Comparing Reduced Bitwidth Representations for Machine Learning - Quantization vs Brain Float

Anson Tsai, Ryan Lund, Chufan Liang

University of California, Berkeley

***Abstract*—Machine learning (ML) has rapidly become an essential component of modern commercial products. As such, researchers have sought to optimize both software and hardware in order to support faster and more energy efficient training and inference, both for warehouse scale computers and for mobile devices. A recent area of focus is to use numerical representations with reduced bitwidth for weights and activations. One technique to reduce bitwidth is quantization, or using narrow integers along with a scaling factor in the place of a wider float value. An alternative technique is to use a novel numerical representation, such as the 16-bit brain floating point format (bf16). In this project, we will compare the impacts on accuracy, performance, and resource utilization of these reduced bitwidth methods by running tests on a Gemmini [2] generated systolic array coupled with a Rocket Core in-order [3] processor.**

I. Introduction

A. Current Constraints

Within the next five years, the global machine learning market is expected to reach nearly $100 billion in value. [4] This market is not confined to a single segment, and the range of applications for ML mirrors the size of the industry. On one side of the spectrum is warehouse scale computing with use cases such as big data analysis, recommendation systems, and user segmenting. Here, algorithms are run on massively parallel server farms, with a vast amount of computing resources at their disposal. In sharp contrast to this is the other side of the ML spectrum, mobile computing, where use cases such as virtual assistants and augmented reality programs must run on resource limited platforms. In spite of the enormous differences in scale, both sides of the ML market find themselves limited by similar constraints. These constraints can be succinctly broken down into three categories: latency, power, and accuracy.

For warehouse scale systems, load times, often driven by neural network latency, can have a significant impact on revenue. [5] Similarly, on mobile and edge devices decreased latency improves the user experience of a product. With regards to power, warehouse scale computers are limited by the amount of cooling capacity for a given server farm, which is often not improved as quickly as the underlying computer hardware. Mobile systems suffer from the same limitation, which is further exacerbated by the finite capacity of a battery. Finally, for both scales, accuracy is a primary goal for maximizing user experiences and revenue. Consider the case of Netflix, where 75% of views are driven by recommendations [6].

B. Benefits of Reduced Bitwidth

Today, many ML workloads are run on CPUs or GPUs. Yet, even as network architectures have evolved, the core computational kernels remain largely the same. This has lead to the rise of research into domain-specific hardware accelerators for deep learning. These accelerators can be constructed via FPGA based designs such as those used for Microsoft Asure, or ASIC based designs such as Google's Tensor Processing Unit (TPU).

Generally speaking, these different forms of neural network (NN) accelerators share a common base architecture designed around matrix multiplication: the systolic array. This architecture is composed of numerous multiply accumulate cells (MACs), which act as the unit level building block of the larger array. The array itself is laid out in a square, with each individual MAC receiving input from the MACs to the north and west of itself, and passing output to the MACs to the south and east of itself. On the highest level, activation data is passed to the north side of the array and weight data is passed to the west side of the array. After propagation, output is produced on the south side of the array for further accumulation. It has been shown that increasing the dimensions of the systolic array either by changing the number of MACs or the bitwidth of data can increase power consumption by up to 3.4x and area by up to 2.3x. [2] However, increasing the number of MACs in the array has also been shown to increase performance by up to 4x for ML inference.

With these metrics in mind, it seems that a logical path to increase performance without increasing resource consumption would be to increase the number of MACs in a systolic array while simultaneously decreasing the bitwidth of data. However, this approach requires the assumption that techniques exist to achieve similar accuracy between fill bitwidth and reduced bitwidth computation, all else constant. Research has focused on two main approaches to achieve this result: quantization and new numerical representation formats.

C. Quantization

Quantization is the processing of taking a set of floating point values (typically fp32) and converting them into a lower bitwidth representation. Typically, this process can be thought of as generating a step function that maps the range of possible values for fp32 to the range of possible values for the smaller representation. Research has shown that neural networks can be quantized to 8-bit integers (int8) with almost no loss in

accuracy [7] and is currently focused on achieving the same result with 4-bit integers [8].

There are dual benefits to quantization into small integers. First, by reducing the bitwidth of data values, less hardware is required for each MAC, which means that computation can run faster and consume lower power. This improvement is also shared by bf16, although to a lesser extent. The second benefit to quantization is that MACs can implement integer-only arithmetic [7]. As integer arithmetic hardware often consumes lower area than the corresponding floating point hardware, this results in additional resource savings.

The downside of these savings is that quantized networks can suffer from accuracy degradation due to the lost of expressible range for weights and activations. However, for most networks this loss can be minimized through careful consideration of outlier weights if converting a fp32 network to int8, or by training a quantization-aware network.

D. New Numerical Representation Formats - bFloat16

Another method that researchers have developed to exploit the advantages of reduced bitwidth is to use new numerical representation formats. These formats are designed to meet the specific needs of machine learning while minimizing the number of bits consumed. Of note is the brain floating point format pioneered by Google [1].

Similar to quantization, researchers realized that neural networks rarely need highly precise weight values to achieve high accuracy. However, instead of moving to small integers and giving up the range of fp32, bf16 acts as a hybrid between 16 bit floating point and 32 bit floating point that preserves the range of the 32 bit value in the footprint of a 16 bit value at the expense of precision. This is achieved by having the same number of exponent bits in bf16 as in fp32, obtaining the space for these extra bits by having less mantissa bits in bf16 than in traditional fp16. With fewer mantissa bits, the floating point multiplier is smaller, which saves additional hardware area beyond the savings from reduction in overall bitwidth. However, this also means that custom hardware is needed to support operations with a non-standard number of exponent and mantissa bits. We will go more into the costs and benefits of bf16 in the next section of this paper.

II. Comparisons of the State of Art

In this section, we will further examine cutting edge research into each method of running neural networks with reduced bitwidth values. For each method, we will take a deeper look at the contributions and findings of a prominent paper as well as any potential gaps we find in the research.

A. bFloat16

In "A Study of BFLOAT16 for Deep Learning Training" [9], Kalamkar et al. present an extensive empirical study on the effectiveness of using bf16 in various deep learning processes. For this study, the researchers adopted a training methodology that utilized a mixed precision, "bf16-input, fp32-accumulator", dataflow.

The paper evaluated the efficacy of using the mixed precision methodology across a comprehensive spectrum of differing frameworks. For convolutional neural networks (CNNs), similar accuracy was achieved between bf16 emulation and a fp32 baseline. Similar results were found in recurrent neural network (RNN) tests using the DeepSpeech2 topology and Google's Machine Translation model. These results, along with those gathered from other deep learning frameworks, support the overall finding that the bf16 datatype is able to achieve similar accuracy as fp32, even in the most demanding applications.

While the emulation strategy of zeroing out the last 16 bits of a fp32 value to create the equivalent of a bf16 value used in this paper does not lead to any concrete results regarding bf16 hardware, the paper claims that it can act as a fair approximation. The study that we aim to conduct should also provide insights on bf16 specific hardware, as it requires the implementation of hardware that operates directly on the bf16 datatype.

B. Quantization

In "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference" [7], Jacob et al. propose a quantization scheme that allows for inference to be carried out using 8 bit integer-only arithmetic while preserving accuracy from the pre-quantized model. By using integer-only operations for inference, this method allows for simple and commonly available integer-only hardware to be used for neural network workloads.

The paper evaluated the effectiveness of this quantization scheme by demonstrating the improvement it made to the latency-vs-accuracy trade-off on mobile cores. It showed that int8 quantized ResNets can achieve accuracy within 2% of their fp32 counterparts, and Inception v3 models accuracy within 4.5%. When given limited latency budgets, this paper also demonstrated that quantized MobileNets achieved better accuracy than non-quantized equivalents. Finally, for a static accuracy goal, quantization resulted in a 2x latency reduction without significant accuracy degradation.

While this paper clearly demonstrates the advantages of using int8 quantization for neural networks, it is important to note that the findings depend on the relative level of floating point optimization in the target chips. Our own study will attempt to remove this potential complication by running all simulations on identical underlying hardware and only making changes to the content of MACs in the Gemmini array.

C. Applications to This Project

While there have been papers published on the effectiveness of reduced bitwidths in ML workloads, either through bf16 or integer quantization, there have been no papers so far that directly compare the benefits of these two methods. Our work will fill this gap by providing insights into the trade-offs between bf16 and quantized int8. Our experiments will provide data for not only model accuracy, but also resource utilization statistics such as area, memory usage, and power

consumption. This will enable an objective comparison of the effectiveness and efficiency of bitwidth reduction methods.

III. Methodology

A. Common Flow

For the majority of testing, we worked within the Chipyard design framework [10]. This allowed us to create four different Rocket Core configurations, each attached to a Gemmini accelerator using a different datatype. The datatypes used were int8 and bf16 for experimentation, as well as fp16 and fp32 to act as a baseline for comparison. The goal of the fp32 baseline was to provide a reference point to the de facto industry standard numerical representation, this is the format that most papers in the field compare to. We added the additional fp16 baseline to add a point of reference for bf16. Given that both formats consume 16 bits, the data for fp16 will allow us to see if bf16 has additional resource savings beyond the traditional fp16 format.

For the int8, bf16, and fp32 configuration, we used the tools within Chipyard to generate a simulation binary via VCS. This allowed us to run a basic test of random matrix multiplication to prove the correctness of the Gemmini computations. Additionally, we ran a small neural network test that simulated attempted classification of 4 images from the CIFAR-10 [11] dataset by a modified version of LeNet-5 [12]. Due to the significant amount of time required to run classification on the simulations, we attempted to the cycles per inference values as a proxy for latency. However, after seeing that this value did not change across the first three configurations, decided to stop pursuing this approach, thus gathering no data for fp16.

Because it was not computationally feasible to run a test set of 6000 images on each simulation in order to determine accuracy, we instead trained four networks with our modified LeNet-5 architecture using Google's Tensorflow. One network was trained as quantization aware targeting int8, one was trained with mixed-precision bf16, one was trained with mixed-precision fp16, and one was trained with default fp32. This test is meant only to showcase the relative accuracies that can be achieved through highly optimized training procedures with each datatype. Of particular note is the accuracy result from fp16. Due to the limited range of fp16, additional methods such as loss scaling are required to attain state-of-the-art (SOTA) results when using mixed precision fp16 training [13]. Loss scaling is a method used to ensure that critical values are scaled within the range of fp16 to preserve accuracy. In effect, this approach addresses the numerical instability of using fp16 for compounding calculations. Due to this instability and the additional work required to address it, the domain-specific architecture industry has favored more stable formats such as bf16 and fp32 that do not require loss scaling.

Despite being run in Tensorflow, we claim that the accuracies from these trained models can provide a rough insight into the relative accuracy losses we would see from switching to each datatype from the baseline fp32. Given that the output of a CNN is deterministic for a given test image once training has finished and that inference is carried out using matrix multiplications and basic math operations, we would expect a very similar result when computation is carried out on a Gemmini generated array. This is particularly true of the quantized model, which uses integer only arithmetic without the need for rounding. For the floating point variants, the size of the systolic array along with Hardfloat's choice of rounding mode could result in the accumulation of slight deviations from the Tensorflow results. This may be accentuated by Tensorflow's utilization of a variety of optimized rounding modes in inference.

The key takeaway from our accuracy results is that both of our non-baseline networks should be able to achieve accuracy near that of our baseline implementations when sufficient tuning is done. This conclusion matches the findings of past works examining mixed precision and quantization [9] [8] [7]. For a more robust understanding of accuracy degradation with different datatypes, future work could be done to run the CIFAR-10 inference workload on the four Gemmini configurations using a cycle-accurate cloud-based simulation such as FireSim [18] to generate on device accuracy metrics. Of interest would be a comparison of Gemmini results for differing array sizes to the accuracies produced by Tensorflow.

To gather power and area statistics, we ran each configuration through the Hammer [14] VLSI flow until the end of place and route (PAR), using Cadence Genus and Innovus tools.

B. BF16 Specific Challenges

Gemmini uses the Hardfloat [15] library as the generator for floating point arithmetic units, which allows new float configurations to be defined by their exponent and significand width. This, paired with Gemmini accepting input as raw bits rather than a typed input, made adding bf16 support to Gemmini hardware a relatively simple task. A similar approach was also used to add fp16 support to Gemmini hardware.

However, neither C nor RISC-V contain support for bf16. This led us to add a barebones bf16 extension to the Softfloat [16] library to add support for the bf16 type and key arithmetic operations. Additionally, to generate test binaries for the bf16 simulation, additions were made to several tests within Gemmini-Rocc-Tests [17] to add proper handling of bf16 values and operations. Key changed tests include weight stationary matrix multiplication, CIFAR-10 quantization, and a new test for basic bf16 operation correctness.

C. Evaluation and Hypothesis

We predict that both bf16 and quantized int8 methods will perform better than the baseline fp32 in all metrics with the exception of model accuracy. When directly comparing bf16 and quantized int8, we expect that each will have advantages under different constraints. Looking strictly at the bitwidths, we hypothesize that quantized int8 will be advantageous in latency or resource bounded environments due to its smaller footprint. In less bounded environments, we expect that bf16 to be the superior choice due to its higher dynamic range.

Using the additional fp16 baseline, we hypothesize that bf16 and fp16 will be highly similar in terms of area and power consumption. Note that due to a lack of useful collected data with regard to latency, we will only focus on resource bonded environments for the rest of this paper.

IV. Measured Results

Fig. 1: Power (mW)

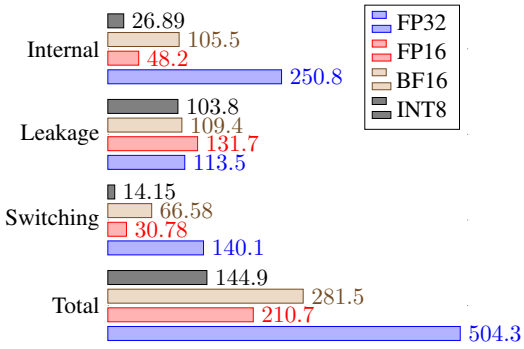

Fig. 2: Area (um^2)

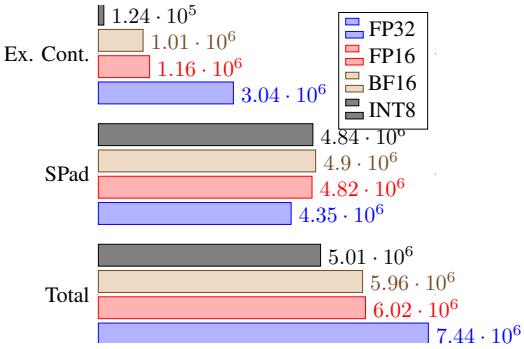

Table 1: Accuracy and Latency Metrics

	FP32	FP16	BF16	INT8
Accuracy	53.11%	54.75%	53.38%	52.21%
Cycles / Inference (Total)	3328271	N/A	3339487	3325983
Cycles / Inference (Matmul)	741768	N/A	749682	737942

The above figures and table summarize the key metrics collected during experimentation. As seen in Figure 1, switching from fp32 to bf16 resulted in a 1.79x reduction in total power and switching to int8 resulted in a 3.60x reduction in total power. Both of these total power changes were driven by decreases in internal and switching power. Unexpectedly, switching from fp16 to bf16 resulted in a 1.34x increase in total power consumption, largely driven by increases in both internal and switching power. Across all four configurations, leakage power remained largely the same, with fp16 did consuming the most leakage power and int8 consuming the least leakage power.

The trends in area are illustrated in Figure 2. The key metric here is the reduction in execute controller area. Given that scratchpad size is fixed, the magnitude of total savings is less dramatic. Regardless, switching to bf16 resulted in a 1.25x reduction in total area and a 3.02x reduction in execute controller area. The savings from switching to int8 were larger with a 1.48x reduction in total area and a dramatic 24.65x reduction in execute controller area. Finally, switching from fp16 to bf16 resulted in a negligible 1.01x reduction in total area but a 1.15x reduction in execute controller area. Note that in spite of scratchpad area remaining as a fixed number of bytes across all configurations, its area slightly increased for non-fp32 variants. This is something that may be worth exploring further.

With regards to accuracy, it is interesting to note that both fp16 and bf16 achieved higher accuracy than the fp32 baseline. Cautions on interpretation of the accuracy data were provided in the methodology section, but some discussion of possible causes will be given in the discussion section of this paper.

V. Discussion

A. *Reflection on Hypothesis*

The result of our experiments largely confirm the hypothesises made earlier in this paper. Both bf16 and int8 consumed less power and area than fp32, and beyond that int8 consumed less power and area than bf16. However, in spite of our prediction, the bf16 model achieved higher accuracy on the CIFAR-10 test set than the original fp32. A possible cause of this discrepancy is that the reduced precision of the bf16 model prevented it from overfitting to the training set. Moreover, the bf16 training used a highly optimized mixed-precision library, which may have resulted in a more efficient training process. Similar reasoning can be applied to the accuracy gains seen for fp16.

While our experimentation addressed a large part of our hypothesis, we failed to gather any meaningful data to inform an analysis of accuracy vs latency trade-off for the various numerical representations. All three tested configurations had nearly identical values (within 0.98x-1.01x) for both total cycles per inference and matrix multiplication cycles per inference. We would hope to address this shortcoming with future work, ideally designing an experiment that is able to codify target latency as a network design criteria. This would require deeper understanding of how delay scaled with the number of weights and layers in a network for a given numerical representation. This would require a large number of more accurate simulations, which could be aided through the use of a tool such as FireSim [18].

Comparing fp16 to bf16, there was a negligible area advantage and a significant power disadvantage to switching from fp16 to bf16. This would seem to indicate that for Gemmini, bf16 is not the most efficient 16-bit floating point format from a pure hardware perspective. However, as detailed in our methodology, on the software side significant effort is required to preserve numerical stability when using fp16 due to its limited range. As bf16 preserves the range of fp32, it does not suffer from such issues. Ultimately. the ideal choice of 16-bit float representation would depend on the resources at the disposal of a designer and their end goals (i.e. is power or stability a greater concern).

B. Ideal Applications for Each Representation

A lack of latency information notwithstanding, it is still possible to make assertions about the ideal application for both of the explored reduced bitwidth numerical representations.

While the int8 configuration suffered from a slight loss in accuracy due to the quantization process both in our tests and in other literature [7], its small area footprint and low power consumption indicate that it would be ideal for highly constrained devices such a smartphones and IoT devices. Furthermore, the ability to implement ML inference on integer only hardware means that an int8 systolic array requires significantly less design (when compared to the many edge case paths of floating point hardware). This makes int8 quantization also ideal for designers that wish to accelerate ML workloads on low cost or budget systems.

In contrast, the bf16 configuration saw an increase in accuracy when compared to its floating point counterpoint and likewise achieved better accuracy than the int8 model. Literature indicates that this trend of near baseline accuracy is expected of bf16 [9] However, the bf16 configuration also consumed more area and power than int8, yet less than the original fp32 baseline. This combination of attributes indicates that bf16 is optimal for warehouse scale computing, where area and power are less precious, but a difference in accuracy can be the difference between a sale and a lack thereof. It is also important to note that in our tests we saw hardware costs when bf16 compared to fp16. While additional software is required to preserve stability with fp16, future research could eliminate or mitigate this complication, which would make fp16 a more ideal 16-bit float representation for use on Gemmini generate systolic arrays. This indicates that continued work examining fp16 for ML workloads is warranted.

Regardless, of these use-cases are in many ways unsurprising, as they match the motivating research questions (optimizing for mobile devices and optimizing for warehouse scale devices) that led to their creations. However, this work has shown that each method of reduced bitwidth representation is ideally suited for its intended task, and not well suited for the opposite task. Thus, we would not expect either of these representations (int8 or some of 16-bit floating point) to become dominant and instead expect that they will continue to co-exist while accelerating different ends of the ML spectrum.

VI. Conclusion

Quantization and novel numerical representation formats are two promising frontiers for reducing the resource consumption of domain-specific neural network architectures. In our work, we aimed to fill a gap in the current research landscape by directly comparing bf16 and int8 quantization on similar hardware. Our experiments revealed that compared to a baseline fp32 based systolic array, switching to int8 resulted in a 1.48x reduction in accelerator area and a 3.60x reduction in accelerator power at the cost of a 0.9% accuracy reduction. Switching to bf16 resulted in a 1.25x reduction in accelerator area and a 1.79x reduction in accelerator power with an accuracy increase of 0.27% rather than an accuracy decrease. Our work also indicated that switching from fp16 to bf16 incurred a power increase of 1.34x with negligible area savings. From a pure hardware perspective this indicates fp16 should be ideal. However with numerical instability of fp16 factored in, the story becomes more dependant on the designers goal. While our work failed to provide any meaningful insight into the latency vs accuracy trade-off across the different representations, we were still able to identify ideal tasks for each bitwidth reduction technique. Quantization promises to reduce power and area for use on mobile devices while the greater accuracy of bFloat16 (and general 16-bit floating point) indicates that it will be best suited to warehouse scale inference tasks. We expect that both of these techniques will continue to gain in popularity, each accelerating their respective corner of the ML space.

References

[1] S. Wang, and P. Kanwar. "BFloat16: The Secret to High Performance on Cloud TPUs" Google, Google, 23 Aug. 2019, cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus

[2] H. Genc, A. Haj-Ali, V. Iyer, A. Amid, H. Mao, J. Wright, C. Schmidt, J. Zhao, A. Ou, M. Banister, Y. S. Shao, B. Nikolić, I. Stoica, and K. Asanovíc. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. arXiv:1911.09925 [cs.DC], 2019.

[3] C. Celio, D. A. Patterson, and K. Asanovic, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, 2015.

[4] Grand View Research, Inc. Machine Learning Market Size Worth $96.7 Billion by 2025 — CAGR: 43.8%: Grand View Research, Inc. PR Newswire, 13 Jan. 2020. https://www.prnewswire.com/news-releases/machine-learning-market-size-worth-96-7-billion-by-2025–cagr-43-8-grand-view-research-inc-300985428.html

[5] J, Brutlag. Speed Matters. Google AI Blog, Google, 23 Jun. 2009, https://ai.googleblog.com/2009/06/speed-matters.html

[6] X. Amatriain, J. Basilico. Netflix Recommendations: Beyond the 5 stars (Part 1). Netflix Technology Blog, Netflix, 6 Apr. 2012, https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429

[7] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. arXiv:1712.05877 [cs.DC] 2017

[8] R. Banner, Y. Nahshan, and D. Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. arXiv:1810.05723 [cs.DC] 2019

[9] D. Kalamkar, D. Mudigere, et al. A Study of BFLOAT16 for Deep Learning Training. arXiv:1905.12322 [cs.DC] 2019

[10] Chipyard. https://github.com/ucb-bar/chipyard

[11] A. Krizhevsky, V. Nair, G. Hinton. CIFAR-10. https://www.cs.toronto.edu/ kriz/cifar.html

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, November 1998.

[13] P. Micikevicius. Mixed-Precision Training of Deep Neural Networks. Nvidia Developer Blog, Nvidia, 11 Oct. 2017, https://devblogs.nvidia.com/mixed-precision-training-deep-neural-networks/

[14] Hammer. https://github.com/ucb-bar/hammer

[15] HardFloat. https://github.com/ucb-bar/berkeley-hardfloat

[16] SoftFloat. https://github.com/ucb-bar/berkeley-softfloat-3

[17] Gemmini-rocc-tests. https://github.com/ucb-bar/gemmini-rocc-tests

[18] Firesim. https://fires.im