## Title

The Effects of Approximate Multiplication on Convolutional Neural Networks

**Permalink**

https://escholarship.org/uc/item/30g8q297

**Journal**

IEEE Transactions on Emerging Topics in Computing, 10(2)

**ISSN**

2376-4562

**Authors**

Kim, Min Soo
Del Barrio, Alberto A
Kim, HyunJin
et al.

**Publication Date**

2022

**DOI**

10.1109/tetc.2021.3050989

The manuscript has been accepted for publication in the IEEE Transactions on Emerging Topics in Computing.

# The Effects of Approximate Multiplication on Convolutional Neural Networks

Min Soo Kim, Alberto A. Del Barrio, *Senior Member, IEEE*, HyunJin Kim, Nader Bagherzadeh, *Fellow, IEEE*

**Abstract**—This paper analyzes the effects of approximate multiplication when performing inferences on deep convolutional neural networks (CNNs). The approximate multiplication can reduce the cost of the underlying circuits so that CNN inferences can be performed more efficiently in hardware accelerators. The study identifies the critical factors in the convolution, fully-connected, and batch normalization layers that allow more accurate CNN predictions despite the errors from approximate multiplication. The same factors also provide an arithmetic explanation of why bfloat16 multiplication performs well on CNNs. The experiments are performed with recognized network architectures to show that the approximate multipliers can produce predictions that are nearly as accurate as the FP32 references, without additional training. For example, the ResNet and Inception-v4 models with Mitch-$w$6 multiplication produces Top-5 errors that are within 0.2% compared to the FP32 references. A brief cost comparison of Mitch-$w$6 against bfloat16 is presented where a MAC operation saves up to 80% of energy compared to the bfloat16 arithmetic. The most far-reaching contribution of this paper is the analytical justification that multiplications can be approximated while additions need to be exact in CNN MAC operations.

**Index Terms**—Machine learning , Computer vision, Object recognition, Arithmetic and logic units, Low-power design

## 1 Introduction

The computational costs of convolutional neural networks (CNNs) have increased as CNNs get wider and deeper to perform better predictions for a variety of applications. For deep learning to have revolutionary impact on real-world applications, their computational costs must meet the timing, energy, monetary, and other design constraints of the deployed services. Many approaches have been studied to reduce the computational costs at all levels of software and hardware, from advances in network architectures [1], [2] down to electronics where even memory devices have been extensively researched [3], [4].

Although training requires more computations when compared to inference, it is still important to reduce the cost of inference as much as possible because it is the inference that is usually subject to more strict real-world design constraints. Many hardware-based approaches have shown significant improvements for the computational costs of CNN inferences, but there are two limitations commonly found in these works. Some techniques are computationally expensive in order to optimize their methods for each network model, or to retrain networks to compensate for the performance degradation from their methods [5], [6]. Also, many techniques such as [7] are only effective for small networks and cannot scale to deeper CNNs as they report much worse performance results when tested for deeper networks. They leverage the fact that a small number of bits are sufficient for small CNNs, but more complex networks require more bits to properly represent the amount of information [8].

One promising hardware-based approach is the application of approximate multiplication to CNN inference [9]. It involves designing and applying multiplication circuits that have reduced hardware costs but produce results that are not exact. Unlike aggressive quantization that trades off numeric precision, the multipliers trade off arithmetic accuracy that is less dependent on the network models, making them better suited for deeper CNNs. The approach does not involve any optimization to a target network model or require additional processing of the network models, allowing easy adaptation into the ASIC and FPGA accelerators.

While optimizing CNN inference through approximate multiplication was demonstrated in several previous studies, there was limited understanding of why it worked well for CNNs. The promising results led to the general observation that CNNs were resilient against small arithmetic errors, but none of them identified the complete reason behind that resilience. Specifically, it was unclear how the CNN layers preserved their functionalities when all their multiplications have a certain amount of error. The lack of understanding made it challenging to identify the suitable approximate multiplier for each network model, leading to expensive search-based methodologies in some studies [10].

This paper investigates how the errors from approximate multiplication affect deep CNN inference. The work is motivated by hardware circuits but it focuses on the implications from the Deep Learning perspective.

The contributions are summarized as follows:

- Explaining how convolution and fully-connected (FC) layers maintain their intended functionalities despite approximate multiplications.
- Demonstrating how batch normalization can prevent the buildup of error in deeper layers when its parameters are properly adjusted.
- Discussing how these findings also explain why bfloat16 multiplication performs well on CNNs despite the reduction of precision.

- Performing experiments to show that deep CNNs with approximate multiplication perform reasonably well.
- Discussing the potential cost benefits of the methodology by briefly comparing the hardware costs against those of bfloat16 arithmetic.

## 2 Preliminaries

The convolution layers in CNNs consist of a large number of multiply-accumulate (MAC) operations and they take up the majority of computations for CNN inferences [11]. The MAC operations are ultimately performed in the hardware circuits, and it is important to minimize the cost of these circuits to perform more computations with the same amount of resources. For MAC operations, multiplications are more complex than additions and consume most resources. The proposed methodology consists of minimizing the cost of multiplication by replacing the conventional multipliers with approximate multipliers.

Approximate multipliers are significantly cheaper compared to the exact multipliers but they introduce errors in the results. There are many different types of approximate multipliers with various costs and error characteristics. Some designs use the electronic properties [12] and some approximate by intentionally flipping bits in the logic [13], while others use algorithms to approximate multiplication [14].

This paper studies the effects of approximate multiplication with the approximate log multiplier presented in [9] as well as a few other promising designs. The approximate log multiplication is based on the Mitchell's Algorithm [15] that performs multiplications in the log domain. Fig. 1 shows the difference between the conventional fixed-point multiplier and the log multiplier. An important benefit of the algorithm-based approximation is the consistent error characteristics which allow for consistent observation of the effects across various CNN instances. The other types of approximation have more inconsistent errors that make them ill-suited for the study. For example, approximate multipliers based on electronic properties depend not only on the operands but also on Process, Voltage, and Temperature (PVT) variations, making it difficult to get consistent observations. The findings of this study are not limited to log multiplication, however, and may help explain the viability of other approaches when they meet the conditions discussed in Section 3.2.

The errors from approximate log multiplication are deterministic and depend on the two input operands, similarly to the other algorithmic approximation methods. Fig. 2 shows the error patterns of the original Mitchell log multiplier [15] and Mitch-$w6$ [9] with a million random input pairs. The relative error is defined as Equation 1 where $|Z|$ is the magnitude of the exact product and $|Z'|$ is the magnitude of the approximate product.

$$error_{relative} = \frac{|Z'| - |Z|}{|Z|}. \tag{1}$$

Approximate log multiplication requires separate sign handling and does not affect the signs of the products [9]. Compared to the original Mitchell log multiplier, Mitch-$w6$ has a small frequency of high relative errors caused by the 1's complement (C1) sign handling, but they are acceptable as CNNs consist of MAC operations [9]. It should be noted that the approximate log multipliers have reasonably even distributions of errors across the input ranges, but can only have negative errors that cause the products to have less magnitudes compared to the exact products. The mean error of an approximate multiplier is measured by repeating many multiplications with random inputs, and the Mitchell multiplier has the biased mean error of -3.9% at 32 bits while Mitch-$w6$ has -5.9%.

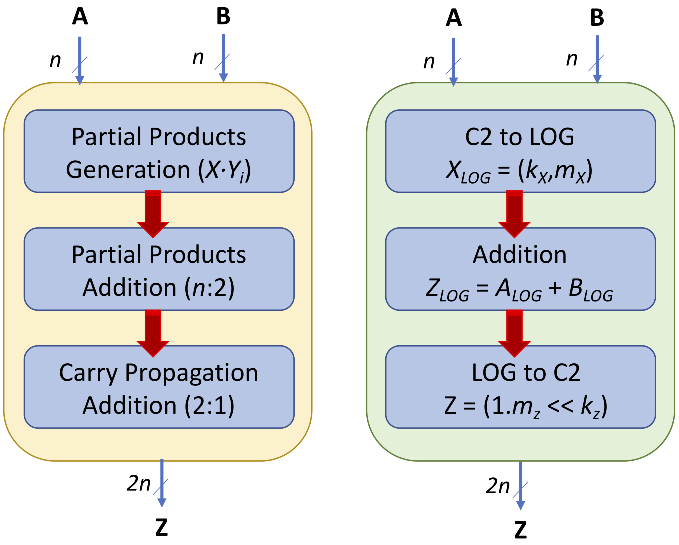
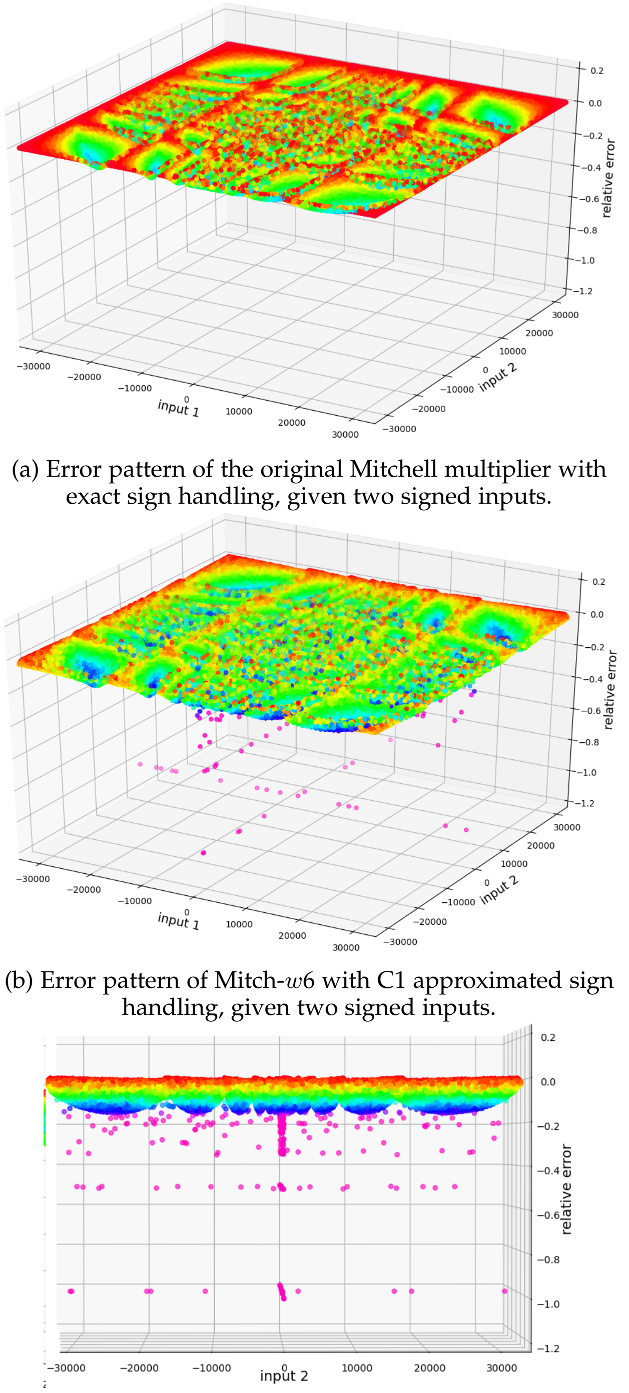

A
B
n
n
Partial Products Generation ($X \cdot Y_i$)
Partial Products Addition ($n$:2)
Carry Propagation Addition (2:1)
2n
Z
A
B
n
n
C2 to LOG $X_{LOG} = (k_X, m_X)$
Addition $Z_{LOG} = A_{LOG} + B_{LOG}$
LOG to C2 $Z = (1.m_z << k_z)$
2n
Z

Fig. 1: Difference between (a) the conventional fixed-point multiplication and (b) the approximate log multiplication. $k$ stands for characteristic and $m$ stands for mantissa of logarithm.

Besides the convolution layers, the FC layers also have MAC operations but they have fewer computations compared to convolution [11]. Our methodology still applies to approximate multiplication of FC layers to be consistent with networks that use 1x1 convolution for classifiers. The effect of approximating FC layers is minimal because of the reasons discussed in Section 3. On the other hand, the operations in batch normalization are not approximated because they can be absorbed into neighboring layers during inferences [16].

It is important to understand the difference between the method of quantization and the approximate multiplication. Quantization is the process of converting floating-point values in the CNN models to fixed-point for more cost-efficient inferences in the hardware [16]. The goal of quantization is to find the minimum number of fixed-point bits that can sufficiently represent the distribution of values. In fact, there are some approximations with small numbers of fixed-point bits that cannot match the range and precision of the floating-point format. The error from this approximation depends on the network models as each has different distributions of values [8], [17]. The network dependency is the reason why more complex networks require a higher number of bits and the benefits of aggressive quantization



(a) Error pattern of the original Mitchell multiplier with exact sign handling, given two signed inputs.

(b) Error pattern of Mitch-$w6$ with C1 approximated sign handling, given two signed inputs.

(c) Error pattern of Mitch-$w6$, viewed from the side.

Fig. 2: Error patterns of approximate log multipliers.

diminish. While many studies have successfully demonstrated the effectiveness of quantization, they usually report significant degradation of CNN prediction accuracies when using only 8 bits on deep CNNs [18].

Approximate multiplication is less dependent on the networks because its source of error is from the approximation methods, not any lack of range and precision. Given proper quantization, the approximate multiplication further minimizes the cost of multipliers for the given number of bits. Approximate multiplication is an orthogonal approach to quantization as approximate multipliers may be designed for any number of bits, and it complements quantization to maximize the computational efficiency of CNN inferences.

## 3 Accumulated Error in Convolution

This section explains how the convolution and FC layers achieve their intended functionalities despite the errors from approximate multiplication.

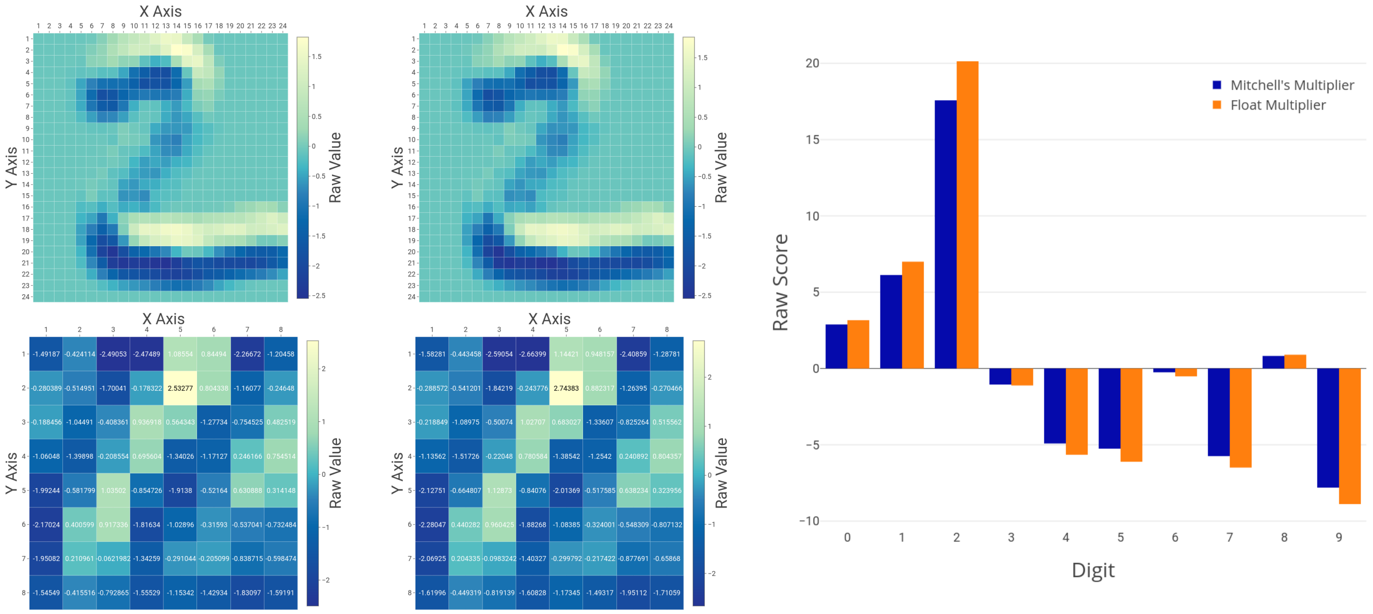
### 3.1 Understanding Convolution and FC Layers

Explaining the effects of approximate multiplication must begin with understanding how the convolution and FC layers achieve their intended functionalities. Fig. 3 is taken from [9] and shown here to visualize the outputs of convolution and FC. The CNN convolution layers achieve abstract feature detection by performing convolution between their input channels and kernels. They produce feature maps, as shown in Fig. 3a and 3b, where the locations that match the kernel are represented by high output values relative to other locations. Unlike a sigmoid or step activation, the widely used ReLU activation function simply forces the negative output values to zero and does not have absolute thresholds with which the abstract features are identified. That means the abstract features are not identified by their absolute values but by the relatively higher values within each feature map, and this claim is also supported by the fact that convolution is often followed by a pooling layer. Similarly, when the FC layers classify an image based on the abstract features, the probabilities of classes are decided by the relative strengths and order among all FC outputs. CNNs simply select the best score as the most probable prediction instead of setting a threshold with which a prediction is made.

Because the features are represented with relative values as opposed to absolute values, it is much more important to minimize the variance of error between the convolution outputs than minimizing the absolute mean of errors when applying approximate multiplication to convolution [9]. In other words, it is acceptable to have a certain amount of error in multiplications as long as the errors affect all outputs of convolution as equally as possible. The FC layers behave in the same way so that it is important to minimize the variance of error between the nodes. Fig. 3 demonstrates this principle and shows that the Mitchell log multiplier can produce a correct inference because all outputs are affected at the same time. Fig. 3 also shows that the variances of accumulated errors in the convolution and FC layers are very small when the approximate log multiplier is applied, and the convolutions are still able to locate the abstract features albeit with smaller magnitudes. The previous work [9], however, did not identify the reason why the variance of accumulated error was minimized when approximate multiplication was applied.

### 3.2 Minimized Variance of Error

This paper provides the analytical explanation for why the variance of accumulated error was minimized in the convolution and FC layers. These layers consist of large numbers of multiplications and accumulations that converge the



| Y Axis \ X Axis | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | -1.49187 | -0.424114 | -2.49053 | -2.47489 | 1.08554 | 0.84494 | -2.26672 | -1.20458 |
| 2 | -0.280389 | -0.514951 | -1.70041 | -0.178322 | 2.53277 | 0.804338 | -1.16077 | -0.24648 |
| 3 | -0.188456 | -1.04491 | -0.408361 | 0.936918 | 0.564343 | -1.27734 | -0.754525 | 0.482519 |
| 4 | -1.06048 | -1.39898 | -0.208554 | 0.695604 | -1.34026 | -1.17127 | 0.246166 | 0.754514 |
| 5 | -1.99244 | -0.581799 | 1.03502 | -0.854726 | -1.9138 | -0.52164 | 0.630888 | 0.314148 |
| 6 | -2.17024 | 0.400599 | 0.917336 | -1.81634 | -1.02896 | -0.31593 | -0.537041 | -0.732484 |
| 7 | -1.95082 | 0.210961 | -0.0621982 | -1.34259 | -0.291044 | -0.205099 | -0.838715 | -0.598474 |
| 8 | -1.54549 | -0.415516 | -0.792865 | -1.55529 | -1.15342 | -1.42934 | -1.83097 | -1.59191 |

(a) Convolution by Log Mult.

| Y Axis \ X Axis | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | -1.58281 | -0.443458 | -2.59054 | -2.66399 | 1.14421 | 0.948157 | -2.40859 | -1.28781 |
| 2 | -0.288572 | -0.541201 | -1.84219 | -0.243776 | 2.74383 | 0.882317 | -1.26395 | -0.270466 |
| 3 | -0.218849 | -1.08975 | -0.50074 | 1.02707 | 0.683027 | -1.33607 | -0.825264 | 0.515562 |
| 4 | -1.13562 | -1.51726 | -0.22048 | 0.780584 | -1.38542 | -1.2542 | 0.240892 | 0.804357 |
| 5 | -2.12751 | -0.664807 | 1.12873 | -0.84076 | -2.01369 | -0.517585 | 0.638234 | 0.323956 |
| 6 | -2.28047 | 0.440282 | 0.960425 | -1.88268 | -1.08385 | -0.324001 | -0.548309 | -0.807132 |
| 7 | -2.06925 | 0.204335 | -0.0983242 | -1.40327 | -0.299792 | -0.217422 | -0.877691 | -0.65868 |
| 8 | -1.61996 | -0.449319 | -0.819139 | -1.60828 | -1.17345 | -1.49317 | -1.95112 | -1.71059 |

(b) Convolution by Float Mult.

(c) The final scores

Fig. 3: Convolution outputs and the final raw scores of a sample inference from LeNet [9].

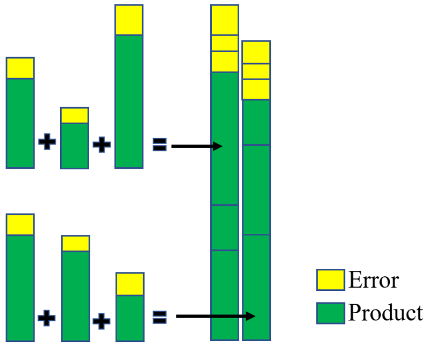


Fig. 4: Accumulation of many products with varying amount of error converges the combined errors to a mean value.

accumulated errors to a mean value. The variance of the accumulated error is minimized and all outputs of the layers are equally affected because of this convergence, and then maintaining the relative magnitudes between the outputs preserves the functionality of abstract feature detection.

Equation 2 shows the multi-channel convolution where feature $s$ at $(i,j)$ is the accumulation of products between kernel $w$ and input $x$ across the kernel dimensions $(m,n)$ and the input channels $(l)$.

$$s_{i,j} = \sum_{l}\sum_{m}\sum_{n} w_{l,m,n} \cdot x_{l,i-m,j-n} \ . \tag{2}$$

The distributions of weights and inputs are different for each CNN model and layer [8], [11], [17]. The input operands to multiplication, weights and input pixels, are numerous and practically unpredictable with pseudo-randomness, which in turn makes the error from approximate multiplication pseudo-random. The approximate log multipliers have evenly distributed error patterns across the input ranges, as shown in Fig. 2, and therefore the expected value of the error is close to the mean error of the approximate multiplier regardless of the different ranges of inputs from CNNs. When each convolution output accumulates many products from approximate multiplication, the accumulated error statistically converges closer to the expected value, which is the mean error of the approximate multiplier. This convergence reduces the variance of the accumulated error between the outputs and the values scale by roughly the same amount, minimizing the effect of varying error on feature detection. Fig. 4 shows the abstraction of this mechanism and Fig. 3 shows an example. Equation 3 describes the feature $s'_{i,j}$ when multiplications are associated with the mean error of $e$.

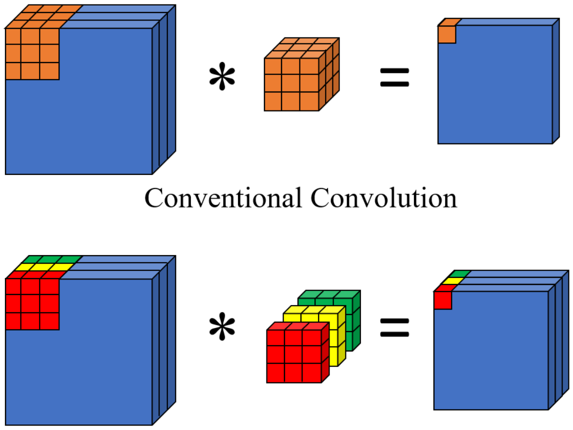
$$s'_{i,j} = \sum_{l}\sum_{m}\sum_{n} w_{l,m,n} \cdot x_{l,i-m,j-n} \cdot (1+e) \ , \tag{3}$$

$$s'_{i,j} = (1+e) \cdot s_{i,j} \ . \tag{4}$$

Therefore, the features are simply scaled by the mean error of the approximate multiplication when a large number of products are accumulated.

The above observations hold only for the approximate multiplications with the symmetric errors between positive and negative products so that Equations 3 and 4 hold. The approximate multipliers studied in this paper satisfy this condition because all of them handle the signs separately from magnitudes.

Although we primarily used the Mitch-$w$ multiplier to develop this hypothesis, the hypothesis does not depend on the inner workings of the log multiplier but only on the output error characteristics. Therefore, the theory can be similarly applied to any approximate multiplier that meets the assumptions made in this section, namely the evenly distributed error and the symmetric errors between positive and negative products. Having only negative errors



Conventional Convolution

Depthwise Convolution

Fig. 5: Depthwise Convolution has a reduced number of accumulations and convergence of error per output.

like Mitch-$w$ is not a requirement. It should be noted that the assumption of an evenly distributed error is used to accommodate different ranges of inputs, and may be relaxed when an approximate multiplier can produce a consistent expected value of error for particular input distributions. In this paper, we also used DRUM6 [19] and the truncated iterative log multiplier [20] for the experiments in Section 6 to show that the hypothesis may be applied to other approximate multipliers.

### 3.3 Impact on Convolution and FC

The number of accumulations in convolution is finite so the convergence does not completely nullify the variance of accumulated error. The small amount of error variance from approximate multiplication is acceptable, however, because CNNs are designed to be general and robust against small variations by nature. The techniques of regularization, such as pooling and dropout, intentionally lose some information to suppress overfitting and increase the generality of CNN predictions. Some studies have observed that small arithmetic errors have similarly positive effects [9], [21], [22]. For example, an eye needs to be recognized as an eye even when it is a little different from the training samples. CNNs are designed to overlook such small differences, and some computational inaccuracies are not only tolerable but often beneficial in providing such generality.

Deep CNNs typically start with smaller numbers of convolution channels to obtain general features, and the number of channels increases in the deeper layers where features become more specific. Approximate multiplication on such CNNs exhibits the desired trend of having smaller effects in the wide and deep layers as required. The larger variance of accumulated error in the shallow layers is tolerable because the feature detection needs to account for the small variations in the input images. In fact, some previous works, such as [14], [23], had claimed that earlier layers can be approximated more in neural networks.

This hypothesis implies the importance of exact additions in CNNs because the multiplication errors will not converge properly with inexact accumulations. This agrees with the work in [13] where approximating the additions had a larger impact on the CNN accuracies. As multipliers in fixed-point arithmetic are much more expensive than adders, approximating only the multipliers gains the most benefit with minimal degradation in CNN inferences.

Approximate multiplication also benefits from the fact that the convolution outputs receive inputs from the same set of input channels. For each convolution output, there are two types of accumulations. One type occurs within each input channel across the kernel dimensions while the other occurs across the input channels to produce the final output. The intra-channel accumulation combines the products from the same input channel and kernel, and therefore each channel has a specific range of values within which features are located. The inter-channel accumulation may have more varying ranges of products because each input channel has its own kernel and input values. Different input ranges may trigger different error characteristics on the approximate multiplier, but every convolution output accumulates from all input channels so that it does not affect the variance of accumulated error between the outputs. An implication of this observation is that approximate multiplication does not work as well when every output does not accumulate from the same set of data, as in the cases of grouped convolution and branches in CNN architectures.

The FC layers are also resilient against the effects of approximate multiplication as the same factors help converge errors in the outputs. There is usually a large number of accumulations per each output and all outputs share the same set of inputs. Thus, CNN accuracies show minimal differences when the FC layers have approximate multiplications as demonstrated in Section 6.

### 3.4 Grouped and Depthwise Convolutions

The benefits of approximate multiplication with conventional convolution are best understood and verified by comparing against grouped and depthwise separable convolution. Depthwise separable convolution consists of depthwise convolution followed by pointwise convolution [2]. Depthwise convolution is a special case of grouped convolution that eliminates the accumulation across input channels, and the reduced number of accumulations leads to an increase in the variance of accumulated error in the outputs. Fig. 5 shows the comparison of the accumulation pattern between conventional convolution and depthwise convolution. Also, each output channel receives inputs from only one input channel and the difference of error between output channels is subject to another approximate multiplication and variance of error before the inter-channel accumulations occur in the following pointwise convolution. More accurate approximate multipliers are required for CNNs that use depthwise separable convolution because errors from approximate multiplication do not converge well. A sufficiently accurate approximate multiplier can still perform reasonably well, as demonstrated in Section 6.

Another technique that reduces the number of accumulations is 1x1 convolution, but it is found to be compatible with approximate multipliers. 1x1 convolution does not have any intra-channel accumulation but accumulates the products across input channels. Because deep CNNs require

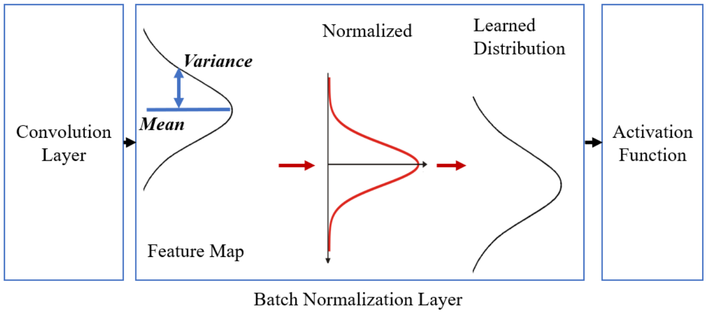


Fig. 6: Abstract overview of batch normalization.

large numbers of channels appropriate for their deep structures, inputs to 1x1 convolutions usually consist of many input channels and therefore provide enough accumulations for the error convergence. Each output of 1x1 convolution also receives inputs from all input channels, which provides more consistent accumulation of error between the outputs.

## 4 Effect of Batch Normalization

The approximate log multiplication with Mitchell's Algorithm generates negative error in the results, meaning that the product has less magnitude compared to the exact multiplication [15]. It is evident from Equation 4 that the features have less magnitudes with the log multiplication in each convolution layer. There are many convolution layers that repeatedly cause the reduction, and the previous work had reported that this became a problem for deeper layers [9]. Its adverse effect on the network performance was observable in AlexNet with only 8 layers of convolution and FC, and it was unclear how the mean error accumulation would behave in much deeper networks. Having tens or hundreds of convolution layers significantly reduces the magnitudes of the features so that the deeper layers receive input distributions that are difficult to distinguish. On the other hand, if an approximate multiplier has a positively biased mean error, it is possible to amplify the values beyond the range set by quantization, resulting in the arithmetic overflow. These adverse effects are under the best-case scenario of ReLU activation, and the other types such as a sigmoid function may suffer additional errors in activations. The ReLU function simply forces the negative values to zero and does not change the magnitudes of positive inputs, but the same is not true for other activation functions where the magnitudes of positive inputs cause changes in activations.

Batch normalization [24], the popular technique used in most deep CNNs, can alleviate this problem and help approximate multiplication go deeper into the networks. A critical function of batch normalization is to redistribute the output feature maps to have more consistent input distributions for deeper layers. While the training process necessitates this function, the inferences on the resulting models still need to go through the normalization with the stored global parameters of expected distributions. These global parameters can be appropriately adjusted to account for the changes in the distributions due to approximate multiplication, and this can prevent the accumulation of mean error across the layers.

The abstract overview of batch normalization is shown in Fig. 6. During training, each batch normalization layer calculates and stores the mean and variance values of the input distributions. These mean and variance values are used to normalize the input distributions to generate the normalized distributions with the mean value of zero and the variance of one. Then, batch normalization uses learnable parameters to scale and shift the normalized distribution to restore the representation power of the network [24]. In essense, batch normalization redistributes the feature maps before or after the activation function so that the next layer may receive consistent distributions of inputs. All these parameters are learned during training and stored as numerical values in CNN models, and they can be easily modified if necessary. CNN inferences use these stored parameters to perform normalization assuming they represent the same input distributions during inferences.

The mean and variance parameters are a source of error for approximate multiplication without proper adjustments because the distribution of convolution outputs changes as the result of approximate multiplication. Equations 7 and 10 show the mean ($\mu'$) and variance ($(\sigma')^2$) of the convolution output distribution, when the features $s'_{i,j}$ have the mean error $e$ from Equation 4.

$$\mu' = 1/m \sum_{i,j} s'_{i,j} \ , \tag{5}$$

$$\mu' = 1/m \sum_{i,j} (1+e) \cdot s_{i,j} \ , \tag{6}$$

$$\mu' = (1+e)\mu \ . \tag{7}$$

$$(\sigma')^2 = 1/m \sum_{i,j} (s'_{i,j} - \mu')^2 \ , \tag{8}$$

$$(\sigma')^2 = 1/m \sum_{i,j} (1+e)^2 (s_{i,j} - \mu)^2 \ , \tag{9}$$

$$(\sigma')^2 = (1+e)^2 \cdot \sigma^2 \ . \tag{10}$$

Therefore, the stored mean values for batch normalization must be scaled by $(1+e)$, while the variance values are scaled by $(1+e)^2$. With the adjusted parameters, the batch normalization layers correctly normalize the convolution outputs and scale them back to the desired distributions. In the process, the mean and variance of the outputs match those of exact multiplication and the effect of mean error accumulation disappears. Failing to adjust these parameters results in incorrect redistribution of feature maps, and worse

CNN accuracies. The proposal only requires the scaling of the stored parameters and significantly improves the performance of approximate multipliers on deep neural networks. It does not introduce any new operations and does not prevent the ability of batch normalization to fold into neighboring layers.

Designing an approximate multiplier with an unbiased mean error near zero is another effective solution, but it is much harder to make changes to hardware designs. The unbiased designs usually have a small amount of mean error because it is difficult to create a perfectly unbiased design, and the problem is only deferred to deeper networks. Also, depending on the approximation method, it may take additional hardware resources to make a design unbiased. The networks that do not use batch normalization have no choice but to use the unbiased multipliers, but otherwise the proposed adjustment is simpler, less costly, and more flexible to accommodate different approximation methods with biased mean errors.

# 5 Arithmetic Reason for Bfloat16 Success

The discoveries in Sections 3 and 4 are not limited to the error of approximate multiplication but apply to all sources of arithmetic error. They also provide deeper understanding of why bfloat16 [22] has been widely successful at accelerating CNNs despite its reduced precision. The bfloat16 format is an approximation of the FP32 floating-point format that simply truncates the 16 least significant bits from the 23 fractional bits. By truncating the less significant fractional bits, converting an FP32 value to bfloat16 generates a small negative error from 0% to -0.78% relative to the original FP32 value. The factors discussed in Section 3 also minimize the adverse effects of this varying error and they explain why using the full FP32 accumulator after bfloat16 multiplication produces the best results [25], in agreement with the observation that the accumulations need to be exact. The accumulation of mean error discussed in Section 4 should also be present, but the mean error of bfloat16 is too small to cause any problems for the studied CNNs.

The successful application of bfloat16 to CNNs has been explained by the high-level interpretation that the small amount of error helps the regularization of a CNN model. The interpretation is still valid and also applies to approximate multiplication, and the findings of this paper provide deeper understanding with the arithmetic explanation. They also explain why the bfloat16 format has slightly degraded performances with the networks that use grouped convolution as presented in Section 6.2.

# 6 Experiments

## 6.1 Setup

The experiments are performed in the Caffe framework to evaluate the impact of approximate multipliers on deep CNN models [26]. Caffe has limited features compared to contemporary tools but its lack of encapsulation allows easy modification of underlying matrix multiplication, making it suitable for the study. The code that performs floating-point matrix multiplication in GPU is replaced by the CUDA C++ functions that emulate the behavior of the target approximate multipliers. These functions are verified against RTL simulations of the HDL code of the multipliers.

TABLE 1: Pre-trained CNN models used for the experiments

| Network | Model Source | BatchNorm | Grouped Conv. |
|---|---|---|---|
| **VGG16** | [26] | | |
| **GoogLeNet** | [26] | | |
| **ResNet-50** | [27] | √ | |
| **ResNet-101** | [27] | √ | |
| **ResNet-152** | [27] | √ | |
| **Inception-v4** | [28] | √ | |
| **Inception-ResNet-v2** | [29] | √ | |
| **ResNeXt-50-32x4d** | [30] | √ | √ |
| **Xception** | [28] | √ | √ |
| **MobileNetV2** | [31] | √ | √ |

The Mitch-$w$6 multiplier with the C1 sign handling is chosen because the comparison against the other multipliers showed that it was cost-efficient while performing well on AlexNet [9]. Mitch-$w$ multipliers consume significantly less resources compared to the Mitchell log multiplier. DRUM6 multiplier [19] is also added to the experiments because it performed very well on AlexNet while being more costly than Mitch-$w$6 [9]. The truncated iterative log multiplier in [20] has higher accuracy than these multipliers and is tested for networks that have depthwise separable convolution. Unlike Mitch-$w$, DRUM6 and the truncated iterative log multiplier have the unbiased mean errors close to zero. The FP32 floating-point results are included for comparison, and the bfloat16 results provide additional data points (see Section 5).

The target application is object classification with the ImageNet ILSVRC2012 validation dataset of 50,000 images. Only single crops are used for experiments because the C++ emulation of the approximate multipliers is very time-consuming compared to the multiplication performed in actual hardware, so the presented CNN accuracies may differ from the original literature that use 10-crops. Table 1 shows the list of CNN models used for the experiments, and the networks that use batch normalization and grouped convolutions are marked for comparative discussion. The pre-trained CNN models for the experiments are publicly available from online repositories, and the source is indicated with each model. Any training or retraining of a network model is purposefully avoided to achieve reproducibility and to show that the proposed methodology works with many network models with only minor scaling of batch normalization parameters.

The experiments assume quantization to 32 fixed-point bits without rounding (statically assigned to 16 integer and 16 fractional bits) as it is sufficient for all the tested network models. As discussed in Section 2, approximate multiplication is an orthogonal approach to quantization and we used generous quantization to minimize the quantization errors and study the effects of approximate multiplication in isolation, in order to clearly evaluate the hypothesis presented in this paper. This paper focuses on establishing approximate multiplication as a viable approach, and combining various

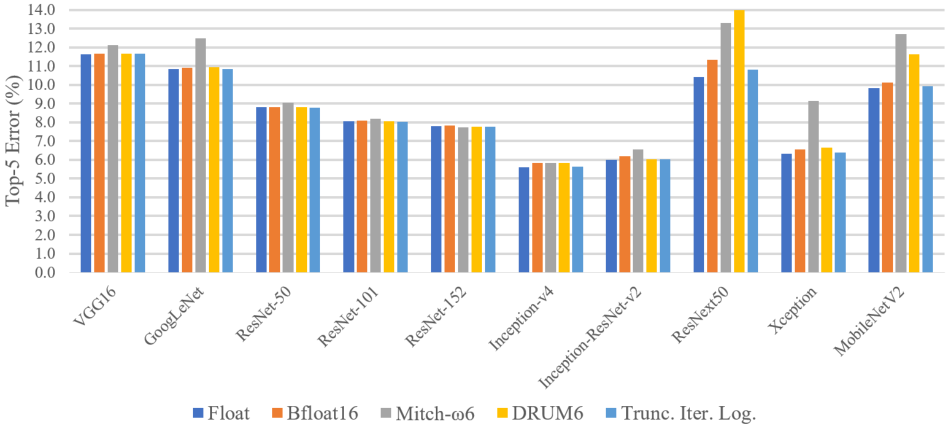


Fig. 7: Comparison of Top-5 errors between the FP32 reference and the approximate multipliers.

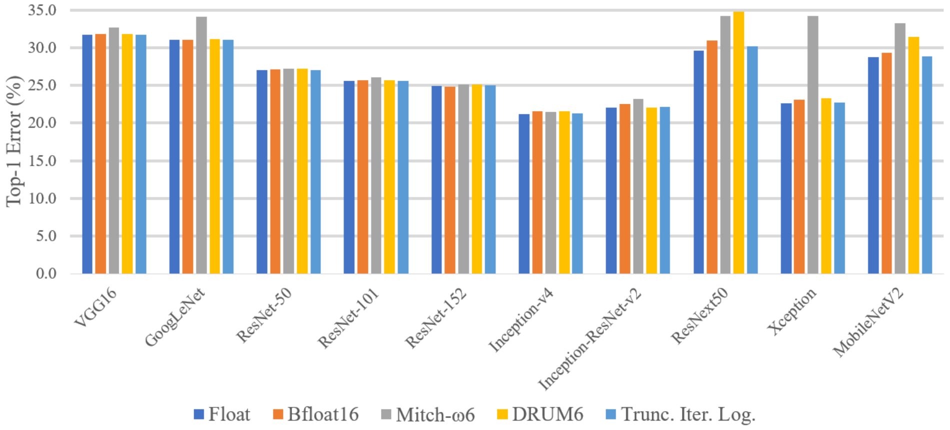


Fig. 8: Comparison of Top-1 errors between the FP32 reference and the approximate multipliers.

quantization methods with approximate multiplication is beyond the scope of this paper.

### 6.2 Impact of Approximate Multiplication on CNNs

Fig. 7 and 8 show the Top-5 and Top-1 errors when the approximate multipliers are applied to the CNNs, compared against the FP32 reference values. For the networks with conventional convolution, the studied approximate multipliers produce predictions that are nearly as accurate as the exact FP32 floating-point as they show Top-5 errors within 0.2% compared to the reference values, except for Mitch-$w6$ on Inception-ResNet-v2 (0.5%) and the networks without batch normalization. On the contrary, the CNNs with grouped convolution suffer degraded accuracies when there are errors in multiplications, from approximate multiplication as well as bfloat16. The difference of CNN accuracies between different convolution types supports the hypothesis presented in Section 3.

In order to demonstrate the increased variance of error for grouped and depthwise convolution, all convolution outputs are extracted for the first 100 sample images of the ILSVRC2012 validation set with FP32 and Mitch-$w6$ multiplications. The errors from approximate multiplication are measured by comparing the results. The variance of accumulated error within each channel is measured as well as the variance between the convolution outputs. The geometric means are taken across all channels as channels had wildly varying ranges of values. Table 2 shows the measured values for various CNNs and it demonstrates the increased variance of accumulated error for grouped and depthwise convolutions as discussed in Section 3.4. The conventional convolution results also provide the evidence that the accumulated errors have much less variance compared to the distribution of outputs, and therefore have less impact on the functionality of feature detection.

While the 100 images may seem like a small number of samples, the geometric means are actually taken across millions of convolution feature maps produced from the images. The samples include sufficient numbers of data points to demonstrate the point. It is extremely difficult to process the entire dataset because of the large amount of internal data generated by CNNs. Changing the sample size had little effect on the observation and the samples likely represent the behavior of the entire set for these models.

The measured variances in Table 2 do not directly correlate to the performance of Mitch-$w6$ in Fig. 7 and 8 because Table 2 only shows the error variance within each channel and does not account for the error variance across chan-

TABLE 2: Measured error variance with Mitch-$w6$

| Conv. Type | Network | Error Vari. | Output Vari. | Pct. |
|---|---|---|---|---|
| **Conventional** | **ResNet-50** | 2.31E-3 | 6.13E-2 | 3.8% |
| | **ResNet-101** | 1.69E-3 | 3.52E-2 | 4.8% |
| | **ResNet-152** | 1.50E-3 | 2.72E-2 | 5.5% |
| | **Inception-v4** | 6.79E-3 | 1.22E-1 | 5.6% |
| | **Inception-ResNet-v2** | 1.18E-3 | 1.85E-2 | 6.3% |
| **Grouped** | **ResNeXt-50-32x4d** | 1.50E-4 | 1.35E-3 | 11.2% |
| **Depthwise** | **Xception** | 1.81E-2 | 8.91E-2 | 20.4% |
| | **MobileNetV2** | 2.00E-2 | 1.34E-1 | 14.9% |

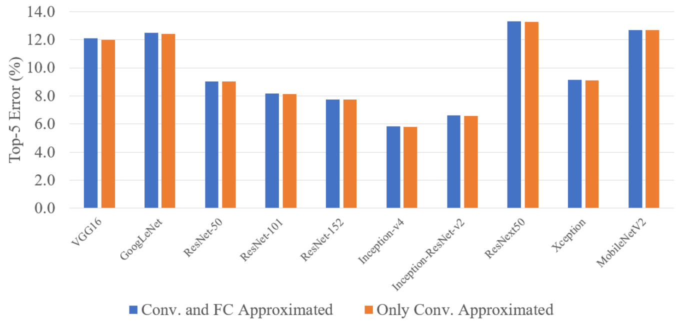


Fig. 9: Low impact on CNN accuracies when FC layers do not use approximate multiplication. The experiments are performed with Mitch-$w6$.

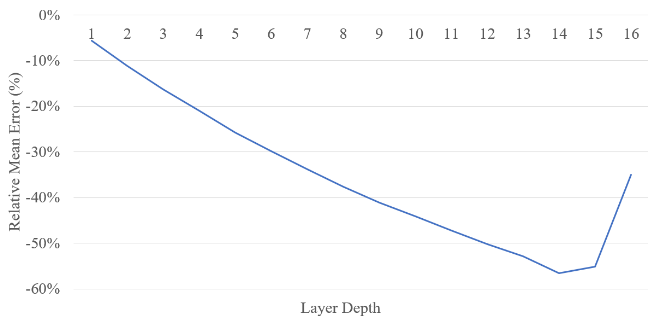


Fig. 10: Accumulation of mean error on VGG16.

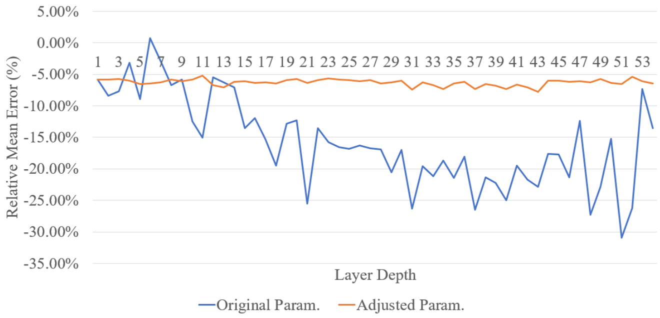


Fig. 11: Effect of batch normalization on ResNet-50.

nels. The approximate multiplication in ResNeXt-50-32x4d causes more degradation in the prediction accuracy because ResNeXt networks have many branches in their architectures where different amounts of error accumulate. The Inception networks have relatively shorter branches and show slightly more degradation compared to the ResNet models that have none. The theoretical principle discussed in Section 3.3 agrees with this analysis, though Table 2 could not capture these differences.

When the convergence of errors diminishes for grouped and depthwise convolutions, the outcomes become statistically uncertain and each CNN model may favor different approximate multipliers depending on their error patterns. DRUM6 has a different error pattern compared to Mitch-$w6$ and it performs worse than Mitch-$w6$ on the ResNeXt50 model despite the fact that it generally produces smaller errors, as shown in Fig. 7 and 8. On the contrary, DRUM6 performs very well on the Xception model and it is conjectured that the errors from DRUM6 work well with this particular pre-trained model.

For CNNs with grouped convolutions, a sufficiently accurate approximate multiplier can still be used to perform accurate inferences, as demonstrated with the truncated iterative log multiplier in Fig. 7 and 8. When the converging effect of accumulation is reduced, the variance of accumulated error may be reduced by producing a smaller range of errors at the cost of more hardware resources.

Fig. 9 shows the effects on CNN accuracies when the FC layers perform exact multiplication instead of approximate multiplication. Despite the fact that approximating later layers in CNNs have more influence on the outputs compared to earlier layers [14], [23], Fig. 9 demonstrates that approximating FC layers at the end of CNNs has minimal impact on CNN accuracies. The FC layers have a large number of accumulations per each output and the higher convergence of error preserves the relative order between the final outputs. This is the desirable property of approximate multiplication for CNN inferences as discussed in Section 3.3.

### 6.3 Effect of Batch Normalization

Fig. 10 demonstrates the accumulation of mean error in VGG16 with Mitch-$w6$, averaged over the 100 sample images. Because the network lacks batch normalization, the deeper layers receive the inputs that are repeatedly scaled down when the errors in multiplication are biased. It explains the poor performance of Mitch-$w6$ on VGG16 and GoogLeNet in Fig. 7, while the unbiased DRUM6 performs well. The last three layers that disrupt the trend are the FC layers where the added bias values become more significant when the inputs have reduced magnitudes.

Fig. 11 shows the effect of batch normalization with properly adjusted parameters, on ResNet-50 with Mitch-$w6$ averaged over the 100 sample images. For Mitch-$w6$ with a mean error of -5.9%, the mean and variance parameters in batch normalization are scaled by 0.941 and 0.885 respectively. With the proper adjustments, batch normalization eliminates the accumulation of mean error across layers and helps approximate multiplication work with deep CNNs. Fig. 11 shows that the mean error per layer hovers around the mean error of Mitch-$w6$, which supports the convergence of accumulated error as well as the effectiveness

TABLE 3: Impact of batch normalization adjustment with Mitch-$w6$ on ResNet models

| | Top-1 Error | | Top-5 Error | |
|---|---|---|---|---|
| | Original | Adjusted | Original | Adjusted |
| **ResNet-50** | 31.7% | 27.2% | 10.5% | 9.0% |
| **ResNet-101** | 31.8% | 26.0% | 12.0% | 8.2% |
| **ResNet-152** | 31.2% | 25.2% | 11.5% | 7.7% |

of the adjusted batch normalization. Failing to adjust the parameters not only accumulates error in deeper layers but also becomes an additional source of error with incorrect redistribution of feature maps, resulting in an unstable pattern of accumulated error. Table 3 shows the impact on the Top-1 and Top-5 errors of the ResNet models. Incorrect batch normalization results in performance degradation while the corrected batch normalization layers help approximate multiplication perform well for deep ResNet models.

## 7 Comparison of Costs

Using the bfloat16 format significantly reduces the hardware costs compared to the FP32 floating-point format and has been widely adopted in Machine Learning hardware accelerators. While its ease of use and the ability to perform training as well as inference are undeniably advantageous, its arithmetic units are slower and consume more energy compared to the discussed multipliers based on the fixed-point format. It is plausible to have a use-case scenario where embedded systems perform only inferences under strict design constraints, while communicating to datacenters where training occurs. This section presents a brief comparison of the hardware costs against a bfloat16 MAC unit to give an idea of the potential benefits of the approximate log multiplication.

Table 4 compares the costs among the MAC units of FP32, bfloat16 and the Mitch-$w$, as synthesized with a 32nm standard library from Synopsys. The Mitch-$w6$ HDL code is available in [32], the FP32 MAC design is from [33], and we modified the FP32 design to create the bfloat16 MAC. Synopsys Design Compiler automatically synthesized the fixed-point MAC, and Mitch-$w6$ is followed by an exact fixed-point adder. The 32-bit Mitch-$w6$ design represents the circuit used for the experiments while the 16-bit design represents what is potentially achievable with the proper quantization such as [18]. It is clear from Table 4 that applying approximate multiplication to CNNs can save a significant amount of resources for inferences.

The presented figures do not consider the potential benefits when adopting multiple log multipliers, where additional optimization for resource sharing can be performed depending on the design of the hardware accelerator. Oliveira et al. [34] proposed that certain parts of the log multiplier can be removed or shared between multiple instances of MAC units depending on the accelerator design.

## 8 Related Works

There have been a number of previous works that applied approximate multipliers to CNN inferences. This paper explains the underlying reason why some of these methods perform well despite the error and how to extend the methodologies to deep CNNs with batch normalization. To the best of our knowledge, this is the first work to demonstrate that one approximate multiplier design can perform successful inferences on the various ResNet and Inception network models without retraining.

TABLE 4: Hardware costs of FP32, bfloat16, fixed-point and Mitch-$w6$ MAC units

| | N=16 | | | N=32 | | |
|---|---|---|---|---|---|---|
| | bfloat16 | Fixed | Mitch-$w6$ | FP32 | Fixed | Mitch-$w6$ |
| **Delay (ns)** | 4.77 | 2.07 | 2.74 | 7.52 | 4.29 | 4.39 |
| **Power (mW)** | 1.47 | 1.17 | 0.50 | 5.80 | 4.36 | 0.98 |
| **Energy (pJ)** | 7.01 | 2.42 | 1.37 | 43.62 | 18.70 | 4.30 |
| **Energy vs. bfloat16** | 100% | 35% | 20% | 622% | 267% | 61% |

One study in [35] applied various approximate multipliers with varying accuracies to the VGG network, and it provided more evidence that approximate multiplication was compatible with CNN inferences. Their work included interesting experimental results that support our hypothesis. They found that approximating the convolution layers with higher numbers of channels resulted in less degradation of CNN accuracy, and this agrees with our finding that variance of accumulated error decreases with more inter-channel accumulations.

The works presented in [10], [13], [21], [36], [37] had used logic minimization to create the optimal approximate multipliers for each network model. Logic minimization intentionally flips bits in the logic to reduce the size of the operators, and these techniques use heuristics to find the optimal targets. While these studies demonstrate promising results for improving the efficiency of CNN inferences, the heuristics involve the costly exploration of a large design space and do not ensure that the optimal multipliers for one situation would be optimal for another.

The Alphabet Set Multiplier proposed in [14] stores multiples of each multiplier value as alphabets and combines these alphabets to produce the products. Because the stored multiples require memory accesses, the authors eventually proposed the design with a single alphabet that had performed reasonably well for the simple datasets. However, the design was too inaccurate to handle the more complex dataset of ImageNet [9].

Approximate log multiplication from Mitchell's Algorithm had been applied to small CNN models in [9], [38], [39]. The iterative log multipliers that increase accuracy by iterating log multiplication had been also studied [5], [20], [40]. They were mostly effective at performing CNN inferences but the reason for the good performances largely remained unsolved. This paper provides deeper understanding of the effects of approximate multiplication on CNNs.

The log multipliers should be distinguished from the log quantization presented in [41], [42]. The log quantization performs all operations in the log domain and suffers from inaccurate additions, which may explain why the performances drop for more complex networks. The Mitchell's Algorithm still performs exact additions in the fixed-point format which helps maintain the CNN performance, as discussed in Section 3.

There are many other ways of approximating multiplication that had not been applied to deep CNNs, such as [43], [44], [45] among countless others. While we believe that the studied multiplier designs are the most promising, there are most likely other related opportunities for improving CNNs.

## 9 Conclusion

This paper provides a detailed explanation of why CNNs are resilient against the errors in multiplication. Approximate multiplication favors the wide convolution layers with many input channels and batch normalization can be adjusted for deeper networks, making it a promising approach as the networks become wider and deeper to handle various real-world applications. The proposed approximate multipliers show promising results for CNN architectures, and the arithmetic explanations provide a new and effective way for designing hardware accelerators. They also help explain some of the phenomenon observed in the related works while providing guidelines for extending to deeper CNNs with batch normalization.

The most widely applicable insight of this paper is that the multiplications in CNNs can be approximated while the additions have to be accurate. The implications are far-reaching and may help analyze and justify a variety of other approximation techniques that were previously only supported by empirical evidence. In this paper, we provide the arithmetic reason behind the success of bfloat16 approximation [22] and also conjecture that log quantization [42] loses CNN accuracy because of inaccurate additions. For quantization, the convergence theory can justify the reduced number of bits used for weights while accumulations are done with a higher number of bits. The findings may help justify the analog processing of neural networks where the multiplication resistors may have some process variation [4]. These are only a few examples and new approximation techniques may be evaluated in the similar fashion in terms of the variance of accumulated error. Various studies on approximation of CNN inferences have relied only on the end results as the inner workings of CNNs are often treated as black boxes. This paper seeks to contribute towards a more analytical understanding of CNN approximation based on arithmetic.

## Acknowledgments

This work has been partially supported by the CPCC at UCI, the Community of Madrid under grant S2018/TCS-4423, the EU (FEDER) and the Spanish MINECO under grant RTI2018-093684-B-I00, and the NRF of South Korea funded by the Ministry of Education (2017R1D1A1B03030348).

## References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[2] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[3] X. Sun, X. Peng, P.-Y. Chen, R. Liu, J.-s. Seo, and S. Yu, “Fully parallel rram synaptic array for implementing binary neural network with (+ 1,- 1) weights and (+ 1, 0) neurons,” in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 574–579.

[4] Y. Shim, A. Sengupta, and K. Roy, “Low-power approximate convolution computing unit with domain-wall motion based “spin-memristor” for image processing applications,” in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[5] J. Kung, D. Kim, and S. Mukhopadhyay, “A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses,” in *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*. IEEE, 2015, pp. 85–90.

[6] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, “Approxann: An approximate computing framework for artificial neural network,” in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 701–706.

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[8] L. Lai, N. Suda, and V. Chandra, “Deep convolutional neural network inference with floating-point weights and fixed-point activations,” *arXiv preprint arXiv:1703.03073*, 2017.

[9] M. S. Kim, A. A. Del Barrio, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, “Efficient Mitchell’s approximate log multipliers for convolutional neural networks,” *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 660–675, 2018.

[10] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, “Design of power-efficient approximate multipliers for approximate artificial neural networks,” in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–7.

[11] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, “Going deeper with embedded fpga platform for convolutional neural network,” in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[12] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, “Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency,” in *Design Automation Conference*. IEEE, 2010, pp. 555–560.

[13] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu, “Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators,” in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 201–206.

[14] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, “Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing,” in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 145–150.

[15] J. N. Mitchell, “Computer multiplication and division using binary logarithms,” *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, Aug 1962.

[16] D. Lin, S. Talathi, and S. Annapureddy, “Fixed point quantization of deep convolutional networks,” in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[17] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, “Stripes: Bit-serial deep neural network computing,” in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. 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*, 2018, pp. 2704–2713.

[19] S. Hashemi, R. I. Bahar, and S. Reda, “Drum: A dynamic range unbiased multiplier for approximate applications,” in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 418–425.

[20] H. Kim, M. S. Kim, A. A. Del Barrio, and N. Bagherzadeh, “A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks,” in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 108–111.

[21] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, “Improving the accuracy and hardware efficiency of

neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2019.

[22] S. Wang and P. Kanwar, "Bfloat16: the secret to high performance on cloud tpus," *Google Cloud Blog, August*, 2019.

[23] D. Kim, J. Kung, and S. Mukhopadhyay, "A power-aware digital multilayer perceptron accelerator with on-chip training based on approximate computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 2, pp. 164–178, 2017.

[24] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[25] G. Henry, P. T. P. Tang, and A. Heinecke, "Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 69–76.

[26] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, 2014, pp. 675–678.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[28] Y. Liu, C. Chen, R. Zhang, T. Qin, X. Ji, H. Lin, and M. Yang, "Enhancing the interoperability between deep learning frameworks by model conversion," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1320–1330.

[29] N. Silberman, "Tf-slim: A lightweight library for defining, training and evaluating complex models in tensorflow," 2017.

[30] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[32] A. Del Barrio, M. S. Kim, R. Hermida, and N. Bagherzadeh, "log-arithmetic," 2019. [Online]. Available: https://github.com/albertodbg/log-arithmetic

[33] A. A. Del Barrio, N. Bagherzadeh, and R. Hermida, "Ultra-low-power adder stage design for exascale floating point units," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, pp. 1–24, 2014.

[34] L. T. Oliveira, M. S. Kim, A. A. Del Barrio, N. Bagherzadeh, and R. Menotti, "Design of power-efficient fpga convolutional cores with approximate log multiplier," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2019, pp. 203–208.

[35] I. Hammad and K. El-Sankary, "Impact of approximate multipliers on vgg deep learning network," *IEEE Access*, vol. 6, pp. 60 438–60 444, 2018.

[36] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "Alwann: Automatic layer-wise approximation of deep neural network accelerators without retraining," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[37] S. De, J. Huisken, and H. Corporaal, "Designing energy efficient approximate multipliers for neural acceleration," in *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, 2018, pp. 288–295.

[38] M. S. Kim, A. A. D. Barrio, R. Hermida, and N. Bagherzadeh, "Low-power implementation of mitchellś approximate logarithmic multiplication for convolutional neural networks," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, 2018, pp. 617–622.

[39] M. S. Ansari, B. F. Cockburn, and J. Han, "An improved logarithmic multiplier for energy-efficient neural computing," *IEEE Transactions on Computers*, 2020.

[40] U. Lotrič and P. Bulić, "Applicability of approximate multipliers in hardware neural networks," *Neurocomputing*, vol. 96, pp. 57–65, 2012.

[41] 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)*. IEEE, 2017, pp. 5900–5904.

[42] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *arXiv preprint arXiv:1603.01025*, 2016.

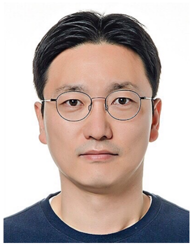
[43] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Transactions on Circuits and Systems*, vol. 65, no. 9, pp. 2856–2868, 2018.

[44] S. Salamat, M. Imani, S. Gupta, and T. Rosing, "Rnsnet: In-memory neural network acceleration using residue number system," in *2018 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2018, pp. 1–12.

[45] M. Imani, M. Masich, D. Peroni, P. Wang, and T. Rosing, "Canna: Neural network acceleration using configurable approximation on gpgpu," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 682–689.

**Min Soo Kim** received the BA.Sc degree in Engineering Science from the University of Toronto in 2008, and the M.S. and Ph.D. degrees in Computer Engineering from the University of California, Irvine, in 2011 and 2020 respectively. He currently works at NGD Systems as an AI Software Engineer, and his research interests include computational storage and hardware acceleration of convolutional neural networks.

**Alberto A. Del Barrio** received the Ph.D. degree in Computer Science from the Complutense University of Madrid (UCM), Madrid, Spain, in 2011. Since 2020, he is an Associate Professor of Computer Science with the Department of Computer Architecture and System Engineering, UCM. His research interests include Design Automation, Arithmetic as well as Video Coding Optimizations.



**HyunJin Kim** is an associate professor in the School of Electronics and Electrical Engineering at Dankook University, Republic of Korea. He received a Ph.D in Electronics and Electrical Engineering (2010) from Yonsei University. He worked as a mixed-signal VLSI circuit designer at Samsung Electromechanics (2002∼2005), and as a senior engineer in a flash memory controller project at Samsung Electronics (2010∼2011). His current research interests include approximate & stochastic computing for neural network implementation methodology, string matching engines, and energy-aware embedded systems.

**Nader Bagherzadeh** is a professor of computer engineering in the Department of Electrical Engineering and Computer Science at the University of California, Irvine, where he served as a chair from 1998 to 2003. Dr. Bagherzadeh has been involved in research and development in the areas of: computer architecture, reconfigurable computing, VLSI chip design, network-on-chip, 3D chips, sensor networks, computer graphics, memory and embedded systems, since he received a Ph.D. degree from the University of Texas at Austin in 1987. He is a Fellow of the IEEE.