DeepShift: Towards Multiplication-Less Neural Networks

Mostafa Elhoushi[1], Zihao Chen[1,2], Farhan Shafiq[1], Ye Henry Tian[1], Joey Yiwei Li[1]
[1]Huawei Technologies, Markham, Canada
[2]Universitiy of Toronto, Toronto, Canada
m.elhoushi@ieee.org, {zihao.chen, farhan.shafiq}@huawei.com, {ye.henry.tian, li.joey922}@gmail.com

Abstract

The high computation, memory, and power budgets of inferring convolutional neural networks (CNNs) are major bottlenecks of model deployment to edge computing platforms, e.g., mobile devices and IoT. Moreover, training CNNs is time and energy-intensive even on high-grade servers. Convolution layers and fully connected layers, because of their intense use of multiplications, are the dominant contributor to this computation budget.

We propose to alleviate this problem by introducing two new operations: convolutional shifts and fully-connected shifts which replace multiplications with bitwise shift and sign flipping during both training and inference. During inference, both approaches require only 5 bits (or less) to represent the weights. This family of neural network architectures (that use convolutional shifts and fully connected shifts) is referred to as DeepShift models. We propose two methods to train DeepShift models: DeepShift-Q which trains regular weights constrained to powers of 2, and DeepShift-PS that trains the values of the shifts and sign flips directly.

Very close accuracy, and in some cases higher accuracy, to baselines are achieved. Converting pre-trained 32-bit floating-point baseline models of ResNet18, ResNet50, VGG16, and GoogleNet to DeepShift and training them for 15 to 30 epochs, resulted in Top-1/Top-5 accuracies higher than that of the original model. Last but not least, we implemented the convolutional shifts and fully connected shift GPU kernels and showed a reduction in latency time of 25% when inferring ResNet18 compared to unoptimized multiplication-based GPU kernels. The code can be found at https://github.com/mostafaelhoushi/DeepShift.

1. Introduction

Reducing the energy consumption, time latency, and memory requirements of deep neural networks for both inference and training are two requirements researchers are trying to achieve. For inference, deep neural networks are increasingly being targeted for mobile and IoT applications. Devices at the edge have a lower power and price budget as well as constrained memory size. Moreover, the amount of communication between different levels of memory in computing platforms also has a major role in the power requirements of a CNN. If communication between device and cloud becomes necessary (e.g. in case of model updates etc), model size could affect the connectivity costs. Therefore, for mobile / IoT inference applications, model optimization, size reduction, faster inference, and lower power consumption are key areas of research.

For training, the energy required to train an average deep learning model on GPU servers is equivalent to using fossil fuel releasing around 78,000 pounds of carbon, which is more than half of a car's output during its lifetime [34]. According to a report by OpenAI [1], the computation power required to train the largest neural network models doubles every 3.4 months, with an increase of 300,000× since 2012. Moreover, training on edge devices, e.g., in limited scenarios (including online learning, model adaptation, one/few-shot learning), is a growing field [7, 37]. Therefore, for both cloud/server and edge / mobile applications, reducing energy and time consumption of training is required.

Several approaches are being considered to address those needs and as such these efforts can be divided into a few categories. One approach is to design efficient models from the ground up resulting in novel network architectures that are more efficient to train and infer [17]. However, that proves to be a task requiring a lot of training resources to explore multiple variants of architectures to find the best fit. Another approach is to start with a big model initially and prune it [10, 8, 22, 26, 23]. Although pruning increases the efficiency for inference, it does not increase the efficiency of training as it requires training larger models from the start. Another large category of techniques to accelerate inference and/or training is quantization, which this paper falls under.

In such quantization techniques, the parameters of a model are converted from high-precision 32-bit floating-point representation to lower-precision smaller bit-width floating-point representations, such as 16-bits [28] or 8-bits

[35, 18], fixed-point representations [12], or other representations with bit-widths that can range from 5-bits to 2-bits [24, 40] and 1-bit [16]. Some quantization techniques start with a full-precision pre-trained model (hence accelerating inference only)[24] while others train from models from scratch and hence accelerating both inference and training [21]. Key attractions of these techniques are that they can be easily applied to various kinds of networks and they not only reduce the model size but also require less complex compute units on the underlying hardware. This results in a smaller model footprint, less working memory (and cache), faster computation on supporting platforms, and lower power consumption. Moreover, some quantization techniques replace multiplication with cheaper operations such as binary XNOR operations [24, 16, 27] or bitwise shifts [39]. Binary XNOR techniques have high accuracy on small datasets such as MNIST or CIFAR10, but they suffer high degradation on complex datasets such as Imagenet [31]. On the other hand, bit-wise shift techniques have proven to show a minimal drop in accuracy on Imagenet [39].

This paper presents an approach to reduce computation and power requirements of CNNs by replacing regular multiplication-based convolution and linear operations, also known as a fully-connected layer or matrix multiplication, with bitwise-shift-based convolution and linear operations respectively. Applying bitwise shift operation on an element is mathematically equivalent to multiplying it by a power of 2. We compare our work with one similar approach that replaces multiplications with bitwise shift and sign flips: Incremental Network Quantization (INQ) [39], and another approach that replaces multiplications with sign flips only: Accurate Binary Convolution (ABC). We propose two approaches: DeepShift-Q and DeepShift-PS. Both our DeepShift approaches are based on training the shift values (i.e. the powers of 2) and sign flips either indirectly or directly, while INQ is based on rounding 32-bit floating-point weights to powers of 2. To the best of our knowledge, this paper is the first to propose training the shift values directly, which is the main novelty of the paper. Moreover, INQ requires starting from a pre-trained model (hence can only accelerate inference) while both of our DeepShift proposed approaches can accelerate both inference and training.

This paper makes the following contributions:

- We introduce two different methods to train DeepShift networks, neural networks with powers of 2 weights: each method trains the powers of 2 at run-time, and when computing the parameters' gradients at train-time (see Section 3).

- We develop bitwise-shift-based matrix multiplication and convolution GPU kernels that are able to run the DeepShift ResNet18 architecture with 25% faster inference times than the baseline ResNet18 model with an unoptimized GPU kernel

2. Related Work

[39] proposed Incremental Network Quantization (INQ) to quantize pre-trained full-precision DNNs with weights constrained to zeros and powers of two. It replaces multiplications in convolutions with bitwise shifts and sign-flips but during inference only and not training. Hence, it cannot be used to reduce the energy consumption of on-device training (e.g., for transfer learning or zero-/one- shot training) nor on the cloud. INQ's approach involves iteratively partitioning the weights into two sets, one of which is quantized while the other is retrained to compensate for accuracy degradation. After each iteration, a bigger portion of the floating-point weights are converted to powers of 2, and the remaining portion is trained again. INQ introduces a new hyperparameter, the fraction of weights that are rounded to powers of 2 at each iteration. The authors of INQ listed the iterative steps they used for each CNN architecture, that was obtained or tuned empirically. Unlike INQ, our method does not introduce a new hyperparameter.

LogNN [30] is another method that converts weights as well as activations to powers of 2. However, it requires manual empirical tuning rather than training. The authors of LogNN did present an algorithm to train weights and activations in powers of 2, but it was only implemented on pre-trained models on CIFAR10 dataset. Also, LogNN does not convert the first layer, which may consume up to 20% of the FLOPs in CNN architectures, to bitwise shifts, while our approach converts all layers. Furthermore, the authors did not explain how sign-flip of weights should be trained. The authors of LogNN also presented a hardware implementation in [20]. LogQuant [4] and ShiftCNN [11] presented methods to convert the weights of a pre-trained model to powers of 2, such that no further training is needed and with a minimal decrease in accuracy. ShiftCNN replaces each multiplication with a group of 2 or 3 bitwise shifts, while our approach replaces each multiplication with a single bitwise shift.

To the best of our knowledge, DeepShift is the first method to train a model from scratch using powers of 2.

3. DeepShift Networks

As shown in Figure 1, the main concept of this paper is to replace multiplication with bitwise shift and sign flip. If the underlying binary representation of an input number, x is in integer or fixed-point format, a bit-wise shift of $\tilde{p}$ bits, where $\tilde{p}$ is an integer, to the left (or right) is mathematically equivalent to multiplying by a positive (or negative power)

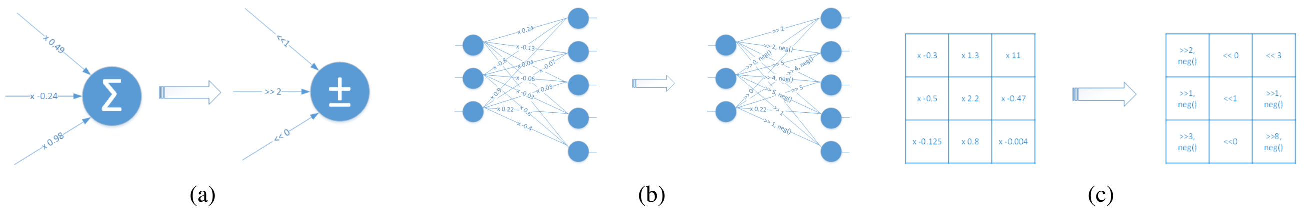


Figure 1: (a) Multiply and Accumulate (MAC), which is the main operation in most neural networks, is replaced in DeepShift with bitwise shift and addition/subtraction. (b) Weights of original linear operator vs. proposed shift linear operator. (c) Weights of original convolution operator vs. proposed shift convolution operator

of 2:

$$2^{\tilde{p}}x = \begin{cases} x << \tilde{p} \text{ if } \tilde{p} > 0 \\ x >> \tilde{p} \text{ if } \tilde{p} < 0 \\ x \text{ if } \tilde{p} = 0 \end{cases} \quad \text{such that } \tilde{p} \in \mathbb{Z} \tag{1}$$

For simplicity, we will use $<< \tilde{p}$ to implicitly denote a left shift if $\tilde{p}$ is positive and right shift if $\tilde{p}$ is negative.

Bitwise shift can only be equivalent to multiplying by a positive number, because $2^{\pm\tilde{p}} > 0$ for any real value of $\tilde{p}$. However, in neural networks, it is necessary for the training to have the equivalent of multiplying by negative numbers in its search space, especially in convolutional neural networks where filters with both positive and negative weights contribute to detecting edges. Therefore, we also need to use the sign flip (a.k.a negation) operation. We use a ternary sign operator, i.e., an operator that can either leave its operand unchanged, replace it with 0, or changes the sign of its operand. The sign flip operation can be represented mathematically as:

$$\text{flip}(x, \tilde{s}) = \begin{cases} -x \text{ if } \tilde{s} = -1 \\ 0 \text{ if } \tilde{s} = 0 \\ x \text{ if } \tilde{s} = +1 \end{cases} \quad \text{such that } \tilde{s} \in \{-1, 0, +1\} \tag{2}$$

Similar to bitwise shift, sign flip is a computationally cheap operation too as it involves returning the 2's complement of a number.

We introduce novel operators, LinearShift and ConvShift, that, in the forward pass, replace multiplication with bitwise shift and sign flip. Hence, the weight matrix, W, whether it is used for linear operation (i.e., $Y = WX + b$) or convolution operation (i.e., $Y = W \circledast X + b$), will be replaced by elementwise product of the shift matrix $\tilde{P}$ and sign matrix $\tilde{S}$.

$$W \rightarrow \text{flip}(2^{\tilde{P}}, \tilde{S}) \tag{3}$$

In the following sub-sections, we shall present two different methods to train the LinearShift and ConvShift operators of our DeepShift models: DeepShift-Q and DeepShift-PS. Both approaches use Equation 3 for the forward pass but differ in the backward pass and differ in how $\tilde{P}$ and $\tilde{S}$ are derived. Both approaches are summarized in Figure 2.

3.1. DeepShift-Q

In the DeepShift-Q approach, training is similar to the regular training approach of linear or convolution operations with weight W, but in the forward pass and backward pass the weight matrix is quantized to $\tilde{W}_q$ by rounding it to the nearest power of 2:

$$\begin{aligned} S &= \text{sign}(W) \\ \tilde{S} &= S \end{aligned} \tag{4}$$

$$\begin{aligned} P &= \log_2(\text{abs}(W)) \\ \tilde{P} &= \text{round}(P) \end{aligned} \tag{5}$$

$$\tilde{W}_q = \text{flip}(2^{\tilde{P}}, \tilde{S}) \tag{6}$$

Hence, the forward pass for our LinearShift operator will be $Y = \tilde{W}_q X = \text{flip}(2^{\tilde{P}}, \tilde{S})X$ and for our ConvShift operator will be $Y = \tilde{W}_q \circledast X + b = \text{flip}(2^{\tilde{P}}, \tilde{S}) \circledast X + b$.

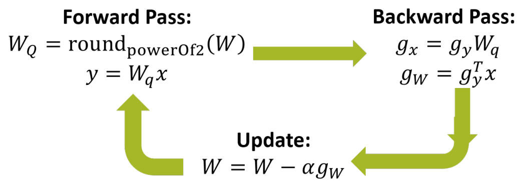
The gradients of the backward pass can be expressed as:

$$\begin{aligned} \frac{\partial C}{\partial X} &= \frac{\partial C}{\partial Y}\frac{\partial Y}{\partial X} = \frac{\partial C}{\partial Y}\tilde{W}_q \\ \frac{\partial C}{\partial W} &= \frac{\partial C}{\partial Y}\frac{\partial Y}{\partial \tilde{W}_q}\frac{\partial \tilde{W}_q}{\partial W} = \frac{\partial C}{\partial Y}X\frac{\partial \tilde{W}_q}{\partial W} \\ \frac{\partial C}{\partial b} &= \frac{\partial C}{\partial Y} \end{aligned} \tag{7}$$

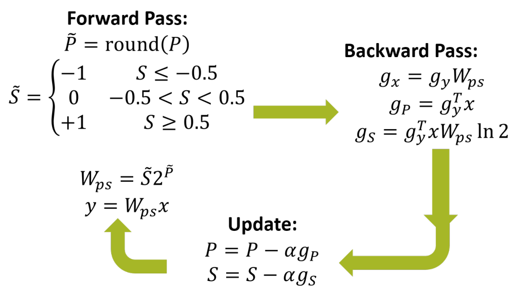
where $\frac{\partial C}{\partial Y}$ is the output gradient to the operator (derivative of the model loss (a.k.a cost function), C, with respect to the operator output), $\frac{\partial C}{\partial X}$ is the input gradient to the operator (derivative of the model loss with respect to the operator input)[1], and $\frac{\partial C}{\partial W}$ is the derivative of the model loss with respect to the operator weights [2].

[1]We have simplified the expression of $\frac{\partial C}{\partial X}$ in Equation 7. To be exact, it is equivalent to $\frac{\partial C}{\partial Y}\tilde{W}_q^T$ for LinearShift and $\tilde{W}_q^{rot180^\circ} \circledast \frac{\partial C}{\partial Y}$ for ConvShift.

[2]For LinearShift, $\frac{\partial C}{\partial W} = X^T\frac{\partial C}{\partial Y}$ and for ConvShift $\frac{\partial C}{\partial W} = X \circledast \frac{\partial C}{\partial Y}$



(a) DeepShift-Q


Forward Pass:
$\tilde{P} = \text{round}(P)$
$\tilde{S} = \begin{cases} -1 & S \le -0.5 \\ 0 & -0.5 < S < 0.5 \\ +1 & S \ge 0.5 \end{cases}$
$W_{ps} = \tilde{S}2^{\tilde{P}}$
$y = W_{ps} x$
Backward Pass:
$g_x = g_y W_{ps}$
$g_P = g_y^T x$
$g_S = g_y^T x W_{ps} \ln 2$
Update:
$P = P - \alpha g_P$
$S = S - \alpha g_S$

(b) DeepShift-PS

Figure 2: We present 2 algorithms to train bitwise shift & negation parameters of a model: (a) DeepShift-Q, (b) DeepShift-PS.

Since W_q is a rounded value of W, the differentiation $\frac{\partial \tilde{W}_q}{\partial W}$ is zero at all points except at the points where W is a power of 2. This will result in W not being updated during backpropagation, and hence no learning or updating of W. We solve this issue by using a straight-through estimator (STE) [38]:

$$\frac{\partial \tilde{W}_q}{\partial W} \approxeq 1 \tag{8}$$

Hence, $\frac{\partial C}{\partial W}$ is set to:

$$\frac{\partial C}{\partial W} \approxeq X \frac{\partial Y}{\partial \tilde{W}_q} \tag{9}$$

Figure 2a summarizes the forward and backward passes of DeepShift-Q.

3.2. DeepShift-PS

DeepShift-PS on the other hand directly uses the shift, P, and sign flip, S, as the trainable parameters:

$$\begin{aligned} \tilde{P} &= \text{round}(P) \\ \tilde{S} &= \text{sign}(\text{round}(S)) \\ \tilde{W}_{ps} &= \text{flip}(2^{\tilde{P}}, \tilde{S}) \end{aligned} \tag{10}$$

Note the definition of sign flip operation, $\tilde{S}$, implies that it is a ternary value rather than a binary parameter, as each element in the matrix can take one of 3 values:

$$\tilde{s} = \begin{cases} -1 \text{ if } s \le -0.5 \\ 0 \text{ if } -0.5 < s < 0.5 \\ +1 \text{ if } s \ge 0.5 \end{cases} \tag{11}$$

where s is an element of matrix S and $\tilde{s}$ is an element of $\tilde{S}$.

The gradients of the backward pass can be expressed as:

$$\begin{aligned} \frac{\partial C}{\partial X} &= \frac{\partial C}{\partial Y}\frac{\partial Y}{\partial X} \\ &= \frac{\partial C}{\partial Y} W_{ps}^T \\ \frac{\partial C}{\partial P} &= \frac{\partial C}{\partial Y}\frac{\partial Y}{\partial \tilde{W}_{ps}}\frac{\partial \tilde{W}_{ps}}{\partial \tilde{P}}\frac{\partial \tilde{P}}{\partial P} \\ \frac{\partial C}{\partial S} &= \frac{\partial C}{\partial Y}\frac{\partial Y}{\partial \tilde{W}_{ps}}\frac{\partial \tilde{W}_{ps}}{\partial \tilde{S}}\frac{\partial \tilde{S}}{\partial S} \\ \frac{\partial C}{\partial b} &= \frac{\partial L}{\partial Y} \end{aligned} \tag{12}$$

We use the straight through estimators to express the derivatives of the round function and the sign function: $\partial \text{round}(x) \approxeq \partial x$, $\partial \text{sign}(x) \approxeq \partial x$. Hence, $\frac{\partial \text{round}(x)}{\partial x} \approxeq 1$, $\frac{\partial \text{sign}(x)}{\partial x} \approxeq 1$. However, for the flip function we use a modified straight through estimator: $\frac{\partial \text{flip}(x,s)}{\partial x} \approxeq \text{flip}(x, s)$, $\frac{\partial \text{flip}(x,s)}{\partial s} \approxeq 1$.

Based on that, we set $\frac{\partial \tilde{P}}{\partial P} \approxeq 1$ and $\frac{\partial \tilde{S}}{\partial S} \approxeq 1$. Hence, $\frac{\partial \tilde{W}_{ps}}{\partial \tilde{S}}$ is evaluated as:

$$\frac{\partial \tilde{W}_{ps}}{\partial \tilde{S}} = \frac{\partial \text{flip}(2^{\tilde{P}}, \tilde{S})}{\partial \tilde{S}} \approxeq 1 \tag{13}$$

and, $\frac{\partial \tilde{W}_{ps}}{\partial \tilde{P}}$ is evaluated as:

$$\begin{aligned} \frac{\partial \tilde{W}_{ps}}{\partial \tilde{P}} &= \frac{\partial \text{flip}(2^{\tilde{P}}, \tilde{S})}{\partial \tilde{P}} = \frac{\partial \text{flip}(2^{\tilde{P}}, \tilde{S})}{\partial (2^{\tilde{P}})}\frac{\partial (2^{\tilde{P}})}{\partial \tilde{P}} \\ &\approxeq \text{flip}(2^{\tilde{P}}, \tilde{S}) 2^{\tilde{P}} \ln 2 = \tilde{W}_{ps} \ln 2 \end{aligned} \tag{14}$$

Hence, the gradients of the weights with respect to loss are set to:

$$\begin{aligned} \frac{\partial C}{\partial P} &\approxeq \frac{\partial C}{\partial Y}\frac{\partial Y}{\partial \tilde{W}_{ps}} \tilde{W}_{ps} \ln 2 \\ \frac{\partial C}{\partial S} &\approxeq \frac{\partial C}{\partial Y}\frac{\partial Y}{\partial \tilde{W}_{ps}} \end{aligned} \tag{15}$$

Figure 2b summarizes the forward and backward passes of DeepShift-PS.

4. Implementation

We implement the forward and backward passes of our two custom ops, LinearShift and ConvShift, using PyTorch. Similar to [16], we emulate the precision of an actual bit-wise shift hardware implementation by rounding the input and bias of both ops to 32-bit fixed-point format precision (16-bit for integer part and 16-bit for fraction part) before applying the forward pass.

To initialize the weights, we use Kaiming initialization [14] for W_q in DeepShift-Q, and uniform random distribution to initialize P and S in DeepShift-PS.

In DeepShift-PS, L2 normalization has been slightly modified to occur on W_{PS} rather than on P on S, i.e. the regularization term added is $\sum W_{PS}^2 = \sum 2^P S^2$ rather than $\sum P^2 + \sum S^2$. This modification of L2 normalization was necessary to avoid gradient descent from pushing shift parameters, P, from values whose squares are large, like -14 or -8, but are actually equivalent to multiplication with small weights, 2^{-8} and 2^{-14}, to values whose squares are small like 0 or 1, but are actually equivalent to multiplying with relatively large weight values, 1.0 or 0.5, respectively.

Since activations are represented as 32-bit fixed point activations, the theoretical range of shift values that we need to support are -31 to +31, and this requires $\log_2(31 - (-31) + 1) = \log_2(63) \approxeq 6$ bits to represent shift. However, we noticed that shift values that result from training are rarely positive (i.e., most of the time $\text{abs}(W_Q) < 1$, hence $P < 0$ as $P = \log_2(\text{abs}(\text{round}(W_Q)))$ [3]). Further testing that we performed showed that high accuracy can be retained if we constrain shift values to be between 0 and -14 or 0 and -15. Therefore, we only need 4 bits to represent shift. Adding the bit required for the sign flip operation, we need 5 bits in total. During training, shift values that fell outside the supported shift ranges were clipped. In some of the results that we will explain in the upcoming sections, we have also experimented with fewer bits to represent weight, i.e., constraint shift values to shorter ranges.

5. Benchmark Results

We have tested the training and inference results on 2 datasets: CIFAR10 [19], and Imagenet [6]. For each dataset, we have tested a group of architectures. We have explored various evaluations for the models:

1. **Original Version**: evaluating the original architecture with standard convolution and linear operators,

2. **DeepShift Versions**:

   (a) **Train from Scratch**: start with randomly initialized weights, convert all the convolution and linear operators to their shift counterparts, and train from scratch using either training method Deepshift-Q or DeepShift-PS,

   (b) **Train from Pre-trained Baseline**: start with baseline model that is pretrained with 32-bits floating points weights, convert all the convolution and linear operators to their shift counterparts, convert the weights (using Equations 4 and 5) and train using either training method Deepshift-Q or DeepShift-PS.

[3]This expression holds for both W_Q and W_{PS}

We have noticed - in terms of validation accuracy results - that the best optimizer for DeepShift-PS is Rectified Adam (RAdam) [25] while the best optimizer for DeepShift-Q and regular 32-bit floating-point training is stochastic gradient descent (SGD).

5.1. CIFAR10 Data Set

ResNet18 [15] and MobileNetv2 [32] models were trained from scratch on CIFAR10 using both DeepShift methods. The models were trained with a momentum of 0.9 and weight decay of 1×10^{-4}. The loss criterion used was categorical cross-entropy. The learning rate used to train was 0.1 that decays by a factor of 0.1 at epochs 80, 120, and 160, the number of epochs for training was 200, and the batch size was 128.

For ResNet18, we have added an additional test of training the baseline (multiplication-based) 32-bit floating-point weights using RAdam, to compare it with DeepShift-PS that uses RAdam. This test was made to ensure that the comparisons are fair and that the use of RAdam as an optimizer is not giving an unfair advantage to DeepShift-PS. The best accuracy reached by the baseline ResNet18 model with SGD was 94.45% and with RAdam was 94.06%.

For training starting with a pre-trained model, the learning rate was 1×10^{-3}, decaying with a rate of 0.1 every 5 epochs, and the number of epochs was 15.

In order to compare with other multiplication-less models, we also trained VGG19-Small [33] from scratch to compare with AdderNet [5], that replaces multiplications with additions, and ShiftAddNet [13] that combined our DeepShift-PS approach with AdderNet. In Table 1 we mark AdderNet and ShiftAddNet with an asterisk to indicate that the accuracies shown are based on the results from the authors' paper rather than being reproduced by us. In order to make a fair comparison, the results shown are based on using 32-bit fixed-point activations as reproduced by [13].

5.1.1 Lower Bit Widths

Table 2 shows the results for using fewer bits to represent weights or activations. For 16-bit activations, we used a

Table 1: CIFAR10 Accuracy Results

Model	Method	Train from Scratch	Train from Pre-trained
ResNet18	Original	94.45%	-
	DeepShift-Q	94.42%	94.25%
	DeepShift-PS	93.20%	94.12%
MobileNet	Original	93.57%	-
	DeepShift-Q	93.63%	93.04%
	DeepShift-PS	92.64%	92.78%
VGG19-Small	Original	92%	-
	DeepShift-PS	91.57%	N/A%
	AdderNet* [5]	80%	N/A%
	ShiftAddNet* [13]	90%	N/A%

Table 2: ResNet18 CIFAR10 Accuracy Results for Lower Bitwidths

Method	W	A	Train from Scratch	Train from Pre-trained
Original	32	32	94.45%	-
DeepShift-PS	5	32	93.20%	94.12%
	5	16	85.54%	94.15%
	5	8	86.28%	93.96%
	4	32	94.12%	94.13%
	3	32	92.85%	91.16%
	2	32	92.80%	90.68%
DeepShift-Q	5	32	94.60%	94.43%
	5	16	94.06%	93.70%
	5	8	94.14%	93.65%
	4	32	94.11%	94.38%
	3	32	88.28%	92.04%
	2	32	69.97%	89.96%

fixed-point representation with 3-bits to represent the integer part, and 13-bits to represent the fraction part. Similarly for 8-bit activations, we used 3-bits to represent the integer part, and 13-bits to represent the fraction part. For training 8-bit or 16-bit activations from scratch, we changed the initial learning rate to 0.001.

We can see that for 3-bits and 2-bits weights, training from scratch obtained higher accuracy than training from a pre-trained baseline. This is because, converting the pre-trained weights to powers of 2 constrained to a short range, caused many weight values to be clipped to the same value. Hence the distribution of weights of a model that is randomly initialized, i.e., training from scratch, is better than weights of a pre-trained model clipped to small values [29].

On the other hand, DeepShift-Q outperforms DeepShift-PS when training from scratch using 16-bit or 8-bit activations, while both are close to baseline accuracies when trained from pre-trained weights.

5.2. Imagenet Data Set

We categorize methods in literature that accelerate neural networks by replacing multiplications with cheaper operations or by quantization, into methods:

- that train from pre-trained baseline: INQ [39], ABC [24], and
- that train from scratch, BWN [31], TWN [21].

In each of the following two sub-sections, we compare DeepShift to each category. Results for various CNN architectures for both categories are shown in Tables 3 and 4. Since different papers report different accuracies for full-precision baselines, we either attempt to reproduce the results of a method or report the difference between the accuracy of the method and its corresponding baseline (method names marked with asterisks).

5.2.1 Training from Pre-Trained Baseline

Table 3 compares DeepShift trained from a pre-trained baseline to INQ and ABC methods. Column "W" shows the number of bits to represent weights in each method. The DeepShift models were trained with momentum of 0.9, weight decay of 1×10^{-4}, the loss criterion categorical cross-entropy, the initial learning rate was 0.001 that decays by a factor of 0.1 every 5 epochs.

INQ [39] (Incremental Network Quantization) replaces multiplications in convolutions with bitwise shifts and sign-flips, but during inference only and not training. It also introduces a hyperparameter, a list of portions per iteration to round the weights to power of 2. This list of portions is obtained in an empirical manner for each architecture. DeepShift on the other hand does not introduce any hyperparameter. INQ was originally developed using Caffe, and the pre-trained baselines reported were significantly lower than what we have using PyTorch. Hence, to start with the same baseline, we used the PyTorch implementation of INQ [3] to report the accuracies in Table 3. In terms of accuracies, both INQ and DeepShift exceed or almost meet the baseline accuracies of most architectures. For ResNet18, we also compare using 4-bits to represent weights.

ABC [24] (Accurate Binary Convolution) neural networks replace multiplication with XNOR (i.e., sign flip) operation. Comparing with the same number of bits, 5, for weight representation, DeepShift obtains higher accuracy than ABC.

Table 3: Imagenet Accuracy Results: Training from Pre-trained Baseline. “W” refers to number of bits to represent weights.

Model	Method	W	Top-1		Top-5	
			Accuracy	Delta	Accuracy	Delta
ResNet18	Original	32	69.76%	-	89.08%	-
	DeepShift-Q [Ours]	5	69.56%	-0.20%	89.17%	+0.09%
	DeepShift-PS [Ours]	5	69.27%	-0.49%	89.00%	-0.08%
	INQ [39]	5	69.36%	-0.40%	89.09%	+0.01%
	ABC* [24]	5	68.30%	-1.00%	87.90%	-1.30%
	DeepShift-Q [Ours]	4	69.56%	-0.20%	89.14%	+0.06%
	DeepShift-PS [Ours]	4	69.02%	-0.74%	88.73%	-0.35%
	INQ* [39]	4	68.88%	+0.63%	89.01%	+0.32%
ResNet50	Original	32	76.13%	-	92.86%	-
	DeepShift-Q [Ours]	5	76.33%	+0.22%	93.05%	+0.19%
	DeepShift-PS [Ours]	5	75.93%	-0.20%	92.90%	+0.04%
	INQ [39]	5	75.01%	-1.12%	92.45%	-0.41%
GoogleNet	Original	32	69.78%	-	89.53%	-
	DeepShift-Q [Ours]	5	70.73%	+0.95%	90.17%	+0.64%
	DeepShift-PS [Ours]	5	69.87%	+0.09%	89.62%	+0.09%
	INQ [39]	5	71.25%	+1.47%	90.33%	+0.70%
VGG16	Original	32	71.59%	-	90.38%	-
	DeepShift-Q [Ours]	5	71.56%	-0.03%	90.48%	+0.10%
	DeepShift-PS [Ours]	5	71.39%	-0.20%	90.33%	-0.05%
	INQ [39]	5	71.32%	-0.27%	90.29%	-0.09%
AlexNet	Original	32	56.52%	-	79.07%	-
	DeepShift-Q [Ours]	5	55.81%	-0.71%	78.79%	-0.28%
	DeepShift-PS [Ours]	5	55.90%	-0.62%	78.73%	-0.34%
	INQ [39]	5	56.13%	-0.39%	78.83%	-0.24%

5.2.2 Training from Scratch

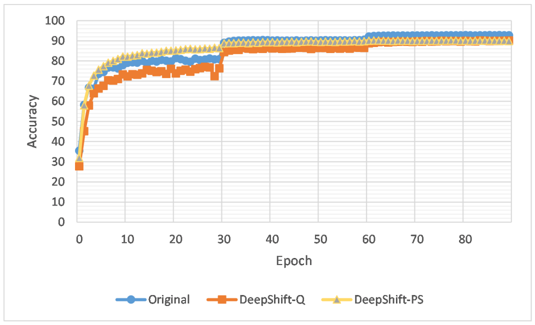
Models were trained from scratch for 90 epochs, the learning rate used to train from scratch was 0.01 that decays by a factor of 0.1 every 30 epochs. Training plots are shown in Figure 3 and results are shown in Table 4.

For ResNet18, we compare the performance of DeepShift that replaces multiplications with bitwise shifts and sign flips, with BWN (Binary Weight Networks) & TWN (Ternary Weight Networks) that replace multiplications with sign flips only.

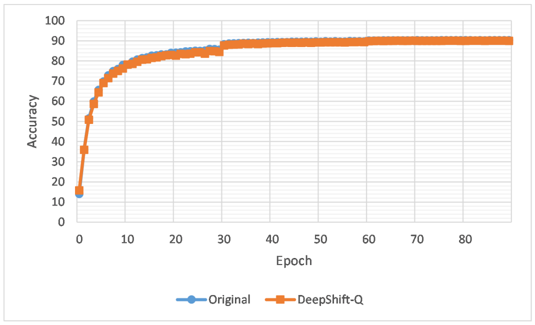
6. Efficiency Analysis

Multiplication in integer or fixed-point format consists of multiple bitwise shifts, ANDs, and additions. Floating point (FP) multiplication consists of more steps such as adding the exponents, & normalizing. In terms of CPU cycles, taking a 32-bit Intel Atom instruction processor as an example, integer and FP multiplication instruction take 5 to 6 cycles, while bit-wise shift takes only 1 cycle ([9]). ([2]) shows for 16-bit architectures, average power, and area of multipliers are at least 9.7×, and 1.45×, respectively, of bitwise shifter: 22.05 nW to 54.83 nW vs 1.32 nW to 2.27 nW, and 8.7K to 29K transistor count vs 2K to 6K only. A common optimization in C++ compilers is to detect an integer multiplication with a power of 2 and replace it with a bitwise shift. Our contribution is enabling bitwise shifts in training neural networks while achieving good accuracy. We emphasize that we replace each multiplication with a single bitwise shift.

DeepShift also achieves model storage compression by representing the weights with fewer bits. This reduces the memory footprint of the model and increases energy savings for transferring weights between different layers of memory of a GPU or CPU ([36]). That is needed in mobile and IoT applications, where memory footprint is a primary constraint. Energy savings is crucial for applications like drones that fly on batteries, whether the computing platform is a low-end CPU, GPU, or FPGA. Also, bitwise shift op-



(a) ResNet50 Top-5 Accuracy



(b) VGG16 Top-5 Accuracy

Figure 3: Imagenet training from scratch

Table 4: Imagenet Accuracy Results: Training from Scratch. "Compute" refers to the type of arithmetic instruction the convolution and fully-connected layers are based on.

Model	Method	Compute	Top-1		Top-5	
			Accuracy	Delta	Accuracy	Delta
ResNet18	Original	Multiply	69.76%	-	89.08%	-
	DeepShift-Q [Ours]	Shift & Sign Flip	65.32%	-4.44%	86.30%	-2.78%
	DeepShift-PS [Ours]	Shift & Sign Flip	65.34%	-4.42%	86.05%	-3.03%
	BWN* [31]	Sign Flip	60.80%	-9.30%	83.00%	-6.20%
	TWN* [21]	Sign Flip	61.80%	-3.60%	84.20%	-2.56%
ResNet50	Original	Multiply	76.13%	-	92.86%	-
	DeepShift-Q [Ours]	Shift & Sign Flip	70.73%	-5.56%	90.16%	-2.70%
	DeepShift-PS [Ours]	Shift & Sign Flip	71.90%	-4.23%	90.16%	-2.70%
VGG16	Original	Multiply	71.59%	-	90.38%	-
	DeepShift-Q [Ours]	Shift & Sign Flip	70.87%	-0.71%	90.09%	-0.29%

erators are a good candidate for in-memory computing research [36], as they're cheaper in size and energy to place in or near memory than multiplication.

7. Future Work

Evaluating DeepShift on custom FPGA design, or extending CPU and GPU instruction sets with dedicated bitwise shift and negate instructions that operate on a vector and/or compressed 5-bit representation is one direction to explore. Also, we suggest training models from scratch with both activations and weights represented as powers of 2, similar to [30] that did that for inference only. Moreover, to speed up training, we suggest to try gradients represented as 8-bit or 16-bit fixed points.

Acknowledgments

We thank Ahmed Eltantawy for providing us valuable feedback to implement the CUDA kernels, and Sara Elkerdawy for providing valuable insights during our discussions.

References

[1] Open AI. Ai and compute. https://openai.com/blog/ai-and-compute/. Accessed: 2020-02-07. 1

[2] Abhijit Rameshwar Asati. *A Comparative Study of High Performance CMOS Multipliers, Barrel Shifters and Modeling of NBTI Degradation in Nanometer Scale Digital VLSI Circuits*. PhD thesis, Birla Institute of Technology and Science, Pilani, Rajasthan, India, 2009. 7

[3] Maxim Bonnaerens. A pytorch implementation of "incremental network quantization: Towards lossless cnns with low-precision weights". Technical report, 2019. 6

[4] Jingyong Cai, Masashi Takemoto, and Hironori Nakajo. A deep look into logarithmic quantization of model parameters in neural networks. In *Proceedings of the 10th International Conference on Advances in Information Technology*, IAIT 2018, pages 6:1–6:8, New York, NY, USA, 2018. ACM. 2

[5] Hanting Chen, Yunhe Wang, Chunjing Xu, Boxin Shi, Chao Xu, Qi Tian, and Chang Xu. Addernet: Do we really need multiplications in deep learning? *CVPR*, 2020. 5, 6

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In

CVPR09, 2009. 5

[7] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. On-device machine learning: An algorithms and learning theory perspective, 2019. 1

[8] Sara Elkerdawy, Hong Zhang, and Nilanjan Ray. Lightweight monocular depth estimation model by joint end-to-end filter pruning. *2019 IEEE International Conference on Image Processing (ICIP)*, Sep 2019. 1

[9] Agner Fog. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, amd and via cpus. Technical report, 2018. 7

[10] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2018. 1

[11] Denis Gudovskiy and Luca Rigazio. ShiftCNN: Generalized low-precision architecture for inference of convolutional neural networks. *arXiv preprint arXiv:1706.02393*, 2017. 2

[12] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1737–1746, Lille, France, 07–09 Jul 2015. PMLR. 2

[13] Yongan Zhang Chaojian Li Sicheng Li Zihao Liu Zhangyang Wang Yingyan Lin Haoran You, Xiaohan Chen. Shiftaddnet: A hardware-inspired deep network. In *Thirty-fourth Conference on Neural Information Processing Systems*, 2020. 5, 6

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. 5

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 5

[16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc., 2016. 2, 5

[17] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. 1

[18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[19] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, Department of Computer Science, 2009. 5

[20] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong. Lognet: Energy-efficient neural networks using logarithmic computation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5900–5904, March 2017. 2

[21] Fengfu Li and Bin Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016. 2, 6, 8

[22] Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression, 2020. 1

[23] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. Dhp: Differentiable meta pruning via hypernetworks. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 608–624, Cham, 2020. Springer International Publishing. 1

[24] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 345–353. Curran Associates, Inc., 2017. 2, 6, 7

[25] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019. 5

[26] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning, 2019. 1

[27] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 2

[28] Paulius Micikevicius. Mixed-precision training of deep neural networks. Technical report, 2017. 1

[29] Dmytro Mishkin and Juan E. Sala Matas. All you need is a good init. *ICLR 2015*, abs/1511.06422, 2015. 6

[30] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *ArXiv*, abs/1603.01025, 2016. 2, 8

[31] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016. 2, 6, 8

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, June 2018. 5

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 5

[34] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *ACL*, 2019. 1

[35] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi (Viji) Srinivasan,

Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alche-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 4900–4909. Curran Associates, Inc., 2019. 2

[36] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017. 7, 8

[37] Mineto Tsukada, Masaaki Kondo, and Hiroki Matsutani. A neural network-based on-device learning anomaly detector for edge devices. *IEEE Transactions on Computers*, page 1–1, 2020. 1

[38] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley J. Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. In *International Conference on Learning Representations*, 2019. 4

[39] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *CoRR*, abs/1702.03044, 2017. 2, 6, 7

[40] 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