StrassenNets: Deep Learning with a Multiplication Budget

Michael Tschannen [1] **Aran Khanna** [2] **Anima Anandkumar** [2 3]

Abstract

A large fraction of the arithmetic operations required to evaluate deep neural networks (DNNs) consists of matrix multiplications, in both convolution and fully connected layers. We perform end-to-end learning of low-cost approximations of matrix multiplications in DNN layers by casting matrix multiplications as 2-layer sum-product networks (SPNs) (arithmetic circuits) and learning their (ternary) edge weights from data. The SPNs disentangle multiplication and addition operations and enable us to impose a budget on the number of multiplication operations. Combining our method with knowledge distillation and applying it to image classification DNNs (trained on ImageNet) and language modeling DNNs (using LSTMs), we obtain a first-of-a-kind reduction in number of multiplications (over 99.5%) while maintaining the predictive performance of the full-precision models. Finally, we demonstrate that the proposed framework is able to rediscover Strassen's matrix multiplication algorithm, learning to multiply 2×2 matrices using only 7 multiplications instead of 8.

1. Introduction

The outstanding predictive performance of deep neural networks (DNNs) often comes at the cost of large model size, and corresponding computational inefficiency. This can make the deployment of DNNs on mobile and embedded hardware challenging. For example, a full-precision ResNet-152 (He et al., 2016a) contains 60.2 million parameters and one forward pass requires 11.3 billion floating point operations. A variety of methods to address this issue were proposed recently, including optimizing the network architecture, factorizing the weight tensors, pruning the weights, and reducing the numerical precision of weights and activations (see Section 1.1 for a detailed overview).

These prior works mainly focused on decreasing the number of multiply-accumulate operations used by DNNs. In contrast, in this paper, the objective that guides our algorithm design is a *reduction of the number of multiplications*. This algorithm design principle has led to many fast algorithms in linear algebra, most notably Strassen's matrix multiplication algorithm (Strassen, 1969). Strassen's algorithm uses 7 instead 8 multiplications to compute the product of two 2×2 matrices (and requires $O(n^{2.807})$ operations for multiplying $n \times n$ matrices). In the context of DNNs, the same design principle led to the Winograd filter-based convolution algorithm proposed by Lavin & Gray (2016). This algorithm only requires 16 instead of 36 multiplications to compute 2×2 outputs of 2D convolutions with 3×3 kernels and achieves a 2–3$\times$ speedup on GPU in practice.

From a hardware perspective, multipliers occupy considerably more area on chip than adders (for fixed-point data types). Field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) can therefore potentially accommodate considerably more adders than multipliers, and trading off multiplications against additions is desirable. In fact, it was demonstrated recently that DNN architectures which rely on a large number of additions and a small number of multiplications (such as (Li et al., 2016)) achieve a 60% higher throughput on FPGA than on GPU, while being 2.3$\times$ better in performance per watt (Nurvitadhi et al., 2017). In the context of ASICs, reducing the number of multiplications is beneficial as multiplication operations consume significantly more energy than addition operations (3–30$\times$ depending on the data type (Horowitz, 2014; Andri et al., 2018)). More generally, replacing multiplications in DNNs by additions leads to a reduction in models size as addition/subtraction can be encoded as a binary weight. This is beneficial in terms of throughput for most deep learning applications, which are typically memory-bound.

Motivated by these observations, we propose a novel framework to drastically reduce the number of multiplications used by DNNs for inference. Specifically, for every DNN layer, we cast the (matrix) multiplication of the weight matrix with the activations as a 2-layer sum-product network

[1]ETH Zürich, Zürich, Switzerland (most of this work was done while MT was at Amazon AI) [2]Amazon AI, Palo Alto, CA, USA [3]Caltech, Pasadena, CA, USA. Correspondence to: Michael Tschannen <michaelt@nari.ee.ethz.ch>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

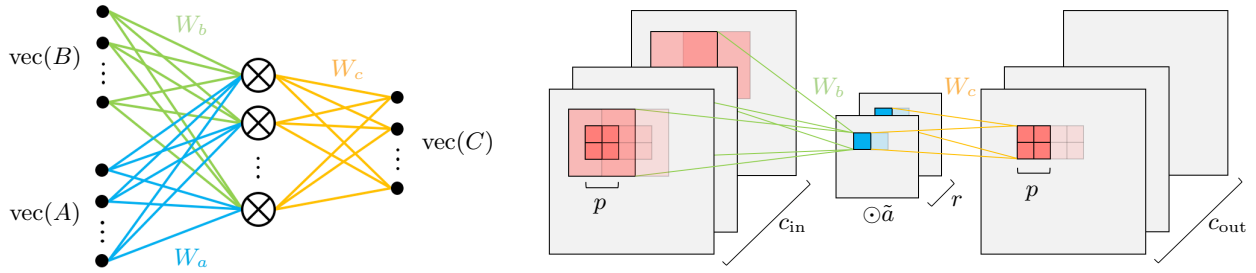

Figure 1. **Left:** Illustration of the 2-layer SPN (1), implementing an (approximate) matrix multiplication. The edges (i.e., the matrices W_a, W_b, W_c) have weights in $\mathbb{K} = \{-1, 0, 1\}$. **Right:** Application of the proposed framework to 2D convolution leads to p-strided 2D convolution with W_b, followed by channel-wise scaling by $\tilde{a} = W_a \text{vec}(A)$, followed by $1/p$-strided transposed 2D convolution with W_c.

(SPN) (arithmetic circuit). The SPNs disentangle (scalar) multiplications and additions in a way similar to Strassen's algorithm. The number of hidden units in the SPNs therefore determines the multiplication budget of the corresponding DNN layers. We then *learn the addition and multiplication operations for all layers jointly from data* by learning the edges of the SPNs, encoded as ternary $\{-1, 0, 1\}$ matrices. As the transforms realized by the SPNs are approximate and adapted to the weight matrices and distribution of the activation tensors in the DNN, this allows us to reduce the number of multiplications much more drastically than hand-engineered transforms like Strassen's algorithm or the more specialized Winograd filter-based convolution. In summary, our main contributions are the following.

- We propose a SPN-based framework for stochastic gradient-based end-to-end learning of fast approximate transforms for the arithmetic operations in DNN layers.
- Our framework allows fine-grained control of the number of multiplications and additions used at inference time, enabling precise adjustment of the tradeoff between arithmetic complexity and accuracy of DNN models.
- Extensive evaluations on CIFAR-10 and ImageNet show that our method applied to ResNet (He et al., 2016a) yields the same or higher accuracy than existing complexity reduction methods while using considerably fewer multiplications. For example, for ResNet-18 our method reduces the number of multiplications by 99.63% while incurring a top-1 accuracy degradation of only 2.0% compared to the full-precision model on ImageNet.
- Our method applied to a language model with convolution and LSTM layers (Kim et al., 2016a) results in a 99.69% reduction in multiplications while inducing an increase of only 3.3% in perplexity.
- Combining our method with knowledge distillation (KD) techniques, we obtain for the first time massive reductions in number of multiplications (99.5% and more) while maintaining the predictive performance of the full-precision models, for both image classification and language modeling.
- We demonstrate that the proposed framework is able to rediscover Strassen's algorithm, i.e., it can learn to (exactly) multiply 2×2 matrices using only 7 multiplications instead of 8.

Two key aspects of our approach that lead to gains compared previous methods are (i) our method is specifically tailored to reduce the number of multiplications whereas some previous works put more emphasis on model size reduction, and (ii) we leverage knowledge distillation which improves our results further.

We continue by reviewing related work in Section 1.1 and then describe our method and its application to 2D convolution in Section 2. A detailed numerical evaluation of our method is presented in Section 3 and concluding remarks can be found in Section 4.

1.1. Related work

We briefly review the most common approaches to compress DNNs, focusing on methods decreasing computational complexity rather than memory footprint. In all cases, there is a tradeoff between the complexity reduction and reduction in the (inference) accuracy of the compressed model.

A popular way to speed up DNNs, in particular convolutional neural networks (CNNs), is to utilize resource-efficient architectures, such as SqueezeNet (Iandola et al., 2016), MobileNet (Howard et al., 2017), and ShuffleNet (Zhang et al., 2017). SqueezeNet reduces the convolution kernel size. MobileNet and ShuffleNet rely on depth-wise separable convolutions and grouped convolutions, respectively. More sophisticated grouping and sharing techniques are studied by Wang et al. (2016).

Another strategy to accelerate CNNs is to exploit the low-rank structure prevalent in weight matrices and convolution kernels. Denton et al. (2014); Novikov et al. (2015); Kim et al. (2016b) use tensor decompositions to obtain low-rank approximations of pretrained weight matrices and filter tensors, then finetune the approximated weight matrices and filters to restore the accuracy of the compressed models.

Other works (Tai et al., 2016; Wen et al., 2017) employ low rank-promoting regularizers to further reduce the rank of the filter tensors. A framework to exploit low-rank structure in the filter responses is presented by Zhang et al. (2016).

Sparsifying filters and pruning channels are popular methods to make DNNs more efficient during inference. Wen et al. (2016) and Lebedev & Lempitsky (2016) rely on group norm-based regularizers and demonstrate their effectiveness in penalizing unimportant filters and channels, promoting hardware-friendly filter shapes, regularizing the network depth, and optimizing the filter receptive fields. Inter-channel and intra-channel redundancy is exploited by Liu et al. (2015) via a two-stage factorization procedure. An energy-aware methodology to prune filters of CNNs is described in (Yang et al., 2017).

Finally, an effective way to adapt DNNs to resource-constrained platforms is to reduce the numerical precision of their weights and/or activations. Examples for DNNs that quantize both weights and activations are DoReFa-Net (Zhou et al., 2016), XNOR-Net (Rastegari et al., 2016), and ABC-Net (Lin et al., 2017). Other works use binary weights (Courbariaux et al., 2015; Rastegari et al., 2016; Lin et al., 2017) and ternary weights (Li et al., 2016; Zhu et al., 2016) but maintain full-precision values for the activations. Keeping the activations in full precision instead of quantizing them leads to a smaller decrease in computational cost, but can yield better predictive performance.

2. Learning Fast Matrix Multiplications via SPNs

2.1. Casting matrix multiplication as SPN

Given square matrices $A, B \in \mathbb{R}^{n\times n}$, the product $C = AB$ can be represented as a 2-layer SPN

$$\mathrm{vec}(C) = W_c[(W_b\mathrm{vec}(B)) \odot (W_a\mathrm{vec}(A))] \qquad (1)$$

where $W_a, W_b \in \mathbb{K}^{r\times n^2}$ and $W_c \in \mathbb{K}^{n^2\times r}$, with $\mathbb{K} := \{-1, 0, 1\}$, are fixed, $\mathrm{vec}(D)$ stands for vectorization of the matrix $D = [d_1 \dots d_m] \in \mathbb{R}^{k\times m}$, i.e., $\mathrm{vec}(D) = [d_1^\top \dots d_m^\top]^\top \in \mathbb{R}^{km}$, and $\odot$ denotes the element-wise product. The SPN (1) disentangles additions (and subtractions), encoded in the ternary matrices W_a, W_b, and W_c, and multiplications, realized exclusively by the operation $\odot$ (see Fig. 1, left). The width of the hidden layer of the SPN, r, hence determines the number of multiplications used for the matrix multiplication. A naïve implementation of the matrix multiplication AB requires $r = n^3$. For $n = 2$,[1] Strassen's matrix multiplication algorithm (Strassen, 1969) specifies the following set of weights that satisfy (1) for $r = 7$ (instead of $r = 8$)

$$W_a = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix}, \; W_b = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix},$$

$$W_c = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}. \qquad (2)$$

An interesting tensor perspective on the SPN (1) (not explored in-depth here) is common in the context of algebraic complexity theory. Specifically, (1) can be written as

$$\mathrm{vec}(C)_i = \sum_{k=1}^{n^2}\sum_{l=1}^{n^2}(M_n)_{i,k,l}\mathrm{vec}(A)_k\mathrm{vec}(B)_l, \text{ where}$$

$$(M_n)_{i,k,l} = \sum_{j=1}^{r}(W_c)_{i,j}(W_a)_{j,k}(W_b)_{j,l}.$$

M_n is the $(n \times n)$-*matrix multiplication tensor*, and r hence corresponds to the rank of M_n. It is known that $\mathrm{rank}(M_2) = 7$ and $19 \leq \mathrm{rank}(M_3) \leq 23$, see (Elser, 2016) for more details and references.

Elser (2016) explores learning exact matrix multiplications via SPNs of the form (1) for $n = 2$ and $n = 3$ from synthetic data. Thereby, the elements of W_a, W_b, and W_c are relaxed to real numbers instead of elements from $\mathbb{K}$. Note that this relaxation leads to an increase in the number of multiplications in general. In contrast, we integrate SPNs with weights from $\mathbb{K}$ into DNN layers and learn them end-to-end (see next section), realizing actual reductions in multiplications.

2.2. Learning fast approximate matrix multiplications for DNNs

Writing matrix products in the form (1) is not specific to square matrices. Indeed, it is easy to see that $r \geq nmk$ is a sufficient condition for the existence of matrices W_a, W_b, W_c with elements in $\mathbb{K}$ such that the product of any two matrices $A \in \mathbb{R}^{k\times m}$ and $B \in \mathbb{R}^{m\times n}$, including matrix-vector products (i.e., $n = 1$), can be written in the form (1). When the matrices A and B are drawn from probability distributions that concentrate on low-dimensional manifolds of $\mathbb{R}^{k\times m}$ and $\mathbb{R}^{m\times n}$, respectively, or if one of the matrices is fixed, it may be possible to find W_a and W_b that satisfy the equality in (1) approximately even when $r \ll nmk$. In this case, (1) approximately computes the product AB while considerably reducing the number of multiplications compared to the naïve implementation. Furthermore, by imposing structure (such as, e.g., sparsity or

[1] The formulation by Strassen (1969) is more general, applying recursively to 4 equally-sized subblocks of square matrices, with the 2×2 case occurring at maximal recursion depth.

block-diagonal structure) into the matrices W_a, W_b, W_c one can tailor sharing or grouping of the operations for the application or platform at hand.

In this paper, we leverage this concept to accelerate and compress the matrix multiplications in DNN layers for inference. Specifically, for layer ℓ, we associate A with the (pretrained) weights/filters W_ℓ and B with the corresponding activations/feature maps F_ℓ. The ternary matrices W_a, W_b, and W_c are then learned end-to-end using a stochastic gradient-based optimizer (one set of weights W_a, W_b, W_c for each layer). After training, W_a and $\mathrm{vec}(A)$ can be collapsed into a vector $\tilde{a} = W_a \mathrm{vec}(A) \in \mathbb{R}^r$ as they are both fixed during inference. Alternatively, $\tilde{a} \in \mathbb{R}^r$, W_b, and W_c can be learned jointly from scratch. The choice of r determines the tradeoff between the computational cost in terms of multiplications and the precision of the the approximate matrix multiplication, and hence the predictive performance of the network. This approach requires r full-precision parameters and $rm(k+n)$ ternary weight parameters. It reduces the number of multiplications by a factor of mnk/r.

Quantizing the elements of W_a, W_b, and W_c to $\mathbb{K}$ during training poses a challenge as quantization is non-differentiable. Different approaches were proposed to overcome this issue (Courbariaux et al., 2015; Li et al., 2016; Rastegari et al., 2016; Zhu et al., 2016; Agustsson et al., 2017). Here, we adopt the method from (Li et al., 2016) and briefly describe it for quantizing W_a (W_b and W_c are quantized in exactly the same way). Specifically, this method maintains a full-precision version W_a^{fp} of W_a during training and quantizes W_a^{fp} in every forward pass by approximately solving the optimization problem

$$\alpha^*, W_a^{\mathrm{t}*} = \underset{\alpha, W_a^{\mathrm{t}}}{\arg\min} \|W_a^{\mathrm{fp}} - \alpha W_a^{\mathrm{t}}\|_F^2 \quad \text{s.t.} \quad \alpha > 0, \quad W_a^{\mathrm{t}} \in \mathbb{K}^{r \times km}, \tag{3}$$

and by setting $W_a = \alpha^* W_a^{\mathrm{t}*}$ (the scaling factors α^* for W_a, W_b, W_c can be absorbed by A or $\tilde{a}$ after training to ensure that W_a, W_b, W_c have elements in $\mathbb{K}$). During the backward pass the quantization function is replaced by the identity function, and the gradient step is applied to W_a^{fp}. Assuming i.i.d. Gaussian weights, Li et al. (2016) derive the approximate solution

$$(W_a^{\mathrm{t}*})_{i,j} = \begin{cases} 1 & \text{if } (W_a^{\mathrm{fp}})_{i,j} > \Delta, \\ -1 & \text{if } (W_a^{\mathrm{fp}})_{i,j} < -\Delta, \\ 0 & \text{otherwise}, \end{cases}$$
$$\alpha^* = \frac{\sum_{(i,j):\,(W_a^{\mathrm{t}*})_{i,j} \neq 0} |(W_a^{\mathrm{fp}})_{i,j}|}{\sum_{i,j} |(W_a^{\mathrm{t}*})_{i,j}|} \tag{4}$$

to (3), where $\Delta = \frac{0.7}{kmr} \sum_{i,j} |(W_a^{\mathrm{fp}})_{i,j}|$. While our framework would allow quantized training from scratch with fixed threshold Δ and fixed quantization level α (e.g., $\Delta = 0.5$ and $\alpha = 1$), we observed that relying on the scheme (4) allows us to pretrain W_a^{fp}, W_b^{fp}, W_c^{fp} without quantization, and then activate quantization to stably continue training. We found that this strategy leads to faster training while inducing no loss in accuracy.

Besides the fully connected case described in this section, we particularize the proposed approach for 2D convolutions for image classification DNNs. We emphasize that any DNN layer operation reducible to a general matrix multiplication (GEMM) can be cast into the form (1), including n-dimensional convolutions, group (equivariant) convolutions (when implemented as a filter bank) (Cohen & Welling, 2016), and deformable convolutions (Dai et al., 2017).

2.3. Knowledge distillation (KD)

KD refers to the process of training a student network using a larger (in terms of the number of layers and hidden units) teacher network (Bucilua et al., 2006; Hinton et al., 2014). As a result, the student network typically has the same or slightly better predictive performance than the teacher network, despite being less complex. KD for training a low-precision student network from a full-precision teacher network with the same architecture and hyper parameters as the student network was investigated recently in (Mishra & Marr, 2018; Zhuang et al., 2018; Polino et al., 2018). Here, we explore the same avenue to improve the predictive performance of networks compressed with our method. Specifically, we follow the method proposed in (Hinton et al., 2014) using the cross entropy between the student softmax output and the teacher softmax output as KD loss term. We set the softmax temperature parameter to 1 throughout and assign the same weight to the KD loss term as to the original loss. For sequence models, we simply apply the described KD loss to the softmax outputs of the unrolled teacher and student models (more sophisticated techniques were proposed in (Kim & Rush, 2016)).

2.4. Application to 2D convolution

Consider the ℓth 2D convolution layer of a CNN applying c_{out} filters of dimension $w \times h \times c_{\mathrm{in}}$ to a feature representation F_ℓ of dimension $W \times H \times c_{\mathrm{in}}$ (width×height×number of channels). To write the computation of all c_{out} output channels as a matrix multiplication, each feature map in F_ℓ is decomposed into WH patches of size $w \times h$ (after appropriate padding) and the vectorized patches are arranged in a matrix $\tilde{F}_\ell$ of dimension $whc_{\mathrm{in}} \times WH$. This transformation is usually referred to as `im2col`, see (Sze et al. (2017), Fig. 19) for an illustration. Accordingly, the filters for all output channels are vectorized and jointly reshaped into a $c_{\mathrm{out}} \times whc_{\mathrm{in}}$ matrix $\tilde{W}_\ell$. The vectorized layer output (before activation) for all c_{out} output channels is obtained as $\tilde{W}_\ell \tilde{F}_\ell$ and has dimension $c_{\mathrm{out}} \times WH$. In principle, one

can now compress the operation $\tilde{W}_\ell \tilde{F}_\ell$ using our method by setting $A = \tilde{W}_\ell$, $B = \tilde{F}_\ell$, plugging them into (1), and proceeding as described in Section 2.2. However, this results in impractically large W_a, W_b, and W_c and ignores the weight sharing structure of the convolution. By associating A with $\tilde{W}_\ell$ and B with single columns of $\tilde{F}_\ell$ we can jointly compress the computations across all input and output channels, while preserving the spatial structure of the convolution. The resulting SPN realizes a convolution with r ternary $w \times h \times c_{\text{in}}$ filters (the rows of W_b), followed by a channel-wise scaling with $\tilde{a} = W_a \text{vec}(\tilde{W}_\ell)$, followed by convolution with a ternary $1 \times 1 \times r$ filter for each of the c_{out} outputs (the rows of W_c) see Fig. 1, right.

To realize *local spatial compression*, we partition the computation of the convolution into subsets corresponding to square output patches. In more detail, we consider the computation of $p \times p$ convolution output patches from $(p-1+w) \times (p-1+h)$ input patches, offset by a stride of p, and approximate this computation with a SPN jointly for all channels. As a result, the number of multiplications is reduced both spatially and across channels. For example, for 3×3 convolution filters, we divide the input feature maps into 4×4 spatial patches with a stride of 2, such that the SPN computes $2 \times 2 \times c_{\text{out}}$ outputs from $4 \times 4 \times c_{\text{in}}$ elements of F_ℓ. Thereby, W_c realizes a $2 \times 2 \times r$ transposed convolution with a stride of $1/2$ (see Fig. 1, right, and pseudocode in Appendix C). For fixed r, this reduces the number of multiplications by a factor of 4 compared to the case without spatial compression (i.e., $p = 1$).

In summary, the described compression of 2D convolution leads to a reduction of the number of multiplications by a factor $c_{\text{in}} c_{\text{out}} whp^2/r$ compared to the standard implementation of the convolution.

Finally, to reduce the number of additions realized through W_b (and thereby the number of nonzero elements of W_b) by a factor of g, we implement W_b as grouped convolution, originally introduced in (Krizhevsky et al., 2012). Specifically, the convolution realized by W_b is assumed to consist of g independent 2D convolutions each with c_{in}/g input channels and r/g output channels. In other words, W_b is assumed to be block-diagonal with blocks of dimension $(r/g) \times (whc_{\text{in}}/g)$.

Relation to prior work in the 2D convolution case. Binary weight networks (BWNs) (Rastegari et al., 2016) and ternary weight networks (TWNs) (Li et al., 2016) rely on binary $\{-1, 1\}$ and ternary $\{-1, 0, 1\}$ weight matrices, respectively, followed by (full-precision) rescaling of the activations (see Section 2.2) and are special cases of our framework. ABC-Nets (Lin et al., 2017) approximate the full-precision weight matrices as a weighted sum of multiple binary $\{-1, 1\}$ weight matrices and can also be cast as (structured) SPNs. However, we do not directly recover the trained ternary quantization (TTQ) approach from (Zhu et al., 2016), which relies on asymmetric ternary weights $\{-c_1, 0, c_2\}$, $c_1, c_2 > 0$. Finally, note that Winograd filter-based convolution (Lavin & Gray, 2016) realizes spatial compression over 2×2 output patches but performs exact computation and does not compress across channels.

3. Experiments[2]

3.1. Rediscovering Strassen's algorithm

Before applying the proposed method to DNNs, we demonstrate that it is able to rediscover Strassen's algorithm, i.e., it can learn to multiply 2×2 matrices using only 7 multiplications instead of 8 (which implies a recursive algorithm for larger matrices). This problem was previously studied by Elser (2016), but for *real-valued* W_a, W_b, W_c, which increases the number of multiplications in general when using these matrices in (1) to compute matrix products. In contrast, our method learns $W_a, W_b \in \mathbb{K}^{7\times 4}$, $W_c \in \mathbb{K}^{4\times 7}$ (i.e., the discrete solution space has size $3^{3\cdot 4\cdot 7} = 3^{84}$), and hence leads to an actual reduction in the number of multiplications.

We generate a training set containing 100k pairs (A_i, B_i) with entries i.i.d. uniform on $[-1, 1]$, train the SPN 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 another epoch (with learning rate 0.001). Around 25 random initializations are necessary to obtain convergence to zero training L2-loss after activation of the quantization; for most initializations the training L2-loss converges to a positive value. A set of ternary weight matrices implementing an *exact* matrix multiplication, found by our method, is

$$W_a = \begin{pmatrix} -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}, \quad W_b = \begin{pmatrix} -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ -1 & -1 & -1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & -1 & 0 \end{pmatrix},$$

$$W_c = \begin{pmatrix} 1 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 1 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

3.2. Image classification

We apply our method to all convolution layers (including the first convolution layer and the projection layers for subsampling) of the ResNet architecture (He et al., 2016a) to create the so-called Strassen-ResNet (ST-ResNet). We evaluate

[2]Code available at https://github.com/mitscha/strassennets.

ST-ResNet on CIFAR-10 (10 classes, 50k training images, 10k testing images) (Krizhevsky & Hinton, 2009) and ImageNet (ILSVRC2012; 1k classes, 1.2M training images, 50k testing images) (Russakovsky et al., 2015) for different choices of r, p, g, and compare the accuracy of ST-ResNet to related works. All models were trained from scratch, meaning we directly learn $\tilde{a} = W_a \text{vec}(A)$ rather than associating A with the weights of pretrained networks and learning W_a. Throughout the training process we used SGD with momentum 0.9 and weight decay 10^{-4}. As most related works involving ternary weights do not report sparsity levels, to facilitate comparisons, we do not make any assumption about the number of zeros among ternary weights. It is the sparsity of the activations, not the weights, that directly impacts the number of multiplications (the focus of this paper). All model sizes are computed without (lossless) compression of the network parameters.

3.2.1. CIFAR-10

We consider ST-ResNet-20 and employ the data augmentation procedure described in (He et al. (2016a), Sec. 4.2.). We train for 250 epochs with initial learning rate 0.1 and mini-batch size 128, multiplying the learning rate by 0.1 after 150 and 200 epochs. We then activate quantization for W_b and W_c, set the learning rate to 0.01 and train the network for 40 epochs, multiplying the learning rate by 0.1 every 10 epochs. Finally, we fix the (now ternary) W_b and W_c and continue training for another 10 epochs. The resulting testing accuracy is shown in Table 1 for different r and p, along with the reduction in the number of multiplications compared to the uncompressed model (for the 32×32 CIFAR-10 images; see Table 5 in Appendix D for the reduction in the number of additions). Additional results for a similar experiment based on the VGG-inspired 7-layer architecture considered in (Courbariaux et al., 2015; Li et al., 2016) can be found in Appendix A.

Table 1. **Left:** Testing accuracy (in %) of compressed ResNet-20 on CIFAR-10. **Right**: Reduction in the number of multiplications.

	testing accuracy					red. in multiplications				
	r					r				
p	c_{out}	$\frac{3}{4}c_{\text{out}}$	$\frac{1}{2}c_{\text{out}}$	$\frac{1}{4}c_{\text{out}}$	p	c_{out}	$\frac{3}{4}c_{\text{out}}$	$\frac{1}{2}c_{\text{out}}$	$\frac{1}{4}c_{\text{out}}$
1	91.24	90.62	88.63	85.46	1	98.96	99.08	99.21	99.33
2	89.87	89.47	87.31	84.01	2	99.33	99.36	99.39	99.42
4	86.13	84.67	82.67	75.01	4	99.42	99.43	99.44	99.44

Discussion. The model obtained for the base configuration with $r = c_{\text{out}}$ and $p = 1$ incurs a negligible accuracy loss compared to the uncompressed ResNet-20 with an accuracy of 91.25% (He et al., 2016a) while reducing the number of multiplications by 98.96% (the evaluation of the uncompressed ResNet-20 requires 41.038M multiply-adds). This model also matches the accuracy of TTQ (Zhu et al., 2016) for ResNet-20 while requiring fewer multiplications (TTQ does not quantize the first convolution layer). As r decreases and/or p increases, the number of multiplications decreases at the cost of further accuracy reduction.

3.2.2. ImageNet

We consider ST-ResNet-18 and, unlike for the experiment on CIFAR-10, we also compress the last (fully connected) layer of ST-ResNet-18 for models with $r \leq c_{\text{out}}$ in convolution layers, setting $r = 1000$ for that layer throughout (we observed that compressing the last layer when $r > c_{\text{out}}$ in convolution layers leads to a considerable reduction in validation accuracy). Following (Rastegari et al., 2016; Li et al., 2016; Zhu et al., 2016), the training images are resized such that the shorter side has length 256 and are then randomly cropped to 224×224 pixels. The validation accuracy is computed from center crops. We use an initial learning rate of 0.05 and mini-batch size 256, with two different learning rate schedules depending on the value of r in the convolution layers: We train for 40 epochs without quantization, multiplying the learning rate by 0.1 after 30 epochs, if $r \leq c_{\text{out}}$, and for 70 epochs, multiplying the learning rate by 0.1 after 40 and 60 epochs, otherwise. Thereafter, we activate quantization and continue training for 10 epochs. Finally, we fix W_b and W_c and train $\tilde{a}$ for another 5 epochs.

In Table 2 we report the validation accuracy of ST-ResNet-18 for different r, p, and g, and the validation accuracy obtained with KD. Table 3 shows the reduction in the number of multiplications compared to the original ResNet-18 model, for different r, p, and g (see Table 7 in Appendix D for reductions in the number of additions and model size). In Fig. 2, we plot the accuracy of ST-ResNet-18 for different r, p, and g, as a function of the number of operations and model size. In addition, we report the validation accuracy for related works (Rastegari et al., 2016; Li et al., 2016; Zhu et al., 2016; Lin et al., 2017) (see also Table 8 in Appendix D). We do not consider $p > 2$ as this leads to (ternary) convolution with impractically large kernels for 224×224 images.

Finally, to demonstrate amenability of our method to larger models, we trained ST-ResNet-34 with $r = 2c_{\text{out}}$, $p = 2$, $g = 1$ (without tuning any hyper parameters) and obtained 69.2%/88.5% top-1/top-5 validation accuracy without KD and 71.9%/90.5% with KD (the full-precision model obtains 73.3%/91.3%; we report the accuracies of the Torch pretrained models for all full-precision ResNets).

Discussion. All ST-ResNet-18 models that require the same number of multiplications as TWN (those with $r = c_{\text{out}}$, $p = 1$, $g = 4$; $r = c_{\text{out}}$, $p = 1$, $g = 1$; $r = 2c_{\text{out}}$, $p = 2$, $g = 1$) obtain a considerably higher top-1 and top-5 accuracy than TWN. In particular, ST-ResNet-18 with $r = 2c_{\text{out}}$, $p = 2$, and $g = 1$ leads to a 7.0% improvement

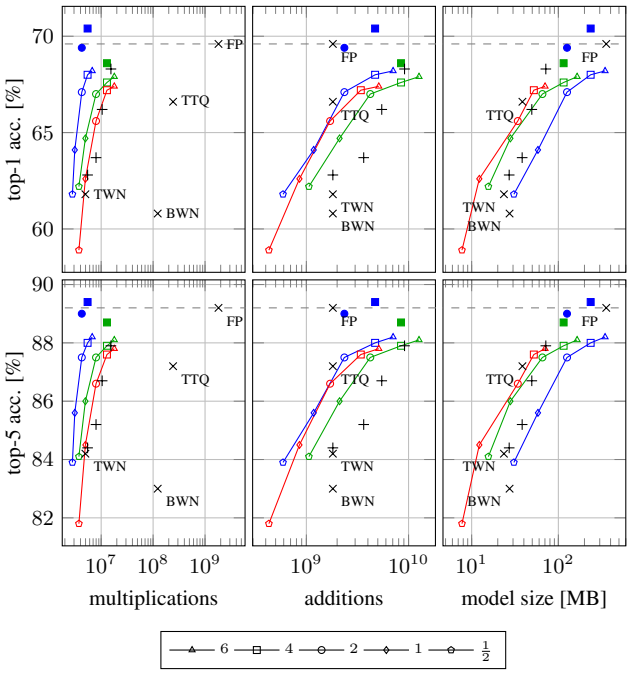

Figure 2. Top-1 and top-5 validation accuracy of ST-ResNet-18 on ImageNet as a function of the number of multiplications, the number of additions, and model size, along with the values obtained in related works BWN (Rastegari et al., 2016), TWN (Li et al., 2016), TTQ (Zhu et al., 2016), ABC-Net-1/2/3/5 (Lin et al., 2017) ("+" signs, the suffix reflects the ranking according to accuracy), and the full-precision model (FP). The numbers associated with the marker types correspond to the ratio of the number of hidden SP units and output channels, r/c_{out}. Different colors indicate different combinations of output patch size p and number of convolution groups g: Blue: $p = 2$, $g = 1$; green: $p = 1$, $g = 1$; red: $p = 1$, $g = 4$. Selected models trained with KD are shown with filled markers.

in top-1 accuracy. Furthermore, ST-ResNet-18 with $r = 2c_{\text{out}}$, $p = 1$, and $g = 1$ outperforms TTQ while using 98.3% fewer multiplications. ST-ResNet-18 with $r = 6c_{\text{out}}$, $p = 2$, and $g = 1$ incurs a 2.0% reduction in top-1 accuracy compared to the full-precision model while reducing the number of multiplications by 99.63%. Our ST-ResNets require fewer multiplications and additions than ABC-Net-1/2/3 while yielding the same accuracy. For $p = 2$, our models lead to a reduction in multiplications of at least 50% compared to the ABC-Net with the same accuracy. Note that TTQ and BWN use considerably more multiplications than ST-ResNet-18, TWN, and ABC-Net as they do not quantize the first convolution layer.

In contrast to the experiments on CIFAR-10, increasing p from 1 to 2 increases the accuracy for fixed $r \geq 2c_{\text{out}}$. A possible explanation for this behavior is that the benefits of the increase in the number of ternary parameters obtained by increasing p outweighs the loss in precision due to the reduction in spatial resolution. This is in accordance with the fact that the images in ImageNet are much larger than those in CIFAR-10, resulting in larger feature maps for most layers.

Table 2. Top-1 and top-5 validation accuracy (in %) of ST-ResNet-18 on ImageNet, for different choices of r, p, g, and with KD.

top-1 accuracy

	r					r (KD)		
(p, g)	$6c_{\text{out}}$	$4c_{\text{out}}$	$2c_{\text{out}}$	c_{out}	$\frac{1}{2}c_{\text{out}}$	$4c_{\text{out}}$	$2c_{\text{out}}$	c_{out}
$(1, 1)$	67.9	67.6	67.0	64.7	62.2	68.6	67.9	66.0
$(2, 1)$	68.2	68.0	67.1	64.1	61.8	70.4	69.4	66.4
$(1, 4)$	67.4	67.2	65.6	62.6	58.9	68.0	66.6	63.9

top-5 accuracy

	r					r (KD)		
(p, g)	$6c_{\text{out}}$	$4c_{\text{out}}$	$2c_{\text{out}}$	c_{out}	$\frac{1}{2}c_{\text{out}}$	$4c_{\text{out}}$	$2c_{\text{out}}$	c_{out}
$(1, 1)$	88.1	87.9	87.5	86.0	84.1	88.7	88.3	87.1
$(2, 1)$	88.2	88.0	87.5	85.6	83.9	89.4	89.0	87.3
$(1, 4)$	87.8	87.6	86.6	84.5	81.8	88.3	87.5	85.5

Table 3. Reduction in the number of multiplications (in %) of ST-ResNet-18 compared to the full-precision model, for 224×224 images.

red. in multiplications

	r				
(p, g)	$6c_{\text{out}}$	$4c_{\text{out}}$	$2c_{\text{out}}$	c_{out}	$\frac{1}{2}c_{\text{out}}$
$(1, 1)$	99.01	99.29	99.56	99.73	99.79
$(2, 1)$	99.63	99.70	99.77	99.83	99.85
$(1, 4)$	99.01	99.29	99.56	99.73	99.79

KD leads to improvements in top-1 accuracy of 1.3–3.5%, see Table 2. In particular, ST-ResNet-18 with $r = 2c_{\text{out}}$, $p = 2$, and $g = 1$ trained using KD essentially matches the accuracy of the full-precision model. Increasing r to $4c_{\text{out}}$ yields a model that even outperforms the full-precision model. To the best of our knowledge, these models are the first to realize massive reductions in the number of multiplications (over 99.5%) while maintaining the accuracy of the full-precision model. Note that student models outperforming their teachers were observed before in different contexts (Mishra & Marr, 2018; Zhuang et al., 2018; Furlanello et al., 2018).

For some of the configurations, the reduction in multiplications comes at the cost of a small to moderate increase in the number of additions. We emphasize that this is also the case for Strassen's algorithm (see (2)) and the Winograd filter-based convolution (see (Lavin & Gray, 2016), Sec. 4.1). The specific application and target platform will determine what increase in the number of additions is acceptable.

Finally, in all our image classification experiments the ratio r/c_{out} is the same for all layers. Since one would expect improvements from allocating more multiplications to layers that require more accurate operations, we also tested

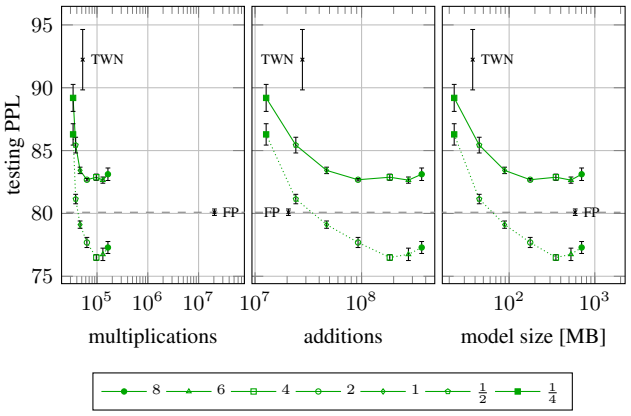

Figure 3. Testing PPL (averaged over 5 runs) for ST-LM as a function of the number of operations and model size, along with the values obtained for TWN quantization (4), and the full-precision model (FP). Solid line: Without KD; dotted line: With KD. The numbers associated with the marker types correspond to the ratio of the number of hidden SP units and hidden units, r/n_{out}.

a simple way to learn the ratio r/c_{out} for each layer from data. Specifically, we chose a large r/c_{out} and applied L1 regularization to the vectors $\tilde{a}$. However, for a given total multiplication budget this strategy led to lower accuracy in our experiments than just fixing r/c_{out} for all layers.

3.3. Language modeling

We apply our method to the character-level language model described in (Kim et al., 2016a) and evaluate it on the English Penn Treebank (PTB with word vocabulary size 10k, character vocabulary size 51, 1M training tokens, standard train-validation-test split, see (Kim et al., 2016a)) (Marcus et al., 1993). We use the large version of the model from (Kim et al., 2016a) which is composed of a convolution layer with 1100 filters (applied to a character-level representation of the words, without aggregation across channels), followed by a 2-layer highway network with 1100 hidden units, feeding into a 2-layer LSTM network with 650 hidden units (see Table 2 in (Kim et al., 2016a) for more details). We obtain Strassen language models (ST-LMs) by replacing the convolution layer and all fully connected layers (both within the LSTM and the output/decode layer) with SPNs. r is set to the number of filters for the convolution layer and is parametrized as $r(\kappa) = \kappa \cdot n_{\text{out}}$ for the fully connected layers, where n_{out} is the number of hidden units. For the output/decode layer we use $r(\kappa) = \kappa \cdot 2000$.

All models are trained for 40 epochs using SGD with mini-batch size 20 and initial learning rate 2, multiplying the learning rate by 0.5 when the validation perplexity per word (PPL; c.f. Eq. (9) in (Kim et al., 2016a)) decreases by less than 0.5 per epoch (a similar schedule was used in (Kim et al., 2016a)). Although the ST-LMs train stably with quantization from scratch, we train them for 20 epochs with full-precision weights before activating quantization for W_b and W_c, which leads to slightly lower validation PPLs. As a baseline, we consider the TWN quantization scheme (4) and apply it to all layers of the language model. As we observed a somewhat higher variability in the validation performance than for the image classification experiments, we train each quantized model 5 times and report the average testing PPL.

In Figure 3, we plot the average testing PPL of our ST-LMs for different r as a function of the number of operations and model size, with and without KD. Table 6 in Appendix D shows the reduction in the number of operations and model size compared to the full-precision model.

Discussion. Our ST-LM models reduce the number of multiplications by over 99% compared to the full-precision model, while incurring an increase in testing PPL of only 3–4%. The PPL obtained via TWN quantization clearly exceeds that of all considered ST-LMs. The ST-LM model with $r = n_{\text{out}}$ requires roughly the same number of multiplications as the TWN model but has a 7.4% lower testing PPL. KD leads to a significant reduction in testing PPL. The distilled ST-LMs outperform the teacher model for $r \geq n_{\text{out}}$. To our knowledge, our models are the first to obtain such massive reductions (over 99.5% for $r \leq 4n_{\text{out}}$) in the number of multiplications while maintaining the PPL of the full-precision model. We observed that KD applied to the teacher model also reduces its testing PPL to values comparable to that of the compressed models with KD for $r \geq n_{\text{out}}$ (see (Furlanello et al., 2018) for more exploration of this phenomenon). On the other hand, KD considerably increases the testing PPL for TWN.

There are only few prior works on compressing sequence models in a *multiplication-reducing* fashion (He et al., 2016b; Hubara et al., 2016). For single-layer LSTM and GRU language models with binary weights He et al. (2016b) report an increase in PPL of 70% and more compared to the full-precision model, whereas Hubara et al. (2016) observed divergence for a single-layer LSTM model with binary weights, but report small degradations for 4-bit weight and activation quantization.

4. Conclusion and Future Work

We proposed and evaluated a versatile framework to learn fast approximate matrix multiplications for DNNs end-to-end. We found that our method leads to the same or higher accuracy compared to existing methods while using significantly fewer multiplications. By leveraging KD we were able to train models that incur no loss in predictive performance despite reducing the number of multiplications by over 99.5%. A natural next step is to incorporate activation quantization into the proposed method. In addition, it will be interesting to see how the theoretical gains reported here translate into actual energy savings and runtime speedups on specialized hardware such as FPGAs and ASICs.

Acknowledgment

The authors would like to thank Eirikur Agustsson, Helmut Bölcskei, Lukas Cavigelli, Asmus Hetzel, Risi Kondor, Andrew Lavin, Michael Lerjen, Zachary Lipton, Weitang Liu, Andrea Olgiati, John Owens, Sheng Zha, and Zhi Zhang for inspiring discussions and comments. This work was supported by the "AWS Cloud Credits for Research" program.

References

Agustsson, E., Mentzer, F., Tschannen, M., Cavigelli, L., Timofte, R., Benini, L., and Gool, L. V. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1141–1151, 2017.

Andri, R., Cavigelli, L., Rossi, D., and Benini, L. YodaNN: An architecture for ultralow power binary-weight CNN acceleration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):48–60, Jan 2018.

Bucilua, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *Proc. Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pp. 535–541, 2006.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *Proc. Int. Conf. on Machine Learning (ICML)*, pp. 2990–2999, 2016.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3123–3131, 2015.

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., and Wei, Y. Deformable convolutional networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 764–773, 2017.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1269–1277, 2014.

Elser, V. A network that learns Strassen multiplication. *Journal of Machine Learning Research (JMLR)*, 17(116): 1–13, 2016.

Furlanello, T., Lipton, Z. C., Tschannen, M., Itti, L., and Anandkumar, A. Born again neural networks. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016a.

He, Q., Wen, H., Zhou, S., Wu, Y., Yao, C., Zhou, X., and Zou, Y. Effective quantization methods for recurrent neural networks. *arXiv:1611.10176*, 2016b.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *NIPS Deep Learning Workshop, arXiv:1503.02531*, 2014.

Horowitz, M. Computing's energy problem (and what we can do about it). In *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*, pp. 10–14, 2014.

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. *arXiv:1704.04861*, 2017.

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

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv:1602.07360*, 2016.

Inan, H., Khosravi, K., and Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. In *Int. Conf. on Learning Representations (ICLR)*, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. Int. Conf. on Machine Learning (ICML)*, pp. 448–456, 2015.

Kim, Y. and Rush, A. M. Sequence-level knowledge distillation. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1317–1327, 2016.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. Character-aware neural language models. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, pp. 2741–2749, 2016a.

Kim, Y.-D., Park, E., Yoo, S., Choi, T., Yang, L., and Shin, D. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Int. Conf. on Learning Representations (ICLR)*, 2016b.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

Lavin, A. and Gray, S. Fast algorithms for convolutional neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 4013–4021, 2016.

Lebedev, V. and Lempitsky, V. Fast convnets using group-wise brain damage. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 2554–2564, 2016.

Li, F., Zhang, B., and Liu, B. Ternary weight networks. *NIPS Workshop on Efficient Methods for Deep Neural Networks (EMDNN), arXiv:1605.04711*, 2016.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 344–352, 2017.

Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. Sparse convolutional neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 806–814, 2015.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Mishra, A. and Marr, D. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *Int. Conf. on Learning Representations (ICLR)*, 2018.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 442–450, 2015.

Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J., Liew, Y. T., Srivatsan, K., Moss, D., Subhaschandra, S., and Boudoukh, G. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In *Proc. ACM/SIGDA Int. Symposium on Field-Programmable Gate Arrays*, pp. 5–14, 2017.

Polino, A., Pascanu, R., and Alistarh, D. Model compression via distillation and quantization. *Int. Conf. on Learning Representations (ICLR)*, 2018.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European Conf. on Computer Vision (ECCV)*, pp. 525–542, 2016.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Strassen, V. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. of the IEEE*, 105(12):2295–2329, 2017.

Tai, C., Xiao, T., Zhang, Y., and Wang, X. Convolutional neural networks with low-rank regularization. In *Int. Conf. on Learning Representations (ICLR)*, 2016.

Wang, M., Liu, B., and Foroosh, H. Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial "bottleneck" structure. *arXiv:1608.04337*, 2016.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2074–2082, 2016.

Wen, W., Xu, C., Wu, C., Wang, Y., Chen, Y., and Li, H. Coordinating filters for faster deep neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 658–666, 2017.

Yang, T.-J., Chen, Y.-H., and Sze, V. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 5687–5695, 2017.

Zhang, X., Zou, J., He, K., and Sun, J. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 38(10):1943–1955, 2016.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv:1707.01083*, 2017.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv:1606.06160*, 2016.

Zhu, C., Han, S., Mao, H., and Dally, W. J. Trained ternary quantization. In *Int. Conf. on Learning Representations (ICLR)*, 2016.

Zhuang, B., Shen, C., Tan, M., Liu, L., and Reid, I. Towards effective low-bitwidth convolutional neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

A. Additional Results on CIFAR-10

We apply our method to the 7-layer VGG-inspired architecture previously considered in (Courbariaux et al., 2015; Li et al., 2016) (see (Li et al. (2016), Sec. 3) for a detailed description of the architecture) and evaluate it on CIFAR-10. We vary r and p for convolution layers and fix $r = 1024$ for the fully connected layer with 1024 units feeding into the softmax layer. The same hyper parameters and schedules as for ST-ResNet-20 are used for training, see Sec. 3.2.1. Table 4 shows the testing accuracy for different r and p. Our method achieves the same or higher accuracy than TWN (Li et al., 2016) for $p = 1$. The impact of increasing p on testing accuracy is analogous to that observed for ST-ResNet-20. Reducing r seems to reduce testing accuracy to a smaller extent than for ST-ResNet-20. A possible reason for this could be that the considered VGG architecture has considerably wider layers (128 to 512 channels) than ResNet-20 (16 to 64 channels).

Table 4. Testing accuracy (in %) of the 7-layer VVG model from (Courbariaux et al., 2015; Li et al., 2016) compressed by our method, on CIFAR-10.

testing accuracy				
	r			
p	c_{out}	$\frac{3}{4}c_{\text{out}}$	$\frac{1}{2}c_{\text{out}}$	$\frac{1}{4}c_{\text{out}}$
1	93.17	93.19	92.39	92.50
2	91.87	91.44	91.39	89.66
4	88.08	88.40	87.65	87.05

B. Additional Results for Language Modeling

To assess the generalization of the ST-LM models described in Section 3.3, we apply the FP and ST-LM (with $r = 2n_{\text{out}}$) models to Wikitext-2[3] (word vocabulary size 33k and 2M training tokens) without changing hyper parameters and obtain a testing PPL of 90.07, 97.40, and 87.72 for the FP model, the ST-LM model, and the ST-LM model with KD, respectively. The PPL of the FP model (19M parameters) is comparable to that of the variational dropout LSTM (VD-LSTM-RE, 22M parameters) from (Inan et al., 2017). While the gap between the FP and ST-LM model is larger than for PTB, the ST-LM model with distillation outperforms the FP model similarly as for PTB.

C. Application to 2D convolution: Pseudocode

In this section, we provide provide pseudocode to facilitate the implementation of the proposed framework for 2D convolutions with $k \times k$ kernels (see Section 2.4) in popular deep learning software packages. Let `W_B`, `a_tilde`, and `W_C` be variables associated with tensors of dimensions $r \times c_{\text{in}} \times (p-1+k) \times (p-1+k)$, $1 \times r \times 1 \times 1$, and $r \times c_{\text{out}} \times p \times p$, respectively. Denote the standard 2D convolution and transposed 2D convolution operations with input `data`, filter tensor `weights`, `in_channels` input channels, `out_channels` output channels, kernel size `kernel_size`, and stride `stride` by `Conv2d` and `ConvTranspose2d`. Let `Multiply` be the broadcasted element-wise multiplication of `weights` with `data` and designate the function implementing the quantization scheme described in (4) by `Quantize`. Then, the forward pass (during training) through a compressed 2D convolution for an input tensor `in_data` of dimensions $b \times c_{\text{in}} \times W \times H$ is given by

```
W_B = Quantize(W_B)
W_C = Quantize(W_C)
conv_out = Conv2d(
                data=in_data,
                weights=W_B,
                in_channels=c_in,
                out_channels=r,
                kernel_size=p - 1 + k,
                stride=p,
                groups=g)
mul_out = Multiply(
                data=conv_out,
                weights=a_tilde)
out_data = ConvTranspose2d(
                data=mul_out,
                weights=W_C,
                in_channels=r,
                out_channels=c_out,
                kernel_size=p,
                stride=p)
```

At inference time, `Conv2d` and `ConvTranspose2d` can be replaced with specialized convolution operations exploiting the fact that `W_B` and `W_C` are ternary. To compute the backward pass, the backpropagation algorithm (Rumelhart et al., 1986) is applied to the sequence of operations in the forward pass, ignoring the `Quantize` operations. We found it beneficial to perform batch normalization (Ioffe & Szegedy, 2015) after the `Conv2d` operation.

D. Additional Tables

Table 5. Reduction (in %) in the number of additions obtained by our method for ResNet-20 on CIFAR-10.

red. in additions				
	r			
p	c_{out}	$\frac{3}{4}c_{\text{out}}$	$\frac{1}{2}c_{\text{out}}$	$\frac{1}{4}c_{\text{out}}$
1	-13.904	14.435	42.774	71.112
2	39.123	54.205	69.287	84.369
4	57.550	68.025	78.501	88.976

[3] Available at https://www.salesforce.com/products/einstein/ai-research/the-wikitext-dependency-language-modeling-dataset/.

Table 6. Testing PPL and reduction (in %) in the number of operations as well as model size for ST-LM compared to the full-precision model. We also report the reductions for the model compressed with TWN quantization (4).

	ST-LM, r							
	$8n_{\text{out}}$	$6n_{\text{out}}$	$4n_{\text{out}}$	$2n_{\text{out}}$	n_{out}	$\frac{1}{2}n_{\text{out}}$	$\frac{1}{4}n_{\text{out}}$	TWN
testing PPL	83.118	82.65	82.88	82.68	83.42	85.44	89.19	92.23
testing PPL (dist.)	77.29	76.74	76.49	77.69	79.11	81.14	86.29	-
multiplications	99.20	99.37	99.53	99.69	99.77	99.82	99.84	99.75
additions	-1682.23	-1238.24	-794.28	-350.31	-128.33	-17.34	38.21	-35.27
model size	-18.76	10.89	40.54	70.19	85.01	92.42	96.13	93.65

Table 7. Reduction in the number of additions and model size (in %) of ST-ResNet-18 compared to the full-precision model, for 224×224 images.

red. in additions

	r				
(p, g)	$6c_{\text{out}}$	$4c_{\text{out}}$	$2c_{\text{out}}$	c_{out}	$\frac{1}{2}c_{\text{out}}$
$(1, 1)$	-596.76	-364.56	-132.36	-16.32	41.73
$(2, 1)$	-288.57	-159.10	-29.63	35.05	67.41
$(1, 4)$	-181.51	-87.73	6.05	52.89	76.33

red. in model size

	r				
(p, g)	$6c_{\text{out}}$	$4c_{\text{out}}$	$2c_{\text{out}}$	c_{out}	$\frac{1}{2}c_{\text{out}}$
$(1, 1)$	53.87	67.76	81.64	92.16	95.63
$(2, 1)$	3.07	33.89	64.71	83.69	91.40
$(1, 4)$	80.31	85.38	90.45	96.56	97.83

Table 8. Top-1 and top-5 validation accuracy (in %) along with the reduction (in %) in the number of multiplications and model size for BWN (Rastegari et al., 2016), TWN (Li et al., 2016), TTQ (Zhu et al., 2016), ABC-Net-3 (Lin et al., 2017), and ST-ResNet-18-2-2-1 ($r = 2c_{\text{out}}$, $p = 2$, $g = 1$), compared to the full-precision model (for the full-precision model, the absolute quantities are given in parentheses).

	top-1	top-5	mul.	model size
BWN	60.8	83.0	93.25	92.33
TWN	61.8	84.2	99.73	93.39
TTQ	66.6	87.2	86.66	89.20
ABC-Net-3	66.2	86.7	99.42	49.39
ST-ResNet-18-2-2-1	67.1	87.5	99.77	125.89
full-precision	69.6	89.2	$(1.82 \cdot 10^9)$	(356.74 MB)