

# The Hidden Power of Pure 16-bit Floating-Point Neural Networks

Juyoung Yun<sup>1</sup> Byungkon Kang<sup>2</sup> Zhoulai Fu<sup>2</sup>

## Abstract

Lowering the precision of neural networks from the prevalent 32-bit precision has long been considered harmful to performance, despite the gain in space and time. Many works propose various techniques to implement half-precision neural networks, but none study *pure* 16-bit settings. This paper investigates the unexpected performance gain of pure 16-bit neural networks over the 32-bit networks in classification tasks. We present extensive experimental results that favorably compare various 16-bit neural networks' performance to those of the 32-bit models. In addition, a theoretical analysis of the efficiency of 16-bit models is provided, which is coupled with empirical evidence to back it up. Finally, we discuss situations in which low-precision training is indeed detrimental.

## 1. Introduction

Today's ubiquitous need for neural network techniques — from autonomous vehicle driving, healthcare, and finance to general artificial intelligence and engineering — has become a faith of fact for many. Significant computing power can be necessary for training neural networks on real-world data, which has stimulated the semiconductor industry to pursue cutting-edge chips and GPU solutions for reduced-precision floating-point arithmetic.

Reduced precision provides remarkable performance gain in speed, memory usage, and energy consumption over traditional CPU-based single and double-precision computing. Many GPUs are now powered with reduced-precision floating-point. In particular, widely accessible NVIDIA GPUs support the IEEE-standardized half-precision, namely, the 16-bit floating-point format.

However, reduced precision alone is also known to cause

accuracy loss. In the IEEE 16-bit format, positive numbers except subnormal ones lie between  $5.96\text{E-}8$  and  $6.5\text{E}04$ , and thus a straightforward rounding or truncation of single or double precision data into half precision can cause overflow, underflow, or subnormal numbers, all of which affects numerical accuracy.

Consider the sigmoid function,  $\text{sigmoid}(x) = 1/(1 + e^{-x})$ . It overflows if  $x$  is small and underflows if  $x$  is large, where both overflow and underflow occur in  $e^{-x}$ , and the error will then be propagated to the function's output. As a quick experiment, one can set a vector  $x$  to be all the 16-bit floating-point numbers (there are 63487 in total) excluding  $\pm\infty$  and NaN, and  $y$  be all the upcast 32bit float ( $y = \text{np.float32}(x)$ ), and element-wise compare  $\text{sigmoid}(x)$  and  $\text{sigmoid}(y)$ . They have a relative error of  $4.85\text{E-}02$  and an absolute error of  $7.09\text{E-}05$  on average.

Errors of such scale can be consequential for general scientific computing, e.g., causing the failure of the Patriot missile system (Skeel, 1992). The question that would naturally arise for an ML practitioner would be: Are floating-point errors of such scale too significant for a reduced precision model in machine learning to make the right prediction?

In the machine learning community, many believe that "deep learning models ... are "very tolerant of reduced-precision computations" (Dean, 2020). ML researchers have actively investigated a wide range of techniques that lower the precision of the floating point numbers used in neural networks but still maintain accuracy. For example, Micikevicius et al. (2018) propose a mixed precision technique, where weight, activation, and gradients are stored in 16 bits, but weight updates are carried out on 32 bits. Kalamkar et al. (2019) uses another mixed precision technique with 32-bit floating-point and BFloat, Google Brain's half-precision format, in which a tensor modification method is used to zero out the lower 16 bits of the 32-bit data flow.

All these algorithms use mixed precision, where two or more precision are chosen from a small number of available precisions, typically half (16-bit) and single (32-bit). To the best of our knowledge, the *pure* 16-bit neural network is rarely used as a standalone solution in the machine learning community. This work targets the 16-bit pure neural network and studies whether we can use it out of the box, without parameter tuning or techniques commonly used in

<sup>1</sup>Department of Computer Science, Stony Brook University, NY, USA <sup>2</sup>Department of Computer Science, State University of New York Korea, Incheon, Republic of Korea. Correspondence to: Byungkon Kang <byungkon.kang@sunykorea.ac.kr>, Zhoulai Fu <zhoulai.fu@sunykorea.ac.kr>.

mixed precision algorithms like loss scaling. Our finding is positive. That is, pure 16-bit neural networks, without any floating-point 32 components, despite being imprecise by nature, can be precise enough to handle a major application of machine learning – the classification problem.

Our work first formalizes the intuitive concept of error tolerance and proposes a lemma that theoretically guarantees 16-bit “locally” achieves the same classification result as 32-bit under certain conditions. Combined with our preliminary observations, we conjecture that the 16-bit and 32-bit have close results in handling classification problems. We then validate our conjecture through extensive experiments on Deep neural network (DNN) and Constitutional Neural Network (CNN) problems. Our contributions follow:

- We aim to debunk the myth that plain 16-bit models do not work well. We demonstrate that training neural network models in pure 16 bits with no additional measures to “compensate” results in competitive, if not superior, accuracy.
- We offer theoretical insights on *why* half precision models work well, as well as empirical evidence that supports our analysis.
- We perform extensive experiments comparing the performance of various *pure* 16-bit neural networks against that of 32-bit and mixed precision networks and find that the 16-bit neural networks can perform as well as their 32-bit counterparts. We also identify factors that could negatively influence the success of 16-bit models.

## 2. Related Work

Several techniques have been proposed to reduce the precision of machine learning models while maintaining their accuracy to some degree. The approach that best aligns with our work is that of lowering the precision of the floating point numbers used in those models, but other approaches involve algorithm modification, fixed point formats, and hardware acceleration.

**Algorithm modification.** These approaches aim to modify certain components of the main algorithm to allow low-precision work. The work by (De Sa et al., 2017) offers a way to address the issues associated with low precision in gradient descent by assigning different precisions to the weights and the gradients. Such a scheme is assisted by special hardware implementation to speed up this mixed-precision process. A similar approach is taken by (Bjorck et al., 2021) in reinforcement learning. This work proposes various mechanisms to perform reinforcement learning on low precision reliably. For example, adopting numerical methods to improve the Adam optimization algorithm so

that underflow can be prevented. Many of such techniques proposed are somewhat ad-hoc to the target problem but might also be useful in a more general setting.

**Fixed point formats.** A *fixed point format* is another number format often used to represent real numbers, although not standard. Several other works have adopted fixed point formats to allow low precision due to the intuitive representation and high extensibility to other configurations. To verify the issues with low precision in deep learning, the authors of (Gupta et al., 2015) investigate the behavior of deep neural networks when the precision is lowered. In this work, the empirical results show that reducing the precision of the real numbers comprising the weights of the network shows a graceful degradation of accuracy. (Chen et al., 2017) performs neural network training in fixed point numbers by quantizing a floating point number into a low-precision fixed point number during training. The idea of quantizing floating point is also used in (Lin et al., 2016), where the authors approach the conversion as an optimization problem. The objective of the optimization is to reduce the network’s size by adopting different bit-width for each layer. On a more system-based approach, (Kumar et al., 2020) proposes compiler support for converting floating point numbers to low-precision fixed point numbers. (Gopinath et al., 2019) also takes compiler- and language-based support for achieving low-precision numbers.

Although not entirely the same as *real* number fixed point, integer quantization can also be considered a special fixed point format and has become a viable choice in low-precision neural network design. In (Wu et al., 2018), floating point weights are quantized to convert them into signed integer representations. The authors discover that adopting an integral quantization technique results in a regularization-like behavior, leading to increased accuracy. Similar to this approach, work by (Das et al., 2018) also proposes to use integer operations to achieve high accuracy. Such integer operations effectively convert the floating point numbers into what is known as dynamic fixed point (DFP) format. The work in (Jacob et al., 2017) also aims to quantize the weights to integers. However, the weights remain integers only during the forward pass and become floating point numbers for back-propagation to account for minute updates. Another work (Courbariaux et al., 2015) takes an extreme quantization approach to binarize the weights to -1 and 1. The weights remain binary during the forward and backward pass but become floating points during the weight update phase. (Xiao et al., 2022) adopts a post-training quantization approach to reduce the memory footprint of large-scale language models.

(Köster et al., 2017) proposes an adaptive numerical format that retains the advantages of floating and fixed point numbers. This is achieved by having a shared exponent that gets updated during training.

**Mixed precision.** On the other hand, *mixed precision* ap-

proaches maintain standard floating point numbers in two different precision. The first practically successful reduced-precision floating point mechanism was proposed by (Micikevicius et al., 2018). In this work, the authors devise a scheme to perform mixed precision training by maintaining a set of master full-precision floating point weights that serves as the ‘original copy’ of the half-precision counterparts. While this technique does result in reduced running time, the mixed-precision nature of it limits the performance gain achieved. (Wang et al., 2019) propose a 4-bit floating point architecture mixed with a small amount of 8-bit precision. In addition to the format, the authors devise a two-phase rounding procedure to counter the low accuracy induced by the low-precision format.

**Hardware support.** Lastly, we would like to point out that many of these works either implicitly or explicitly require hardware support due to the individual floating/fixed point formats. Most approaches typically use FPGAs and FPU to implement these formats, but other works such as (Sharma et al., 2017) propose novel architectures tailored to addressing the bit formats of the floating point numbers. Some previously mentioned works, such as (Wang et al., 2019), also hint at the possibility of leveraging hardware assistance.

Unlike these previous works, our work focuses explicitly on the IEEE floating point format, which is the de-facto standard for general computing machinery. More precisely, we investigate the pros and cons of using a pure 16-bit IEEE floating point format in training neural networks without external support.

### 3. Background

**General Notation** The real numbers and integers are denoted by  $\mathbf{R}$  and  $\mathbf{Z}$ , respectively. Given a vector  $x = (x_0, \dots, x_{n-1}) \in \mathbf{R}^n$ , its infinity norm or maximum norm, denoted by  $|x|_\infty$ , is the maximum element in the vector, namely  $|x|_\infty \stackrel{\text{def}}{=} \max_i |x_i|$ .

#### 3.1. Floating-Point Representation

We write  $\mathbf{F}_{16}$  to denote the set of 16-bit floating-point numbers excluding  $\pm\infty$  and NaN (Not-a-Number). Following IEEE-754 standard (IEEE Computer Society, 2008), each  $x \in \mathbf{F}_{16}$  can be written as

$$x = (-1)^s \times g_0.g_1\dots g_{10} \text{ (2)} \times 2^e \quad (1)$$

where  $s \in \{0, 1\}$ ,  $g_i \in \{0, 1\}$  ( $0 \leq i \leq 10$ ), and  $e \in \mathbf{Z}$ . We call  $s$ ,  $g_0.g_1\dots g_{10}$  and  $e$  the sign, the significand, and the exponent, respectively. They satisfy  $g_0 \neq 0$  and  $-14 \leq e \leq 15$ . The case where  $g_0 = 0$  and  $e = -14$  is called a subnormal number.

We write  $\mathbf{F}_{32}$  for the set of 32-bit (single-precision) floating-

point numbers excluding  $\pm\infty$  and NaN. The following property holds:

$$\mathbf{F}_{16} \subset \mathbf{F}_{32} \subset \mathbf{R} \quad (2)$$

Namely, a 16-bit or 32-bit floating-point number is a real, and a 16-bit can be exactly represented as a 32-bit (by padding with zeros). Tab. 3.1 lists the range and the precision of 16-bit and 32-bit floating-point numbers.

Table 1. Some characteristics of the 16-bit and 32-bit floating-point formats.

Type	Size	Range	Machine-epsilon
Half	16 bits	6.55E±4	4.88E-04
Single	32 bits	3.4E±38	5.96E-08

#### 3.2. Floating-point Errors

Rounding is necessary when representing real numbers that cannot be written as Eq. 1. *Rounding error* of a real  $x$  at precision  $p$  refers to  $|x - \circ_p(x)|$  where the rounding operation,  $\circ_p : \mathbf{R} \rightarrow \mathbf{F}_p$ , defines the nearest floating-point number of  $x$ . Namely,  $\circ_p(x) \stackrel{\text{def}}{=} \operatorname{argmin}_{y \in \mathbf{F}_p} |x - y|$ .<sup>1</sup> Rounding error is usually small, on the order of machine epsilon (Tab. 3.1), but it can be propagated and become more significant. For example, the floating-point code `sin(0.1)` goes through three approximations. First, 0.1 is rounded to the floating-point  $\circ_p(0.1)$  for some precision  $p$ . Then, the rounding error is *propagated* by the floating-point code `sin`. Lastly, the calculation output is rounded again if an exact representation is not possible.

Floating-point errors are usually measured in *absolute error* or *relative error*. This paper focuses on classification problems where output numbers are probabilities between 0 and 1. Thus, we use the absolute error  $|x - y|$  for quantifying the difference between two floating point numbers  $x$  and  $y$ .

### 4. Theory

Suppose  $M_{16}$  and  $M_{32}$  are 16-bit and 32-bit deep learning models trained by the same neural network architecture and hyper-parameters. By abuse of notation, we consider  $M_{16}$  and  $M_{32}$  as classifiers or functions that return the probability vector from the last layer given an input  $x$  (e.g., an image).

Let  $x$  be an arbitrarily chosen input. Suppose  $M_{32}(x)$  returns  $(p_0, \dots, p_{N-1})$ , and  $M_{16}(x)$  returns  $(p'_0, \dots, p'_{N-1})$ . Clearly, the classification result of an input  $x$  made by a clas-

<sup>1</sup>For simplicity, this definition of the rounding operation ignores the case where a tie needs to be broken.

Table 2. The columns of "Floating-point errors" and "Error tolerance" refer to statistics of  $\delta(M_{32}, M_{16}, x)$  and  $\Gamma(M_{32}, x)$  respectively, where  $x$  ranges over the images of the MNIST dataset.

Epochs	Floating-point error				Error tolerance			
	Min	Max	Mean	Variance	Min	Max	Mean	Variance
10	0.00E+00	1.68E-01	2.76E-03	5.62E-05	9.29E-05	1.00E+00	7.66E-01	7.58E-02
20	7.84E-15	2.07E-01	2.88E-03	8.53E-05	3.65E-05	1.00E+00	8.24E-01	6.31E-02
50	0.00E+00	3.82E-01	3.69E-03	1.97E-04	2.74E-06	1.00E+00	8.79E-01	4.72E-02
100	0.00E+00	5.64E-01	4.12E-03	3.27E-04	3.24E-04	1.00E+00	9.16E-01	3.42E-02
200	0.00E+00	6.75E-01	4.23E-03	4.76E-04	1.93E-04	1.00E+00	9.47E-01	2.19E-02
500	0.00E+00	9.35E-01	3.76E-03	6.18E-04	1.44E-03	1.00E+00	9.79E-01	7.72E-03
1000	0.00E+00	9.95E-01	3.14E-03	5.77E-04	1.91E-03	1.00E+00	9.91E-01	2.98E-03

sifier  $M$  is given as

$$\text{pred}(M, x) \stackrel{\text{def}}{=} \arg\max_i \{p_i | p_i \in M(x)\}. \quad (3)$$

Due to the floating-point error, the classification results of  $M_{32}$  and  $M_{16}$  on  $x$  can be different. To quantify this difference, we define the *floating point error* as follows.

**Definition 4.1.** Given the 16-bit classifier  $M_{16}$ , the 32-bit  $M_{32}$ , and an input  $x$ , the *floating point error* between the classifiers is given as

$$\delta(M_{32}, M_{16}, x) \stackrel{\text{def}}{=} |M_{32}(x) - M_{16}(x)|_\infty \quad (4)$$

The degree to which this difference affects the outcome is an important question we investigate in this work. In fact, we can show a sufficient condition (denoted by C) that guarantees the absence of difference between two classifiers.

(C.) If the difference between the largest of  $p_i$  and the second largest is greater than twice the floating point error, then the two classifiers  $M_{16}$  and  $M_{32}$  have the same classification result on  $x$ .

Illustration for condition (C): suppose  $M_{32}(x) = (0.8, 0.1, 0.05, 0.05)$  for a classification problem of four labels. Let the largest error between this probability vector and  $M_{16}(x)$  be  $\delta$ . Then in the worst case, 0.8 can drop to  $0.8 - \delta$  for the 16-bit, and the second largest probability becomes  $0.1 + \delta$ . If  $0.8 - \delta > 0.1 + \delta$ , then  $M_{16}$  and  $M_{32}$  must have the same classification result on  $x$ , e.g.,  $M_{16}(x) = (0.7, 0.15, 0.1, 0.05)$ .

Below, we formalize condition (C) following introduction of the notion of *error tolerance*.

**Definition 4.2.** The *error tolerance* of a classifier  $M$  with respect to an input  $x$  is defined as the gap between the largest probability and the second-largest one:

$$\Gamma(M, x) \stackrel{\text{def}}{=} p_0 - p_1, \quad (5)$$

where  $p_0 = |M(x)|_\infty$ , and  $p_1 = |M(x) \setminus p_0|_\infty$ .

Here,  $M_{32}(x) \setminus p_0$  refers to a vector of elements in  $M_{32}(x)$  but with  $p_0$  removed. The error tolerance can be thought of as quantifying the stability of the prediction. We have the following lemma corresponding to condition C.

**Lemma 4.3.** Consider a classification problem characterized by a pair  $(X, Y)$  where  $X$  is the space of input data, and  $Y = \{0, \dots, N-1\}$  is the labels of classification. Suppose a learning algorithm trains a 32-bit model  $M_{32} : X \rightarrow \mathbf{F}_{32}^N$  and a 16-bit model  $M_{16} : X \rightarrow \mathbf{F}_{16}^N$  on a dataset  $D \subseteq X \times Y$ .

We have: If

$$\Gamma(M_{32}, x) \geq 2\delta(M_{32}, M_{16}, x) \quad (6)$$

then  $\text{pred}(M_{32}, x) = \text{pred}(M_{16}, x)$ .

*Proof.* Let  $M_{32}(x)$  be  $(p_0, \dots, p_{N-1})$ . Without loss of generality we assume  $p_0$  is the largest one in  $\{p_i\}$  ( $0 \leq i \leq N-1$ ). We denote  $\delta(M_{32}, M_{16}, x)$  by  $\delta$  hereafter. Following Eq. 6, we have

$$\forall i \in \{1, \dots, N-1\}, p_0 - p_i \geq 2\delta. \quad (7)$$

Let  $M_{16}(x)$  be  $(p'_0, \dots, p'_{N-1})$ . Then for each  $i \in \{1, \dots, N-1\}$ , we have

$$\begin{aligned} p'_0 &\geq p_0 - \delta && \text{By Eq. 4 and Def. of } p_0, p'_0 \\ &\geq p_i + \delta && \text{By Eq. 5 and Eq. 7} \\ &\geq p'_i && \text{By Eq. 4} \end{aligned}$$

Thus  $p'_0$  remains the largest in the elements of  $M_{16}(x)$ .  $\square$

Table 3. Comparing accuracy and loss results between 32-bit and 16-bit neural networks on the MNIST dataset.

Epochs	Train accuracy		Test accuracy		Train loss		Test loss	
	32-bit	16-bit	32-bit	16-bit	32-bit	16-bit	32-bit	16-bit
10	90.3%	90.0%	90.8%	91.0%	3.49E-01	3.69E-01	3.23E-01	3.39E-01
20	92.2%	92.3%	92.8%	92.8%	2.71E-01	2.94E-01	2.57E-01	2.74E-01
50	94.9%	95.3%	94.9%	94.7%	1.81E-01	2.10E-01	1.80E-01	2.04E-01
100	96.7%	96.4%	96.3%	95.8%	1.16E-01	1.49E-01	1.28E-01	1.52E-01
200	98.4%	97.3%	97.4%	97.3%	6.05E-02	9.32E-02	8.97E-02	1.10E-01
500	99.8%	99.1%	97.7%	98.1%	1.38E-02	4.00E-02	7.87E-02	8.54E-02
1000	100.0%	99.8%	97.8%	98.1%	2.97E-03	2.04E-02	9.02E-02	8.29E-02

Below we illustrate Lemma 4.3 through a simple neural network trained on MNIST. Our 32-bit implementation has three Dense layers followed by a softmax layer at the end. Our 16-bit implementation uses the same architecture, except all floating-point operations are performed on 16-bit. Table 4 shows our results of error tolerance  $\Gamma$  and floating-point error  $\delta$ . Observe that the mean floating-point error is of the magnitude of  $1E-3$  with a variance of  $1E-5$  or  $1E-4$ ; the error tolerance is  $1E-1$  with a variance of  $1E-2$ . Thus, one can argue that Eq. 6, namely,  $\Gamma > 2\delta$ , holds for most data in MNIST. The table also shows that floating-point errors can be larger than the tolerance in some corner cases. Thus, we expect our 16-bit and 32-bit implementations to have close but different accuracy results.

Results at training and testing are presented. Table 4 shows the accuracy and loss results in MNIST comparing 16-bit and 32-bit implementations. We can see consistently that the 16-bit results have accuracy close to those of the 32-bit models, sometimes even better. This result motivates us to study whether the 16-bit model is similar to the 32-bit model for more complex neural networks.

In theory, if 80% of data in a dataset satisfy Eq. 6, Lemma 4.3 tells us that the 32-bit and 16-bit models will have at least 80% of classification results being the same. The main challenge here is that we cannot determine if Eq. 6 always holds or the percentage of data that satisfies it. We believe that for complex neural networks, most data meet Eq. 6. This is because the loss function for the classification problem is a cross-entropy in the form of  $-\sum \log(p_i)$ , which should guide  $p_i$  toward 1 during training and in turn, causes a large error tolerance  $\Gamma$  compared to relatively small  $\delta$ . In fact, Table 4 shows that the floating point errors ( $\delta$ 's) are nearly two orders of magnitude smaller than the  $\Gamma$ 's. Although the gap might close over the epochs, the difference remains sufficiently large to satisfy the condition of the lemma.

With this theoretical development and observations, we pro-

pose the following conjecture.

The accuracy of a 16-bit neural network for classification problems, in the absence of significant errors involving floating-point overflow/underflow, will be close to that of a 32-bit neural network.

We anticipate a situation where floating-point errors can become significant due to overflow or underflow since 16-bit floating-point is known to have a smaller range (Table 3.1). The conjecture may be surprising, so we devote our next section to in-depth validation.

## 5. Experiments

We aim to compare the performance of 16-bit operations to 32-bit operations in deep neural network (DNN)<sup>2</sup> and convolutional neural network (CNN) models. we experiment over three CNN models, including AlexNet (Krizhevsky et al., 2012), VGG16 (Simonyan & Zisserman, 2015), and ResNet-34 (He et al., 2016). We will see how the different precision settings affect the computational time and accuracy of the models. Unlike case studies in the previous section, we use 100 epochs for all experiments and gradually increase the batch size from 64 to 384 to see how it affects 16-bit training. All random seeds used in this study's experiments are fixed to facilitate the comparison. The experiments were conducted on NVIDIA's RTX3080 Laptop GPU.

Table 4 gives the result most representative of our work. A more detailed description and analysis of these results will follow in the subsequent subsections.

<sup>2</sup>While DNNs subsume CNNs, we use the term DNN to refer to fully-connected, non-convolutional neural networks.

Figure 1. Top-1 accuracy (top row) and computational time (bottom row) on MNIST Classification using DNN

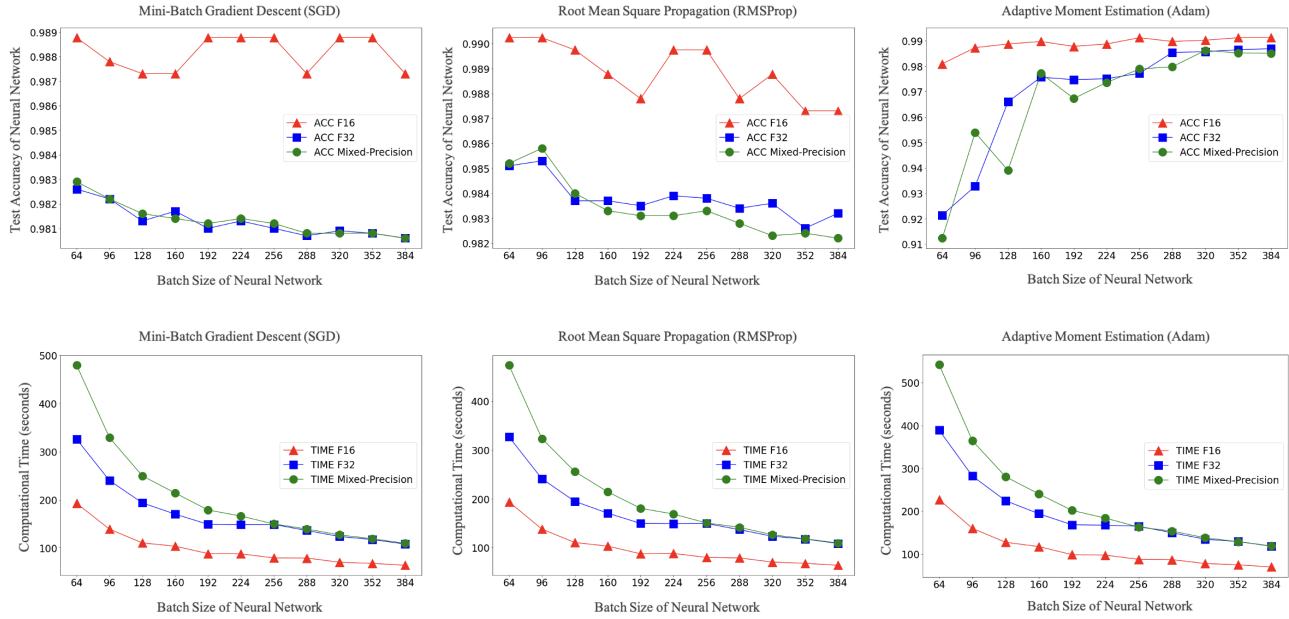


Figure 2. Top-1 Accuracy, Top-2 Accuracy, and Computational Time for Cifar-10 Classification

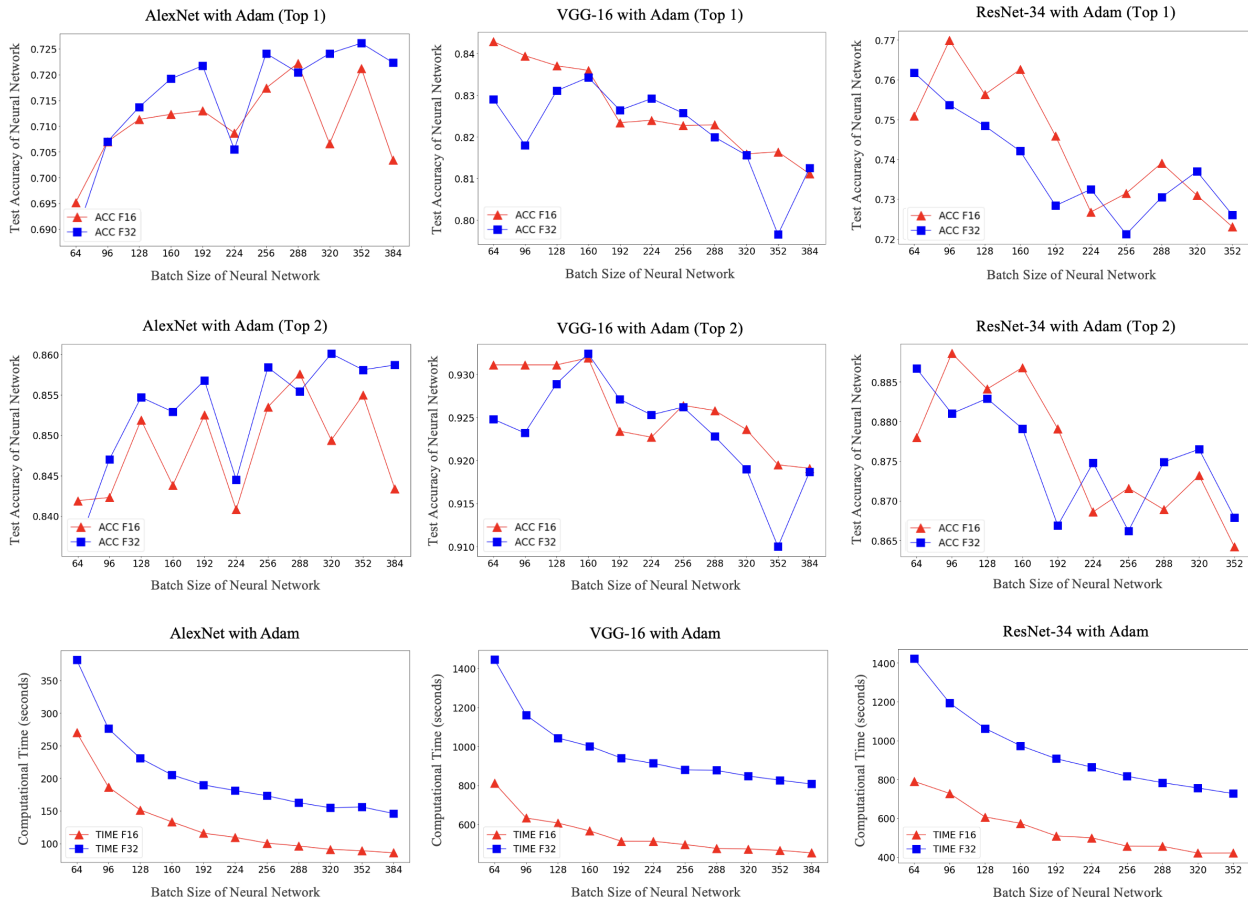


Table 4. Summary of the time and accuracy performances of the three CNNs.

Model	Results	FP32	FP16
AlexNet	Time	381s	270s
	Accuracy	68.9%	69.5%
VGG16	Time	1445s	812s
	Accuracy	82.9%	84.3%
ResNet-34	Time	1914s	1058s
	Accuracy	76.6%	76.1%

### 5.1. DNN Experiments

In the DNN experiments, we train a DNN with three hidden layers to perform the MNIST classification task and compare the performances between those of the 16-bit model and the 32-bit model. Each of the three layers in the DNN has 4096 neurons, whose outputs are fed to the next layer after passing through ReLU activation. Other works, such as (Micikevicius et al., 2018), use a 32-bit softmax layer regardless of the overall precision settings to prevent potential numerical instability, but we leave stick with a 16-bit softmax to see how a "pure" 16-bit model fares against 32-bit ones. We also vary the types of optimizers among RMSProp, Adam, and SGD to examine their effects on performance. In addition, we confirm from this experiment that not only SGD but also other optimizers, such as RMSProp and Adam, can be used in 16-bit if  $\epsilon$  is properly set.

In all experiments in this section, we use the learning rate of  $10^{-3}$  and fix  $\epsilon$  in RMSProp and Adam to  $10^{-3}$  as well. We direct readers to the Appendix for experiments in other settings. Finally, in addition to the 32-bit baseline, we also compare the mixed-precision training algorithm proposed by (Micikevicius et al., 2018) (as provided by TensorFlow). Figure 1 shows that 16-bit deep neural networks are better than 32-bit and mixed precision in terms of computational time while maintaining similar test accuracy. In detail, (see Figure 3 in the Appendix) 16-bit SGD's computational time is decreased by 40.9%, and the test accuracy was increased 0.6%, respectively, compared to 32-bit while 16-bit computational time was decreased by 59.8% and accuracy was increased 0.59% compared to mixed precision when the batch size was the smallest at 64. Other batch sizes yielded consistent trends, albeit with smaller magnitude (Keskar et al., 2017).

Optimization-wise, 16-bit RMSProp reduced the runtime by 40.4% and 59.6% compared to 32-bit and mixed, respectively, and accuracy is increased by 0.3% and 0.4%. 16-bit Adam improved the runtime by 41.6% and 58.1%, and 6.4% and 7.5% in terms of accuracy compared to 32-bit and mixed precision, respectively. Of the total of 33

experimental groups calculated with three optimizers, 31 decreased running time by more than 40% compared to 32 bits while maintaining the accuracy to a similar level. Every computational time and test accuracy in 16-bit deep neural networks is better than those of 32-bit and mixed precision.

### 5.2. CNN Experiments

16-bit CNN experiments were conducted to determine whether 16-bit is enough for training a more complex image classification problem and how numerically different it is from 32-bit. We used the CIFAR10 dataset and three convolutional neural networks (CNN) models: AlexNet, VGG16, and ResNet-34. All of these experiments were carried out using Adam in a 16-bit environment. Since the batch normalization (BN) layer is not implemented as an off-the-shelf module in 16 bits, the experiment was conducted focusing on CNN models that could be used without the batch normalization layer. See the Appendix for treatment on 16-bit BN implementations.

#### 5.2.1. TRAINING TIME AND ACCURACY

Figure 1 shows that 16-bit CNNs are better than 32-bit and mixed precision in terms of computational time and test accuracy. Figure 2 shows that 16-bit operations can also be applied to CNN models for image classification. We found that 16-bit CNNs maintain similar accuracy to 32-bit.

**AlexNet** At the smallest batch size of 64, 16-bit AlexNet's top-1 and top-2 accuracy results are increased by 0.9% and 0.6% compared to 32-bit, respectively, and training time decreased by 29.1%. The running time of 16-bit AlexNet is reduced by more than 29% compared to its 32-bit counterpart.

**VGG-16** In terms of top-1 and top-2 accuracy, 16-bit VGG16 increased by 1.6% and 0.6% compared to 32-bit, respectively, and learning time decreased by 43.7%. All computational speeds of 16-bit VGG16 were decreased by more than 40% compared to 32-bit.

**ResNet-34** 16-bit ResNet-34 decreased 0.6% and 0.4% in terms of top-1 and top-2 accuracy, and computational time decreased 44.7% compared to 32-bit. The running time of 16-bit ResNet-34 were reduced by more than 39%.

Overall, a 2.6% (AlexNet, Batch Size: 384) decrease in 16-bit top1-accuracy compared to 32-bit in 33 experimental sets was the largest decrease, and the largest increase is 2.7% (ResNet-34, Batch Size: 192). The smallest decrease in running time is by 29.1% (AlexNet, Batch Size: 64) compared to 32 bits, and 45.6% (VGG16, Batch Size: 288) the largest decrease. In other words, all 16-bit experimental groups used in all experiments decreased their computational speed by at least 29.1%.

This experiment shows that by using 16-bit operations in

image classification with CNN models, the running time (whether training or testing) can be greatly reduced while maintaining similar or higher accuracy compared to 32-bit operations. Our results demonstrate the efficiency of neural networks in training CNN models in a low-precision setting.

### 5.2.2. MODEL SIZE

The preservation of the trained model weights is an important aspect of contemporary deep learning.

Table 5. Sizes of saved CNN models in 16- and 32-bit.

Model	FP16	FP32
AlexNet	41.99 MB	83.94 MB
VGG16	67.34 MB	134.61 MB
ResNet-34	171.99 MB	343.87 MB

It is useful for model training and storage if a model with similar accuracy has less storage size. Table 5 shows the stored model has about half the size of the 32-bit size, faithfully reflecting the half-precision size reduction. 16-bit neural network allows for reducing the size of the model by half while maintaining similar accuracy. This opens up possibilities for 16-bit models to afford more complex and accurate architectures to attain better results.

### 5.3. Limitation and Discussion

This subsection reports the limitations we find while using the 16-bit neural network.

**Light hyperparameter-tuning.** During our experiments, we do not need to tune hyperparameters of 16-bit neural network training, e.g. learning rates, except the epsilon in the settings of the optimizers. The SGD (Goodfellow et al., 2016) optimizer does not have epsilon, so it can be used directly. For the other optimizers we have tested in our experiments, RMSProp (Hinton et al.) and Adam (Kingma & Ba., 2015), we will need to change epsilon, whose default value (in Tensorflow), 1E-7, can easily trigger significant inaccuracy for 16-bit training.

As a detail, the parameter epsilon corresponds to  $\epsilon$  in the weight updates below:

$$\begin{aligned} \text{RMSProp} : w_t &= w_{t-1} - \eta \frac{g_t}{\sqrt{v_t} + \epsilon} \\ \text{ADAM} : w_t &= w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned}$$

These optimizers introduced  $\epsilon$  in to enhance numerical stability, but  $\epsilon = 1\text{E-}7$  in the denominator causes floating-point

overflow when  $v_t$  in RMSProp or  $\hat{v}_t$  in Adam are close to 0. As mentioned previously, we set  $\epsilon = 1\text{E-}3$  for 16-bit training.

**Missing 16-bit Batch normalization.** Tensorflow’s current batch normalization layer does not directly support pure 16-bit operations. The batch normalization layer presumably originates from mixed precision works, which would cast 16-bit input values to 32-bit for calculation and then down-cast to 16 bits. This type conversion only results in runtime overhead due to the intermediate 32-bit computation.

Since this work aims to report results on pure 16-bit neural networks, we have to implement a 16-bit batch normalization layer on our own. Our implementation of the 16bit batch normalization can be found in Appendix.

**Batch size.** Our CNN experiments show that the accuracy of 16-bit neural networks decreases in larger batch sizes. This can be due to the precision loss incurred when averaging the cost over a larger number of samples in the mini-batches. This is probably a minor limitation since neural networks in memory-constrained environments usually do not use large batch sizes.

Despite these limitations, we have confirmed that 16-bit neural networks learn as well as 32-bit with only minor fine-tuning ( $\epsilon$  in the optimizers). Thus, we believe ML practitioners can readily benefit from 16NN when it comes to solving op optimization problems since it is faster, less memory-consuming, yet achieves similar accuracy as the 32-bit. Instead of spending the same amount of time and space as 32-bit, we can form a network with an enhanced cost-to-benefit ratio.

## 6. Conclusion

In this work, we have shown that for classification problems, the 16-bit floating-point neural network can have an accuracy close to the 32-bit. We have proposed a conjecture on their accuracy closeness and have validated it theoretically and empirically. Our experiments also suggest a small amount of accuracy gain, possibly due to the regularizing effect of lowering the precision. These findings show that it is much safer and more efficient to use 16-bit precision than what is commonly perceived: the runtime and memory consumption is lowered significantly with little or no loss in accuracy. In the future, we plan to expand our work to verify similar characteristics in other types of architectures and problems such as generative models and regression.

## References

Bjorck, J., Chen, X., Sa, C. D., Gomes, C. P., and Weinberger, K. Q. Low-precision reinforcement learning: Running soft actor-critic in half precision. In *Proceedings of*



- International Conference on Machine Learning*, 2021.
- Chen, X., Hu, X., Zhou, H., and Xu, N. FxpNet: Training a deep convolutional neural network in fixed-point representation. In *Proceedings of the International Joint Conference on Neural Networks*, 2017.
- Courbariaux, M., Bengio, Y., and David, J.-P. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Proceedings of Neural Information Processing Systems*, 2015.
- Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D., Avancha, S., Banerjee, K., Sridharan, S., Vaidyanathan, K., Kaul, B., Georganas, E., Heinecke, A., Dubey, P., Corbal, J., Shustrov, N., Dubtsov, R., Fomenko, E., and Pirogov, V. Mixed precision training of convolutional neural networks using integer operations. In *Proceedings of International Conference on Learning Representations*, 2018.
- De Sa, C., Feldman, M., Ré, C., and Olukotun, K. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of International Symposium on Computer Architecture*, 2017.
- Dean, J. The deep learning revolution and its implications for computer architecture and chip design. In *Proceedings of IEEE International Solid State Circuits Conference*, 2020.
- Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- Gopinath, S., Ghanathe, N., Seshadri, V., and Sharma, R. Compiling KB-sized machine learning models to tiny IoT devices. In *Proceedings of Programming Language Design and Implementation*, 2019.
- Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. In *Proceedings of International Conference on Machine Learning*, 2015.
- He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision*, 2016.
- Hinton, G., Srivastava, N., and Swersky, K. Neural networks for machine learning, lecture 6e. URL [http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf).
- IEEE Computer Society. Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pp. 1–70, 2008. doi:10.1109/IEEESTD.2008.4610935.
- Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., Yang, J., Park, J., Heinecke, A., Georganas, E., Srinivasan, S., Kundu, A., Smelyanskiy, M., Kaul, B., and Dubey, P. A study of BFLOAT16 for deep learning training, 2019. URL <https://arxiv.org/abs/1905.12322>.
- Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. In *Proceedings of International Conference on Learning Representations*, 2017.
- Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*, 2015.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *Proceedings of Neural Information Processing Systems*, 2012.
- Kumar, A., Seshadri, V., and Sharma, R. Shiftry: RNN inference in 2kb of RAM. *Proc. ACM Program. Lang.*, 4 (OOPSLA):182:1–182:30, 2020.
- Köster, U., Webb, T. J., Wang, X., Nassar, M., Bansal, A. K., Constable, W. H., Elibol, O. H., Gray, S., Hall, S., Hornof, L., Khosrowshahi, A., Kloss, C., Pai, R. J., and Rao, N. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Proceedings of Neural Information Processing Systems*, 2017.
- Lin, D. D., Talathi, S. S., and Annapureddy, V. S. Fixed point quantization of deep convolutional networks. In *Proceedings of the International Conference on Machine Learning*, 2016.
- Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. In *Proceedings of International Conference on Learning Representations*, 2018.
- Sharma, H., Park, J., Suda, N., Lai, L., Chau, B., Kim, J. K., Chandra, V., and Esmaeilzadeh, H. Bit Fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of International Symposium on Computer Architecture*, 2017.

- Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations*, 2015.
- Skeel, R. Roundoff error and the patriot missile. *SIAM News*, (4), 1992.
- Wang, N., Chen, C.-Y., and Gopalakrishnan, K. Ultra-low-precision training of deep neural networks. In *Proceedings of Neural Information Processing Systems*, 2019.
- Wu, S., Li, G., Chen, F., and Shi, L. Training and inference with integers in deep neural networks. In *Proceedings of International Conference on Learning Representations*, 2018.
- Xiao, G., Lin, J., Seznec, M., Demouth, J., and Han, S. SmoothQuant: Accurate and efficient post-training quantization for large language models, 2022. URL <https://arxiv.org/abs/2211.10438>.

## A. Extra experiments

### A.1. Top-1 and time results

Figure 3. MNIST classification top-1 accuracy and computational time

DNN Top-1 Test Accuracy												
Batch Size	Optimizers											
	Mini-Batch Gradient Descent (SGD)				Root Mean Square Propagation (RMSProp)				Adaptive Moment Estimation (Adam)			
	Float16	Float32	Mixed Precision	Relative Error (16-32)	Float16	Float32	Mixed Precision	Relative Error (16-32)	Float16	Float32	Mixed Precision	Relative Error (16-32)
64	98.8%	98.2%	98.2%	6.31E-03	99.0%	98.6%	98.5%	3.85E-03	98.0%	92.1%	91.2%	6.45E-02
96	98.7%	98.2%	98.2%	5.59E-03	98.9%	98.5%	98.4%	4.05E-03	98.7%	93.2%	95.3%	5.85E-02
128	98.7%	98.1%	98.1%	6.11E-03	98.9%	98.3%	98.3%	6.61E-03	98.8%	96.6%	93.9%	2.34E-02
160	98.7%	98.1%	98.1%	5.70E-03	98.8%	98.3%	98.3%	5.18E-03	98.9%	97.5%	97.7%	1.43E-02
192	98.8%	98.1%	98.1%	7.84E-03	98.7%	98.3%	98.2%	4.47E-03	98.7%	97.4%	96.7%	1.34E-02
224	98.8%	98.1%	98.1%	7.54E-03	98.9%	98.2%	98.4%	7.32E-03	98.8%	97.5%	97.3%	1.40E-02
256	98.8%	98.1%	98.1%	7.84E-03	98.9%	98.3%	98.3%	6.61E-03	99.1%	97.7%	97.8%	1.44E-02
288	98.7%	98.0%	98.0%	6.72E-03	98.7%	98.3%	98.2%	4.57E-03	98.9%	98.5%	97.9%	4.36E-03
320	98.8%	98.0%	98.0%	8.05E-03	98.8%	98.2%	98.2%	6.61E-03	99.0%	98.5%	98.6%	4.56E-03
352	98.8%	98.0%	98.0%	8.15E-03	98.8%	98.2%	98.3%	6.31E-03	99.1%	98.6%	98.5%	4.76E-03
384	98.7%	98.0%	98.0%	6.83E-03	98.7%	98.2%	98.2%	5.19E-03	99.1%	98.6%	98.5%	4.45E-03

DNN Training Time (Seconds)												
Batch Size	Optimizers											
	Mini-Batch Gradient Descent (SGD)				Root Mean Square Propagation (RMSProp)				Adaptive Moment Estimation (Adam)			
	Float16	Float32	Mixed Precision	Relative Error (16-32)	Float16	Float32	Mixed Precision	Relative Error (16-32)	Float16	Float32	Mixed Precision	Relative Error (16-32)
64	192.5	325.9	479.6	4.09E-01	194.0	326.0	480.6	4.04E-01	226.8	388.9	541.7	4.16E-01
96	138.8	239.9	329.1	4.21E-01	138.2	239.6	323.2	4.23E-01	159.7	281.9	364.3	4.33E-01
128	110.1	193.4	249.5	4.30E-01	110.7	193.9	254.4	4.28E-01	127.2	224.3	280.3	4.32E-01
160	103.3	169.9	213.9	3.91E-01	103.6	169.5	214.1	3.88E-01	117.2	193.9	240.2	3.95E-01
192	88.0	148.8	178.5	4.08E-01	87.8	148.2	180.9	4.07E-01	98.4	168.0	201.7	4.14E-01
224	87.4	148.2	165.9	4.10E-01	88.0	148.3	165.9	4.06E-01	97.2	167.2	183.7	4.18E-01
256	79.0	148.6	149.5	4.68E-01	80.2	149.0	148.3	4.61E-01	87.6	164.7	162.8	4.68E-01
288	78.8	136.0	139.1	4.20E-01	78.5	136.6	139.3	4.24E-01	86.7	149.6	153.6	4.20E-01
320	69.9	123.4	126.8	4.33E-01	70.3	122.9	125.8	4.27E-01	77.8	134.5	137.8	4.21E-01
352	67.6	117.2	118.7	4.22E-01	68.2	117.4	117.3	4.18E-01	74.6	129.0	128.5	4.21E-01
384	63.6	107.6	109.3	4.08E-01	63.7	108.2	108.7	4.10E-01	69.7	118.1	118.8	4.09E-01

### A.2. Using 16-bit Batch Normalization

As stated in the main text, the CNN results we provide in Section 5 are based on architectures without the batch normalization (BN) layers. Figure 4 shows that 16-bit and 32-bit CNN models' performances without BN layers. The main reason for excluding those layers is that there are no off-the-shelf 16-bit implementations for BN layers.

As a remedy, we implemented those manually and present the results. We chose to move this result from the main text to here because we were not able to fully verify that the implementation is truly a pure 16-bit one. That is, although we take every possible precaution to adhere to 16-bit computation on the Python level, we do not know if any unprecedented upcasting occurs in the library level (*e.g.*, `libcuda` or `cublas`). Hence, the results in Figure 5 are gathered from 16-bit neural networks 'to the best of our knowledge'.

From the figures, we can see that incorporating 16-bit BN layers still results in a similar pattern as the one given in the main text. With the exception of AlexNet, 16-bit networks give similar to superior performance compared to the 32-bit models.

# The Hidden Power of Pure 16-bit Floating-Point Neural Networks

Figure 4. CIFAR-10 classification top-1 and top-2 accuracy and computational time without BN Layers

CNN Top-1 Test Accuracy									
Batch Size	Convolutional Neural Network Models								
	AlexNet			VGG-16			ResNet-34		
	Float16	Float32	Relative Error (16-32)	Float16	Float32	Relative Error (16-32)	Float16	Float32	Relative Error (16-32)
64	69.5%	68.8%	9.58E-03	84.2%	82.9%	1.67E-02	76.0%	76.5%	6.39E-03
96	70.7%	70.7%	0.00E+00	83.9%	81.8%	2.62E-02	75.0%	76.1%	1.43E-02
128	71.1%	71.3%	3.36E-03	83.7%	83.1%	7.21E-03	76.9%	75.3%	2.14E-02
160	71.2%	71.9%	9.59E-03	83.6%	83.4%	2.03E-03	75.6%	74.8%	1.05E-02
192	71.3%	72.1%	1.20E-02	82.3%	82.6%	3.63E-03	76.2%	74.2%	2.76E-02
224	70.8%	70.5%	4.53E-03	82.4%	82.9%	6.27E-03	74.5%	72.8%	2.40E-02
256	71.7%	72.4%	9.25E-03	82.2%	82.5%	3.63E-03	72.6%	73.2%	7.78E-03
288	72.2%	72.0%	2.49E-03	82.2%	81.9%	3.65E-03	73.1%	72.1%	1.42E-02
320	70.6%	72.4%	2.41E-02	81.5%	81.5%	3.67E-04	73.9%	73.0%	1.17E-02
352	72.1%	72.6%	6.74E-03	81.6%	79.6%	2.49E-02	73.1%	73.7%	8.14E-03
384	70.3%	72.2%	2.61E-02	81.1%	81.2%	1.72E-03	72.3%	72.6%	4.13E-03

CNN Top-2 Test Accuracy									
Batch Size	Convolutional Neural Network Models								
	AlexNet			VGG-16			ResNet-34		
	Float16	Float32	Relative Error (16-32)	Float16	Float32	Relative Error (16-32)	Float16	Float32	Relative Error (16-32)
64	84.1%	83.6%	6.81E-03	93.1%	92.4%	6.81E-03	88.7%	89.1%	4.15E-03
96	84.2%	84.7%	5.54E-03	93.1%	92.3%	8.55E-03	87.8%	88.6%	9.81E-03
128	85.1%	85.4%	3.27E-03	93.1%	92.8%	2.36E-03	88.8%	88.1%	8.62E-03
160	84.3%	85.2%	1.06E-02	93.1%	93.2%	5.36E-04	88.4%	88.2%	1.35E-03
192	85.2%	85.6%	5.01E-03	92.3%	92.7%	3.99E-03	88.6%	87.9%	8.75E-03
224	84.0%	84.4%	4.38E-03	92.2%	92.5%	2.80E-03	87.9%	86.6%	1.40E-02
256	85.3%	85.8%	5.70E-03	92.6%	92.6%	2.15E-04	86.8%	87.4%	7.08E-03
288	85.7%	85.5%	2.57E-03	92.5%	92.2%	3.25E-03	87.1%	86.6%	6.23E-03
320	84.9%	86.0%	1.24E-02	92.3%	91.9%	5.00E-03	86.8%	87.4%	6.85E-03
352	85.5%	85.8%	3.61E-03	91.9%	91.0%	1.04E-02	87.3%	87.6%	3.76E-03
384	84.3%	85.8%	1.78E-02	91.9%	91.8%	4.35E-04	86.4%	86.7%	4.26E-03

CNN Training Time									
Batch Size	Convolutional Neural Network Models								
	AlexNet			VGG-16			ResNet-34		
	Float16	Float32	Relative Error (16-32)	Float16	Float32	Relative Error (16-32)	Float16	Float32	Relative Error (16-32)
64	270.1	381.4	2.91E-01	812.4	1445.3	4.37E-01	1057.6	1914.4	4.47E-01
96	186.5	276.0	3.24E-01	632.9	1160.2	4.54E-01	789.8	1422.2	4.44E-01
128	151.3	230.8	3.44E-01	607.2	1043.6	4.18E-01	727.6	1194.0	3.90E-01
160	133.1	205.1	3.51E-01	567.0	1001.4	4.33E-01	606.4	1062.6	4.29E-01
192	115.7	189.6	3.89E-01	513.7	940.6	4.53E-01	573.5	973.7	4.11E-01
224	109.2	181.2	3.97E-01	513.4	913.5	4.38E-01	508.8	906.9	4.38E-01
256	100.5	173.0	4.19E-01	497.1	879.9	4.35E-01	499.1	863.5	4.21E-01
288	96.1	162.5	4.08E-01	476.9	877.5	4.56E-01	455.9	815.9	4.41E-01
320	90.9	154.7	4.12E-01	474.0	848.3	4.41E-01	455.2	783.1	4.18E-01
352	88.7	155.9	4.30E-01	466.7	826.9	4.35E-01	420.2	755.7	4.43E-01
384	85.6	145.9	4.13E-01	454.2	807.4	4.37E-01	420.5	726.6	4.21E-01

Figure 5. Results from using 16-bit BN layers

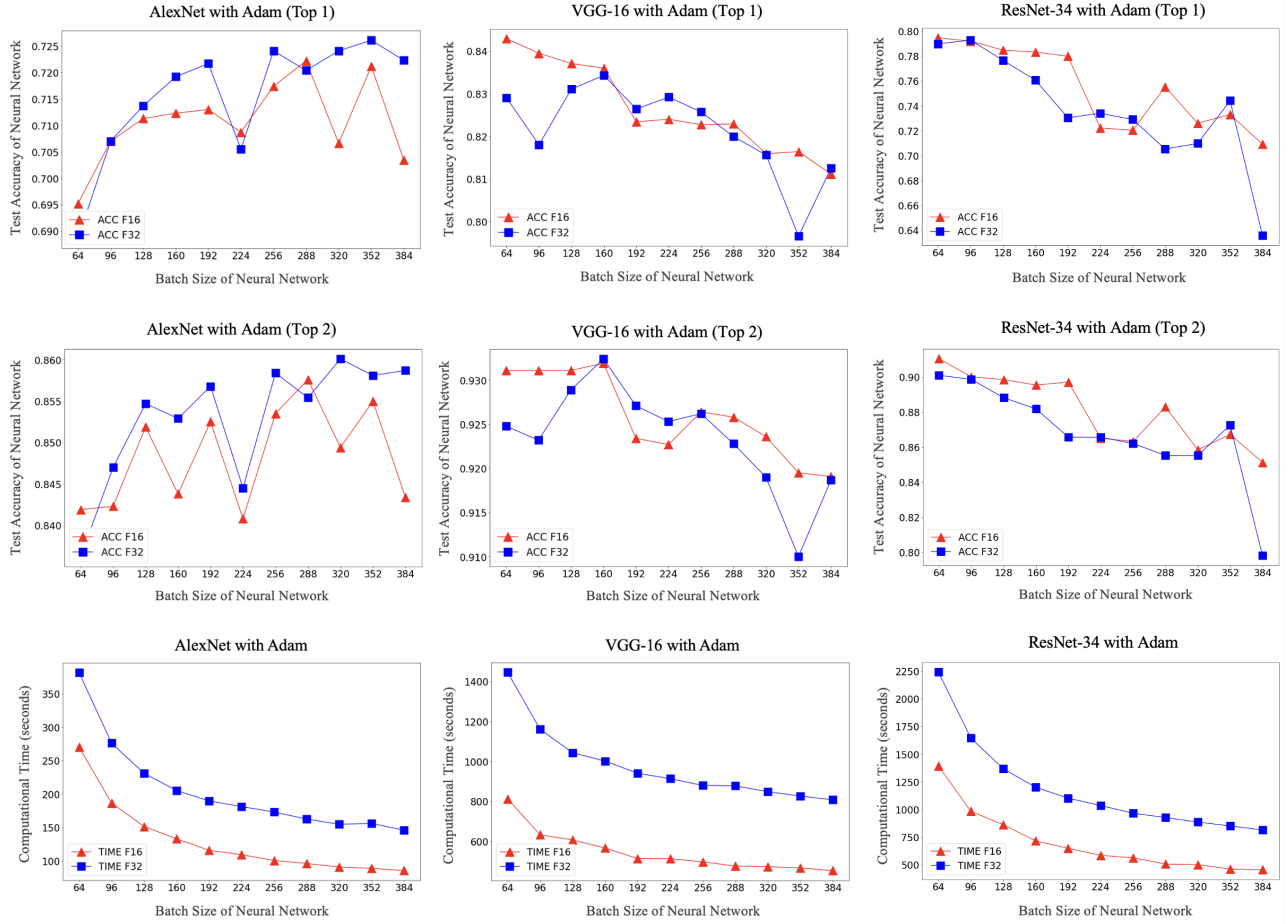


Figure 6. ResNet-34 with BN Layers CIFAR-10 classification top-1 and top-2 accuracy and computational time

Batch Size	ResNet-34								
	Top 2 Test Accuracy			Top 2 Test Accuracy			Training Time		
	Float16	Float32	Relative Error (16-32)	Float16	Float32	Relative Error (16-32)	Float16	Float32	Relative Error (16-32)
64	79.4%	78.9%	6.20E-03	91.0%	90.0%	1.04E-02	1393.8	2241.1	3.78E-01
96	79.2%	79.2%	1.13E-03	90.0%	89.8%	1.66E-03	982.5	1644.9	4.02E-01
128	78.4%	77.6%	1.05E-02	89.8%	88.8%	1.15E-02	861.1	1366.7	3.69E-01
160	78.3%	76.0%	2.97E-02	89.5%	88.1%	1.54E-02	715.2	1199.7	4.03E-01
192	77.9%	73.0%	6.77E-02	89.7%	86.5%	3.62E-02	647.9	1101.1	4.11E-01
224	72.2%	73.4%	1.60E-02	86.5%	86.5%	6.93E-04	584.2	1034.5	4.35E-01
256	72.0%	72.9%	1.19E-02	86.3%	86.2%	1.39E-03	562.1	965.1	4.17E-01
288	75.5%	70.5%	7.08E-02	88.2%	85.5%	3.22E-02	505.3	926.0	4.54E-01
320	72.5%	70.9%	2.28E-02	85.8%	85.5%	3.85E-03	499.8	885.4	4.35E-01
352	73.3%	74.4%	1.47E-02	86.7%	87.2%	6.07E-03	460.4	849.8	4.58E-01
384	70.9%	63.5%	1.15E-01	85.1%	79.8%	6.65E-02	454.5	814.1	4.41E-01