is the number of remaining channels generated by the ternary strassen filters at the corresponding convolutional layer, r is the hidden layer width of the strassenified convolutions. The last column shows the throughput of proposed models on an area-equivalent hardware accelerator comprising both MAC and adder units when compared to the throughput of baseline MobileNets with 16-bit floating-point weights on a MAC-only accelerator.

Network	Alpha (α)	r	Acc. (%)	Muls, Adds	MACs	Model size	Energy/inference (normalized)	Throughput (normalized)
ResNet-20	-	-	92.1	-	40.81M	530.64KB	1	1
ST-ResNet-20 [39]	0	$0.25c_{out}$	85.46	0.05M, 11.78M	-	21.97KB	0.05	6.92
		$0.5c_{out}$	88.63	0.1M, 23.35M	-	41.15KB	0.11	3.49
		$0.75c_{out}$	90.62	0.15M, 34.93M	-	60.32KB	0.17	2.33
		c_{out}	91.24	0.2M, 46.51M	-	79.5KB	0.23	1.75
ResNet-20 (Hybrid filter banks)	0.125	$0.25c_{out}$	88.5	0.04M, 10.18M	5.1M	85.30KB	0.17	4.0
		$0.5c_{out}$	90.39	0.09M, 20.16M	5.1M	101.83KB	0.22	2.68
		$0.75c_{out}$	91.03	0.13M, 30.14M	5.1M	118.36KB	0.27	2.02
		c_{out}	91.45	0.18M, 40.12M	5.1M	134.88KB	0.32	1.62
ResNet-20 (Hybrid filter banks)	0.25	$0.25c_{out}$	89.83	0.04M, 8.62M	10.2M	148.71KB	0.29	2.81
		$0.5c_{out}$	90.79	0.08M, 17.05M	10.2M	162.66KB	0.33	2.17
		$0.75c_{out}$	91.55	0.11M, 25.48M	10.2M	176.61KB	0.37	1.77
		c_{out}	91.79	0.15M, 33.9M	10.2M	190.56KB	0.41	1.5
ResNet-20 (Hybrid filter banks)	0.375	$0.25c_{out}$	90.36	0.03M, 7.11M	15.3M	212.18KB	0.4	2.16
		$0.5c_{out}$	91.19	0.06M, 14.02M	15.3M	223.63KB	0.44	1.82
		$0.75c_{out}$	91.38	0.09M, 20.94M	15.3M	235.07KB	0.47	1.58
		c_{out}	91.88	0.13M, 27.85M	15.3M	246.52KB	0.51	1.39

D. Comparison against Prior Works

Table 5: Top-1 and top-5 accuracy (%) of Mobilenet (full resolution and multiplier of 0.5) on Imagenet for different number of bits per weight and activation.

Method	#bits per weight/activation	Top-1 Acc. (%)	Top-5 Acc. (%)
Baseline MobileNets[4]	32/32	65.53	86.48
Baseline MobileNets[5]	16/16	65.2	86.34
ST-MobileNets ($r = 0.5c_{out}$)	2/16	48.92	73.68
ST-MobileNets ($r = 0.75c_{out}$)	2/16	56.95	80.25
ST-MobileNets ($r = c_{out}$)	2/16	61.8	83.97
ST-MobileNets ($r = 2c_{out}$)	2/16	65.14	86.26
Hybrid MobileNets ($\alpha = 0.25, r = c_{out}$)	2,16/16	63.62	84.98
Hybrid MobileNets ($\alpha = 0.25, r = 1.33c_{out}$)	2,16/16	63.47	85.11
Hybrid MobileNets ($\alpha = 0.25, r = 2c_{out}$)	2,16/16	64.84	85.86
Hybrid MobileNets ($\alpha = 0.375, r = c_{out}$)	2,16/16	64.13	85.4
Hybrid MobileNets ($\alpha = 0.375, r = 1.6c_{out}$)	2,16/16	64.17	85.38
Hybrid MobileNets ($\alpha = 0.375, r = 2c_{out}$)	2,16/16	65.2	86.05
Hybrid MobileNets ($\alpha = 0.5, r = c_{out}$)	2,16/16	64.69	85.66
Hybrid MobileNets ($\alpha = 0.5, r = 2c_{out}$)	2,16/16	65.17	85.98
Baseline MobileNets[6]	32/32	63.3	84.9
Baseline MobileNets[7]	8/8	62.2	-
Alpha-blending [27]	8/8	63	-
Alpha-blending [27]	4/8	58.4	-
HAQ [40]	2/-	57.14	81.87
Relaxed Quantization [28]	Does not demonstrate potential for MobileNets with ternary weights		
Quantization Networks [43]	Does not demonstrate potential for MobileNets with ternary weights		
Differentiable Soft Quantization [11]	Does not demonstrate potential for MobileNets with ternary weights		
Quantization Intervals [24]	Does not demonstrate potential for MobileNets with ternary weights		
Post-training 4-bit quantization [4]	Does not demonstrate potential for MobileNets with ternary weights		
Low-bit Quantization [9]	Does not demonstrate potential for MobileNets with ternary weights		