AdderNet: Do We Really Need Multiplications in Deep Learning?

Hanting Chen[1]*, Yunhe Wang[2], Chunjing Xu[2], Boxin Shi[3,4], Chao Xu[1], Qi Tian[2], Chang Xu[5]
[1] Key Lab of Machine Perception (MOE), CMIC, School of EECS, Peking University, China
[2] Huawei Noah's Ark Lab, [3] National Engineering Laboratory for Video Technology, Peking University,
[4] Peng Cheng Laboratory, [5] School of Computer Science, Faculty of Engineering, The University of Sydney, Australia
{htchen, shiboxin}@pku.edu.cn, xuchao@cis.pku.edu.cn, c.xu@sydney.edu.au
{yunhe.wang, xuchunjing, tian.qi1}@huawei.com

Abstract

Compared with cheap addition operation, multiplication operation is of much higher computation complexity. The widely-used convolutions in deep neural networks are exactly cross-correlation to measure the similarity between input feature and convolution filters, which involves massive multiplications between float values. In this paper, we present adder networks (AdderNets) to trade these massive multiplications in deep neural networks, especially convolutional neural networks (CNNs), for much cheaper additions to reduce computation costs. In AdderNets, we take the ℓ_1-norm distance between filters and input feature as the output response. The influence of this new similarity measure on the optimization of neural network have been thoroughly analyzed. To achieve a better performance, we develop a special back-propagation approach for AdderNets by investigating the full-precision gradient. We then propose an adaptive learning rate strategy to enhance the training procedure of AdderNets according to the magnitude of each neuron's gradient. As a result, the proposed AdderNets can achieve 74.9% Top-1 accuracy 91.7% Top-5 accuracy using ResNet-50 on the ImageNet dataset without any multiplication in convolution layer.

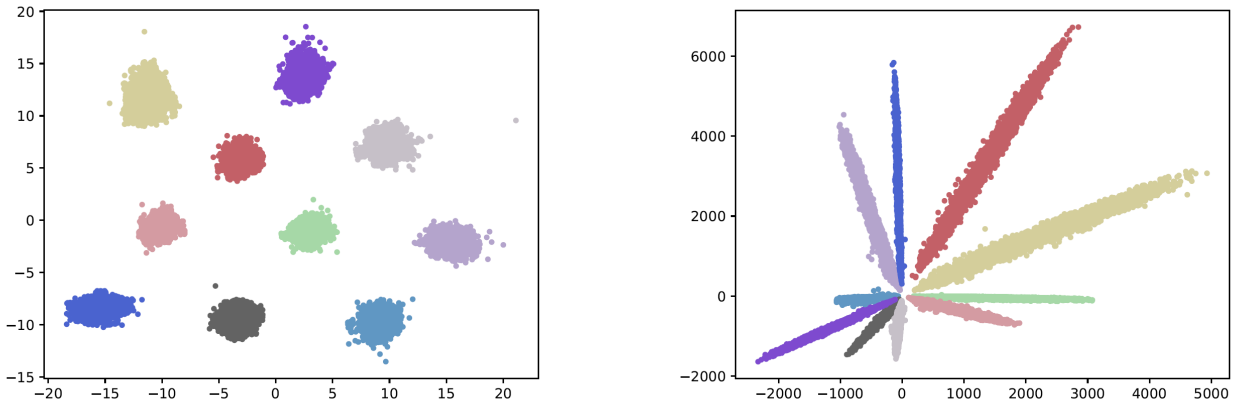
1. Introduction

Given the advent of Graphics Processing Units (GPUs), deep convolutional neural networks (CNNs) with billions of floating number multiplications could receive speed-ups and make important strides in a large variety of computer vision tasks, *e.g.* image classification [26, 17], object detection [23], segmentation [19], and human face verification [30]. However, the high-power consumption of these high-end GPU cards (*e.g.* 250W+ for GeForce RTX 2080 Ti) has blocked modern deep learning systems from being deployed on mobile devices, *e.g.* smart phone, camera, and watch. Existing GPU cards are far from svelte and cannot be easily mounted on mobile devices. Though the GPU itself only takes up a small part of the card, we need many other hardware for supports, *e.g.* memory chips, power circuitry, voltage regulators and other controller chips. It is therefore necessary to study efficient deep neural networks that can run with affordable computation resources on mobile devices.

Addition, subtraction, multiplication and division are the four most basic operations in mathematics. It is widely known that multiplication is slower than addition, but most of the computations in deep neural networks are multiplications between float-valued weights and float-valued activations during the forward inference. There are thus many papers on how to trade multiplications for additions, to speed up deep learning. The seminal work [5] proposed BinaryConnect to force the network weights to be binary (*e.g.*-1 or 1), so that many multiply-accumulate operations can be replaced by simple accumulations. After that, Hubara *et al.* [15] proposed BNNs, which binarized not only weights but also activations in convolutional neural networks at runtime. Moreover, Rastegari *et al.* [22] introduced scale factors to approximate convolutions using binary operations and outperform [15, 22] by large margins. Zhou *et al.* [38] utilized low bit-width gradient to accelerate the training of binarized networks. Cai *et al.* [4] proposed an half-wave Gaussian quantizer for forward approximation, which achieved much closer performance to full precision networks.

Though binarizing filters of deep neural networks significantly reduces the computation cost, the original recognition accuracy often cannot be preserved. In addition, the training procedure of binary networks is not stable and usually requests a slower convergence speed with a small learning rate. Convolutions in classical CNNs are actually cross-correlation to measure the similarity of two inputs. Researchers and developers are used to taking convolution

*Work was done while visiting Huawei Noah's Ark Lab



(a) Visualization of features in AdderNets

(b) Visualization of features in CNNs

Figure 1. Visualization of features in AdderNets and CNNs. Features of CNNs in different classes are divided by their angles. In contrast, features of AdderNets tend to be clustered towards different class centers, since AdderNets use the ℓ_1-norm to distinguish different classes. The visualization results suggest that ℓ_1-distance can served as a similarity measure the distance between the filter and the input feature in deep neural networks

as a default operation to extract features from visual data, and introduce various methods to accelerate the convolution, even if there is a risk of sacrificing network capability. But there is hardly no attempt to replace convolution with another more efficient similarity measure that is better to only involve additions. In fact, additions are of much lower computational complexities than multiplications. Thus, we are motivated to investigate the feasibility of replacing multiplications by additions in convolutional neural networks.

In this paper, we propose adder networks that maximize the use of addition while abandoning convolution operations. Given a series of small template as "filters in the neural network, ℓ_1-distance could be an efficient measure to summarize absolute differences between the input signal and the template as shown in Figure 1. Since subtraction can be easily implemented through addition by using its complement code, ℓ_1-distance could be a hardware-friendly measure that only has additions, and naturally becomes an efficient alternative of the convolution to construct neural networks. An improved back-propagation scheme with regularized gradients is designed to ensure sufficient updates of the templates and a better network convergence. The proposed AdderNets are deployed on several benchmarks, and experimental results demonstrate AdderNets advantages in accelerating inference of deep neural networks while preserving comparable recognition accuracy to conventional CNNs.

This paper is organized as follows. Section 2 investigates related works on network compression. Section 3 proposes Adder Networks which replace the multiplication in the conventional convolution filters with addition. Section 4 evaluates the proposed AdderNets on various benchmark datasets and models and Section 5 concludes this paper.

2. Related works

To reduce the computational complexity of convolutional neural networks, a number of works have been proposed for eliminating useless calculations.

2.1. Network Pruning

Pruning based methods aims to remove redundant weights to compress and accelerate the original network. Denton *et al*. [6] decomposed weight matrices of fully-connected layers into simple calculations by exploiting singular value decomposition (SVD). Han *et al*. [8] proposed discarding subtle weights in pre-trained deep networks to omit their original calculations without affecting the performance. Wang *et al*. [29] further converted convolution filters into the DCT frequency domain and eliminated more floating number multiplications. In addition, Hu *et al*. [13] discarded redundant filters with less impacts to directly reduce the computations brought by these filters. Luo *et al*. [21] discarded redundant filters according to the reconstruction error. He *et al*. [10] utilized a LASSO regression to select important channels by solving least square reconstruction. Zhuang *et al*. [39] introduce additional losses to consider the discriminative power of channels and selected the most discriminative channels for the portable network.

2.2. Efficient Blocks Design

Instead of directly reducing the computational complexity of a pre-trained heavy neural network, lot of works focused on designing novel blocks or operations to replace the conventional convolution filters. Iandola *et al*. [16] introduced a bottleneck architecture to largely decrease the computation cost of CNNs. Howard *et al*. [12] designed MobileNet, which decompose the conventional convolution filters into the point-wise and depth-wise convolution filters with much fewer FLOPs. Zhang *et al*. [36] combined group convolutions [33] and a channel shuffle operation to build efficient neural networks with fewer computations. Hu *et al*. [14] proposed the squeeze and excitation block, which focuses on the relationship of channels by modeling interdependencies between channels, to improve the performance at slight additional computational cost. Wu *et al*. [32] presented a parameter-free "shift" operation with zero flop and zero parameter to replace conventional filters and largely reduce the computational and storage cost of CNNs. Zhong *et al*. [37] further pushed the shift-based primitives into channel shift, address shift and shortcut shift to reduce the inference time on GPU while keep the performance. Wang *et al*. [28] developed versatile convolution filters to generate more useful features utilizing fewer calculations and parameters.

2.3. Knowledge Distillation

Besides eliminating redundant weights or filters in deep convolutional neural networks, Hinton *et al*. [11] proposed the knowledge distillation (KD) scheme, which transfer useful information from a heavy teacher network to a portable student network by minimizing the Kullback-Leibler divergence between their outputs. Besides mimic the final outputs of the teacher networks, Romero *et al*. [25] exploit the hint layer to distill the information in features of the teacher network to the student network. You *et al*. [35] utilized multiple teachers to guide the training of the student network and achieve better performance. Yim *et al*. [34] regarded the relationship between features from two layers in the teacher network as a novel knowledge and introduced the FSP (Flow of Solution Procedure) matrix to transfer this kind of information to the student network.

Nevertheless, the compressed networks using these algorithms still contain massive multiplications, which costs enormous computation resources. As a result, subtractions or additions are of much lower computational complexities when compared with multiplications. However, they have not been widely investigated in deep neural networks, especially in the widely used convolutional networks. Therefore, we propose to minimize the numbers of multiplications in deep neural networks by replacing them with subtractions or additions.

3. Networks without Multiplication

Consider a filter $F \in \mathbb{R}^{d\times d\times c_{in}\times c_{out}}$ in an intermediate layer of the deep neural network, where kernel size is d, input channel is c_{in} and output channel is c_{out}. The input feature is defined as $X \in \mathbb{R}^{H\times W\times c_{in}}$, where H and W are the height and width of the feature, respectively. The output feature Y indicates the similarity between the filter and the input feature,

$$Y(m,n,t) = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} S\big(X(m+i,n+j,k), F(i,j,k,t)\big), \tag{1}$$

where $S(\cdot,\cdot)$ is a pre-defined similarity measure. If cross-correlation is taken as the metric of distance, *i.e*. $S(x,y) = x\times y$, Eq. (1) becomes the convolution operation. Eq. (1) can also implies the calculation of a fully-connected layer when $d = 1$. In fact, there are many other metrics to measure the distance between the filter and the input feature. However, most of these metrics involve multiplications, which bring in more computational cost than additions.

3.1. Adder Networks

We are therefore interested in deploying distance metrics that maximize the use of additions. ℓ_1 distance calculates the sum of the absolute differences of two points vector representations, which contains no multiplication. Hence, by calculating ℓ_1 distance between the filter and the input feature, Eq. (1) can be reformulated as

$$Y(m,n,t) = -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} |X(m+i,n+j,k) - F(i,j,k,t)|. \tag{2}$$

Addition is the major operation in ℓ_1 distance measure, since subtraction can be easily reduced to addition by using complement code. With the help of ℓ_1 distance, similarity between the filters and features can be efficiently computed.

Although both ℓ_1 distance Eq. (2) and cross-correlation in Eq. (1) can measure the similarity between filters and inputs, there are some differences in their outputs. The output of a convolution filter, as a weighted summation of values in the input feature map, can be positive or negative, but the output of an adder filter is always negative. Hence, we resort to batch normalization for help, and the output of adder layers will be normalized to an appropriate range and all the activation functions used in conventional CNNs can then be used in the proposed AdderNets. Although the batch normalization layer involves multiplications, its computational cost is significantly lower than that of the convolutional layers and can be omitted. Considering a convolutional layer with a filter $F \in \mathbb{R}^{d\times d\times c_{in}\times c_{out}}$, an input $X \in \mathbb{R}^{H\times W\times c_{in}}$ and an output $Y \in \mathbb{R}^{H'\times W'\times c_{out}}$, the computation complexity of convolution and batch normalization

is $\mathcal{O}(d^2 c_{in} c_{out} HW)$ and $\mathcal{O}(c_{out} H'W')$, respectively. In practice, given an input channel number $c_{in} = 512$ and a kernel size $d = 3$ in ResNet [9], we have $\frac{d^2 c_{in} c_{out} HW}{c_{out} H'W'} \approx 4068$. Since batch normalization layer has been widely used in the state-of-the-art convolutional neural networks, we can simply upgrade these networks into AddNets by replacing their convolutional layers into adder layers to speed up the inference and reduces the energy cost.

Intuitively, Eq. (1) has a connection with template matching [3] in computer vision, which aims to find the parts of an image that match the template. F in Eq. (1) actually works as a template, and we calculate its matching scores with different regions of the input feature X. Since various metrics can be utilized in template matching, it is natural that ℓ_1 distance can be utilized to replace the cross-correlation in Eq. (1).

3.2. Optimization

Neural networks utilize back-propagation to compute the gradients of filters and stochastic gradient descent to update the parameters. In CNNs, the partial derivative of output features Y with respect to the filters F is calculated as:

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = X(m+i, n+j, k), \tag{3}$$

where $i \in [m, m+d]$ and $j \in [n, n+d]$. To achieve a better update of the parameters, it is necessary to derive informative gradients for SGD. In AdderNets, the partial derivative of Y with respect to the filters F is:

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = \text{sgn}(X(m+i,n+j,k) - F(i,j,k,t)), \tag{4}$$

where $\text{sgn}(\cdot)$ denotes the sign function and the value of the gradient can only take +1, 0, or -1.

Considering the derivative of ℓ_2-norm

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = X(m+i, n+j, k) - F(i,j,k,t), \tag{5}$$

Eq. (4) can therefore lead to a signSGD [2] update of ℓ_2-norm. However, signSGD almost never takes the direction of steepest descent and the direction only gets worse as dimensionality grows [1]. It is unsuitable to optimize the neural networks of a huge number of parameters using signSGD. Therefore, we propose using Eq. (5) to update the gradients in our AdderNets. The convergence of taking these two kinds of gradient will be further investigated in the supplementary material. Therefore, by utilizing the full-precision gradient, the filters can be updated precisely.

Besides the gradient of the filters, the gradient of the input features X is also important for the update of parameters. Therefore, we also use the full-precision gradient (Eq. (5)) to calculate the gradient of X. However, the magnitude of the full-precision gradient may be larger than +1 or -1. Denote the filters and inputs in layer i as F_i and X_i. Different from $\frac{\partial Y}{\partial F_i}$ which only affects the gradient of F_i itself, the change of $\frac{\partial Y}{\partial X_i}$ would influence the gradient in not only layer i but also layers before layer i according to the gradient chain rule. If we use the full-precision gradient instead of the sign gradient of $\frac{\partial Y}{\partial X}$ for each layer, the magnitude of the gradient in the layers before this layer would be increased, and the discrepancy brought by using full-precision gradient would be magnified. To this end, we clip the gradient of X to $[-1, 1]$ to prevent gradients from exploding. Then the partial derivative of output features Y with respect to the input features X is calculated as:

$$\frac{\partial Y(m,n,t)}{\partial X(m+i, n+j, k)} = \text{HT}(F(i,j,k,t) - X(m+i, n+j, k)). \tag{6}$$

where $\text{HT}(\cdot)$ denotes the HardTanh function:

$$\text{HT}(x) = \begin{cases} x & \text{if} \quad -1 < x < 1, \\ 1 & x > 1, \\ -1 & x < -1. \end{cases} \tag{7}$$

3.3. Adaptive Learning Rate Scaling

In conventional CNNs, assuming that the weights and the input features are independent and identically distributed following normal distribution, the variance of the output can be roughly estimated as:

$$\begin{aligned} Var[Y_{CNN}] &= \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_{in}} Var[X \times F] \\ &= d^2 c_{in} Var[X] Var[F]. \end{aligned} \tag{8}$$

If variance of the weight is $Var[F] = \frac{1}{d^2 c_{in}}$, the variance of output would be consistent with that of the input, which will be beneficial for the information flow in the neural network. In contrast, for AdderNets, the variance of the output can be approximated as:

$$\begin{aligned} Var[Y_{AdderNet}] &= \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_{in}} Var[|X - F|] \\ &= (1 - \frac{2}{\pi}) d^2 c_{in} (Var[X] + Var[F]), \end{aligned} \tag{9}$$

when F and X follow normal distributions. In practice, the variance of weights $Var[F]$ is usually very small [7], *e.g.* 10^{-3} or 10^{-4} in an ordinary CNN. Hence, compared with multiplying $Var[X]$ with a small value in Eq. (8), the addition operation in Eq. (9) tends to bring in a much larger variance of outputs in AdderNets.

We next proceed to show the influence of this larger variance of outputs on the update of AdderNets. To promote

Algorithm 1 The feed forward and back propagation of adder neural networks.

Input: An initialized adder network $\mathcal{N}$ and its training set $\mathcal{X}$ and the corresponding labels $\mathcal{Y}$, the global learning rate γ and the hyper-parameter η.

1: **repeat**
2: Randomly select a batch $\{(\mathrm{x}, \mathrm{y})\}$ from $\mathcal{X}$ and $\mathcal{Y}$;
3: Employ the AdderNet $\mathcal{N}$ on the mini-batch: $\mathrm{x} \rightarrow \mathcal{N}(\mathrm{x})$;
4: Calculate the full-precision derivative $\frac{\partial Y}{\partial F}$ and $\frac{\partial Y}{\partial X}$ for adder filters using Eq. (5) and Eq. (6);
5: Exploit the chain rule to generate the gradient of parameters in $\mathcal{N}$;
6: Calculate the adapative learning rate α_l for each adder layer according to Eq. (13).
7: Update the parameters in $\mathcal{N}$ using stochastic gradient descent.
8: **until** convergence

Output: A well-trained adder network $\mathcal{N}$ with almost no multiplications.

the effectiveness of activation functions, we introduce batch normalization after each adder layer. Given input x over a mini-batch $\mathcal{B} = \{x_1, \cdots, x_m\}$, the batch normalization layer can be denoted as:

$$y = \gamma \frac{x - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} + \beta, \tag{10}$$

where γ and β are parameters to be learned, and $\mu_{\mathcal{B}} = \frac{1}{m}\sum_i x_i$ and $\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_i (x_i - \mu_{\mathcal{B}})^2$ are the mean and variance over the mini-batch, respectively. The gradient of loss ℓ with respect to x is then calculated as:

$$\frac{\partial \ell}{\partial x_i} = \sum_{j=1}^{m} \frac{\gamma}{m^2 \sigma_{\mathcal{B}}} \left\{ \frac{\partial \ell}{\partial y_i} - \frac{\partial \ell}{\partial y_j} [1 + \frac{(x_i - x_j)(x_j - \mu_{\mathcal{B}})}{\sigma_{\mathcal{B}}}] \right\}. \tag{11}$$

Given a much larger variance $Var[Y] = \sigma_{\mathcal{B}}$ in Eq. (9), the magnitude of the gradient w.r.t X in AdderNets would be much smaller than that in CNNs according to Eq. (11), and then the magnitude of the gradient w.r.t the filters in AdderNets would be decreased as a result of gradient chain rule.

Table 1. The ℓ_2-norm of gradient of weight in each layer using different networks at 1st iteration.

Model	Layer 1	Layer 2	Layer 3
AdderNet	0.0009	0.0012	0.0146
CNN	0.2261	0.2990	0.4646

Table 1 reports the ℓ_2-norm of gradients of filters $\|F\|_2$ in LeNet-5-BN using CNNs and AdderNets on the MNIST dataset during the 1st iteration. LeNet-5-BN denotes the LeNet-5 [18] adding an batch normalization layer after each convolutional layer. As shown in this table, the norms of gradients of filters in AdderNets are much smaller than that in CNNs, which could slow down the update of filters in AdderNets.

A straightforward idea is to directly adopt a larger learning rate for filters in AdderNets. However, it is worth noticing that the norm of gradient differs much in different layers of AdderNets as shown in Table 1, which requests special consideration of filters in different layers. To this end, we propose an adaptive learning rate for different layers in AdderNets. Specifically, the update for each adder layer l is calculated by

$$\Delta F_l = \gamma \times \alpha_l \times \Delta L(F_l), \tag{12}$$

where γ is a global learning rate of the whole neural network (*e.g.* for adder and BN layers), $\Delta L(F_l)$ is the gradient of the filter in layer l and α_l is its corresponding local learning rate. As filters in AdderNets act subtraction with the inputs, the magnitude of filters and inputs are better to be similar to extract meaningful information from inputs. Because of the batch normalization layer, the magnitudes of inputs in different layers have been normalized, which then suggests a normalization for the magnitudes of filters in different layers. The local learning rate can therefore be defined as:

$$\alpha_l = \frac{\eta \sqrt{k}}{\|\Delta L(F_l)\|_2}, \tag{13}$$

where k denotes the number of elements in F_l to average the ℓ_2-norm, and η is a hyper-parameter to control the learning rate of adder filters. By using the proposed adaptive learning rate scaling, the adder filters in different layers can be updated with nearly the same step. The training procedure of the proposed AdderNet is summarized in Algorithm 1.

4. Experiment

In this section, we implement experiments to validate the effectiveness of the proposed AdderNets on several benchmark datasets, including MNIST, CIFAR and ImageNet. Ablation study and visualization of features are provided to further investigate the proposed method. The experiments are conducted on NVIDIA Tesla V100 GPU in PyTorch.

4.1. Experiments on MNIST

To illustrate the effectiveness of the proposed AdderNets, we first train a LeNet-5-BN [18] on the MNIST dataset. The images are resized to 32×32 and are preprocessed following [18]. The networks are optimized using Nesterov Accelerated Gradient (NAG), and the weight decay and the momentum were set as 5×10^{-4} and 0.9, respectively. We train the networks for 50 epochs using the cosine learning rate decay [20] with an initial learning rate

Table 2. Classification results on the CIFAR-10 and CIFAR-100 datasets.

Model	Method	#Mul.	#Add.	XNOR	CIFAR-10	CIFAR-100
VGG-small	BNN	0	0.65G	0.65G	89.80%	65.41%
	AddNN	0	1.30G	0	93.72%	72.64%
	CNN	0.65G	0.65G	0	93.80%	72.73%
ResNet-20	BNN	0	41.17M	41.17M	84.87%	54.14%
	AddNN	0	82.34M	0	91.84%	67.60%
	CNN	41.17M	41.17M	0	92.25%	68.14%
ResNet-32	BNN	0	69.12M	69.12M	86.74%	56.21%
	AddNN	0	138.24M	0	93.01%	69.02%
	CNN	69.12M	69.12M	0	93.29%	69.74%

0.1. The batch size is set as 256. For the proposed AdderNets, we replace the convolutional filters in LeNet-5-BN with our adder filters. Note that the fully connected layer can be regarded as a convolutional layer, we also replace the multiplications in the fully connect layers with subtractions.

The convolutional neural network achieves a 99.4% accuracy with ∼435K multiplications and ∼435K additions. By replacing the multiplications in convolution with additions, the proposed AdderNet achieves a 99.4% accuracy, which is the same as that of CNNs, with ∼870K additions and almost no multiplication.In fact, the theoretical latency of multiplications in CPUs is also larger than that of additions and subtractions. There is an instruction table [1] which lists the instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. For example, in VIA Nano 2000 series, the latency of float multiplication and addition is 4 and 2, respectively. The AdderNet using LeNet-5 model will have ∼1.7M latency while CNN will have ∼2.6M latency in this CPU. In conclusion, the AdderNet can achieve similar accuracy with CNN but have fewer computational cost and latency. Noted that CUDA and cuDNN optimized adder convolutions are not yet available, we do not compare the actual inference time.

4.2. Experiments on CIFAR

We then evaluate our method on the CIFAR dataset, which consist of 32×32 pixel RGB color images. Since the binary networks [38] can use the XNOR operations to replace multiplications, we also compare the results of binary neural networks (BNNs). We use the same data augmentation and pro-precessing in He *et al*. [9] for training and testing. Following Zhou *et al*. [38], the learning rate is set to 0.1 in the beginning and then follows a polynomial learning rate schedule. The models are trained for 400 epochs with a 256 batch size. We follow the general setting in binary networks to set the first and last layers as full-precision convolutional layers. In AdderNets, we use the same setting for a fair comparison.

The classification results are reported in Table 2. Since computational cost in batch normalization layer, the first layer and the last layer are significantly less than other layers, we omit these layers when counting FLOPs. We first evaluate the VGG-small model [4] in the CIFAR-10 and CIFAR-100 dataset. As a result, the AdderNets achieve nearly the same results (93.72% in CIFAR-10 and 72.64% in CIFAR-100) with CNNs (93.80% in CIFAR-10 and 72.73% in CIFAR-100) with no multiplication. Although the model size of BNN is much smaller than those of AdderNet and CNN, its accuracies are much lower (89.80% in CIFAR-10 and 65.41% in CIFAR-100). We then turn to the widely used ResNet models (ResNet-20 and ResNet-32) to further investigate the performance of different networks. As for the ResNet-20, the convolutional neural networks achieve the highest accuracy (*i.e*. 92.25% in CIFAR-10 and 68.14% in CIFAR-100) but with a large number of multiplications (41.17M). The proposed AdderNets achieve a 91.84% accuracy in CIFAR-10 and a 67.60% accuracy in CIFAR-100 without multiplications, which is comparable with CNNs. In contrast, the BNNs only achieve 84.87% and 54.14% accuracies in CIFAR-10 and CIFAR-100. The results in ResNet-32 also suggest that the proposed AdderNets can achieve similar results with conventional CNNs.

4.3. Experiments on ImageNet

We next conduct experiments on the ImageNet dataset [17], which consist of 224×224 pixel RGB color images. We use ResNet-18 model to evaluate the proposed AdderNets follow the same data augmentation and pro-precessing in He *et al*. [9]. We train the AdderNets for 150 epochs utilizing the cosine learning rate decay [20]. These networks are optimized using Nesterov Accelerated Gradient (NAG), and the weight decay and the momentum are set as 10^{-4} and 0.9, respectively. The batch size is set as 256 and the hyper-parameter in AdderNets is the same as that in

[1] www.agner.org/optimize/instruction_tables.pdf

Table 3. Classification results on the ImageNet datasets.

Model	Method	#Mul.	#Add.	XNOR	Top-1 Acc.	Top-5 Acc.
ResNet-18	BNN	0	1.8G	1.8G	51.2%	73.2%
	AddNN	0	3.6G	0	67.0%	87.6%
	CNN	1.8G	1.8G	0	69.8%	89.1%
ResNet-50	BNN	0	3.9G	3.9G	55.8%	78.4%
	AddNN	0	7.7G	0	74.9%	91.7%
	CNN	3.9G	3.9G	0	76.2%	92.9%

(a) Visualization of filters of AdderNets

(b) Visualization of filters of CNNs

Figure 2. Visualization of filters in the first layer of LeNet-5-BN on the MNIST dataset. Both of them can extract useful features for image classification.

CIFAR experiments.

Table 3 shows the classification results on the ImageNet dataset by exploiting different nerual networks. The convolutional neural network achieves a 69.8% top-1 accuracy and an 89.1% top-5 accuracy in ResNet-18. However, there are 1.8G multiplications in this model, which bring enormous computational complexity. Since the addition operation has smaller computational cost than multiplication, we propose AdderNets to replace the multiplications in CNNs with subtractions. As a result, our AdderNet achieve a 66.8% top-1 accuracy and an 87.4% top-5 accuracy in ResNet-18, which demonstrate the adder filters can extract useful information from images. Rastegari *et al*. [22] proposed the XNOR-net to replace the multiplications in neural networks with XNOR operations. Although the BNN can achieve high speed-up and compression ratio, it achieves only a 51.2% top-1 accuracy and a 73.2% top-5 accuracy in ResNet-18, which is much lower than the proposed AdderNet. We then conduct experiments on a deeper architecture (ResNet-50). The BNN could only achieve a 55.8% top-1 accuracy and a 78.4% top-5 accuracy using ResNet-50. In contrast, the proposed AdderNets can achieve a 74.9% top-1 accuracy and a 91.7% top-5 accuracy, which is closed to that of CNN (76.2% top-1 accuracy and 92.9% top-5 accuracy).

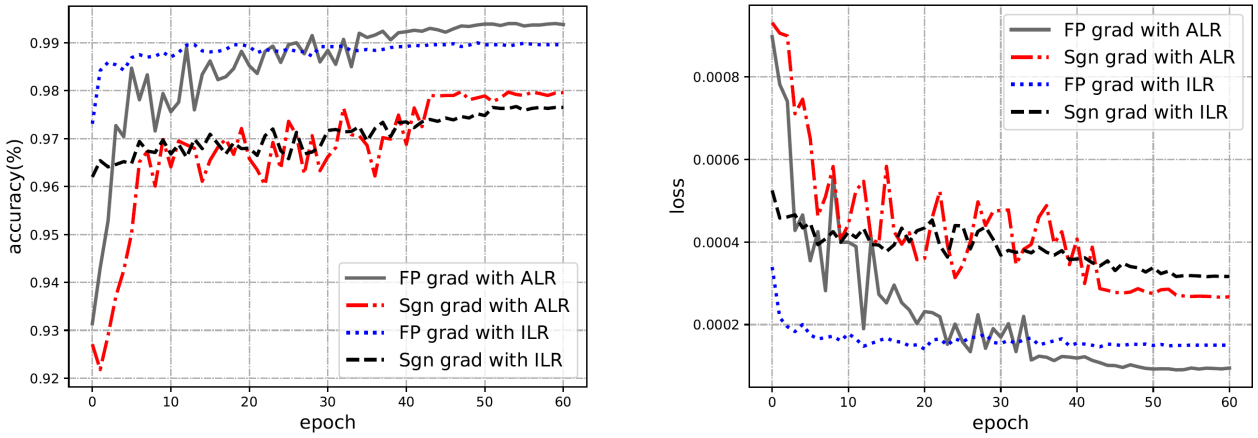
4.4. Visualization Results

Visualization on features. The AdderNets utilize the ℓ_1-distance to measure the relationship between filters and input features instead of cross correlation in CNNs. Therefore, it is important to further investigate the difference of the feature space in AdderNets and CNNs. We train a LeNet++ on the MNIST dataset following [31], which has six convolutional layers and a fully-connected layer. Numbers of neurons in each convolutional layer are 32, 32, 64, 64, 128, 128, and 2, respectively. For the proposed AdderNets, convolutional and fully connected layers are replaced with the proposed add filters.

The visualization results are shown in Figure 1. The convolutional neural network calculates the cross correlation between filters and inputs. If filters and inputs are approximately normalized, convolution operation is then equivalent to calculate cosine distance between two vectors. That is probably the reason that features in different classes are divided by their angles in Figure 1. In contrast, AdderNets utilize the ℓ_1-norm to distinguish different classes. Thus, features tend to be clustered towards different class centers. The visualization results demonstrate that the proposed AdderNets could have the similar discrimination ability to classify images as CNNs.

Visualization on filters. We visualize the filters of the LeNet-5-BN network in Figure 2. Although the AdderNets and CNNs utilize different distance metrics, filters of the proposed adder networks (see Figure 2 (a)) still share some similar patterns with convolution filters (see Figure 2 (b)). The visualization experiments further demonstrate that the filters of AdderNets can effectively extract useful information from the input images and features.

Visualization on distribution of weights As the proposed AdderNets use ℓ. We then visualize the distribution of weights for the 3th convolution layer on LeNet-5-BN. As shown in Figure 4, the distribution of weights with AdderNets is close to a Laplace distribution while that with CNNs looks more like a Gaussian distribution. In fact, the prior distribution of ℓ_1-norm is Laplace distribution [27] and that of ℓ_2-norm is Gaussian distribution [24] and the ℓ_2-norm



(a) Accuracy (b) Loss

Figure 3. Learning curve of AdderNets using different optimization schemes. FP and Sgn gradient denotes the full-precision and sign gradient. The proposed adaptive learning rate scaling with full-precision gradient achieves the highest accuracy (99.40%) with the smallest loss.

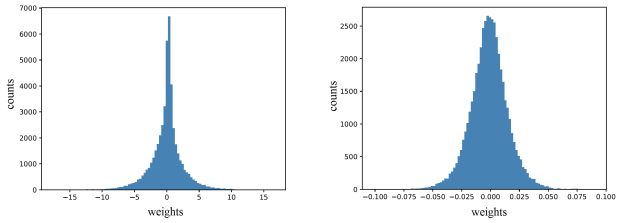


Figure 4. Histograms over the weights with AdderNet (left) and CNN (right). The weights of AdderNets follow Laplace distribution while those of CNNs follow Gaussian distribution.

is exactly same as the cross correlation, which will be analyzed in the supplemental material.

4.5. Ablation Study

We propose to use a full-precision gradient to update the filters in our adder filters and design an adaptive learning rate scaling for deal with different layers in AdderNets. It is essential to evaluate the effectiveness of these components. We first train the LeNet-5-BN without changing its learning rate, which results in 54.91% and 29.26% accuracies using full-precision gradient and sign gradient, respectively. The networks can be hardly trained since its gradients are very small. Therefore, it is necessary to increase the learning rate of adder filters.

We directly increase the learning rate for filters in AdderNets by 100, which achieves best performance with full-precision gradient compared with other values from the pool $\{10, 50, 100, 200, 500\}$. As shown in Figure 3, the AdderNets using adaptive learning rate (ALR) and increased learning rate (ILR) achieve 97.99% and 97.72% accuracy with sign gradient, which is much lower than the accuracy of CNN (99.40%). Therefore, we propose the full-precision gradient to precisely update the weights in AdderNets. As a result, the AdderNet with ILR achieves a 98.99% accuracy using the full-precision gradient. By using the adaptive learning rate (ALR), the AdderNet can achieve a 99.40% accuracy, which demonstrate the effectiveness of the proposed ALR method.

5. Conclusions

The role of classical convolutions used in deep CNNs is to measure the similarity between features and filters, and we are motivated to replace convolutions with more efficient similarity measure. We investigate the feasibility of replacing multiplications by additions in this work. An AdderNet is explored to effectively use addition to build deep neural networks with low computational costs. This kind of networks calculate the ℓ_1-norm distance between features and filters. Corresponding optimization method is developed by using regularized full-precision gradients. Experiments conducted on benchmark datasets show that AdderNets can well approximate the performance of CNNs with the same architectures, which could have a huge impact on future hardware design. Visualization results also demonstrate that the adder filters are promising to replace original convolution filters for computer vision tasks. In our future work, we will investigate quantization results for AdderNets to achieve higher speed-up and lower energy comsumption, as well as the generality of AdderNets not only for image classification but also for detection and segmentation tasks.

References

[1] Jeremy Bernstein, Kamyar Azizzadenesheli, Yu-Xiang Wang, and Anima Anandkumar. Convergence rate of sign stochastic gradient descent for non-convex functions. 2018. 4

[2] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018. 4

[3] Roberto Brunelli. *Template matching techniques in computer vision: theory and practice*. John Wiley & Sons, 2009. 4

[4] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *CVPR*, pages 5918–5926, 2017. 1, 6

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NeuriPS*, pages 3123–3131, 2015. 1

[6] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NeuriPS*, 2014. 2

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 4

[8] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 3, 6

[10] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, pages 1389–1397, 2017. 2

[11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 3

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

[13] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 2

[14] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, pages 7132–7141, 2018. 3

[15] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeuriPS*, pages 4107–4115, 2016. 1

[16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. 2017. 3

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeuriPS*, pages 1097–1105, 2012. 1, 6

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 5

[19] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015. 1

[20] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 5, 6

[21] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5058–5066, 2017. 2

[22] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542. Springer, 2016. 1, 7

[23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeuriPS*, pages 91–99, 2015. 1

[24] Jason Rennie. On l2-norm regularization and the gaussian prior. 2003. 7

[25] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 3

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1

[27] Stephen M Stigler. *The history of statistics: The measurement of uncertainty before 1900*. Harvard University Press, 1986. 7

[28] Yunhe Wang, Chang Xu, Chunjing Xu, Chao Xu, and Dacheng Tao. Learning versatile filters for efficient convolutional neural networks. In *NeuriPS*, pages 1608–1618, 2018. 3

[29] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. Cnnpack: Packing convolutional neural networks in the frequency domain. In *NeuriPS*, pages 253–261, 2016. 2

[30] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *ECCV*, pages 499–515. Springer, 2016. 1

[31] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *ECCV*, 2016. 7

[32] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *CVPR*, pages 9127–9135, 2018. 3

[33] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 1492–1500, 2017. 3

[34] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *CVPR*, pages 4133–4141, 2017. 3

[35] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *SIGKDD*, pages 1285–1294. ACM, 2017. 3

[36] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, pages 6848–6856, 2018. 3

[37] Huasong Zhong, Xianggen Liu, Yihui He, Yuchun Ma, and Kris Kitani. Shift-based primitives for efficient convolutional neural networks. *arXiv preprint arXiv:1809.08458*, 2018. 3

[38] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 1, 6

[39] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *NeuriPS*, pages 875–886, 2018. 2