Fine-Grained Exploitation of Mixed Precision for Faster CNN Training

Travis Johnston, Steven R. Young, Catherine D. Schuman, Junghoon Chae,
Don D. March, Robert M. Patton, and Thomas E. Potok
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37830
Email: {johnstonjt, youngsr, schumancd, chaej, marchdd, pattonrm, potokte}@ornl.gov

***Abstract*—As deep convolutional neural networks (CNNs) have become increasingly popular and successful at an ever-widening number of machine learning tasks specialized hardware has become increasingly available for training and deploying them. NVIDIA's recent Volta architecture includes *tensor cores* which perform a fused operation reduced and mixed precision (16-bit multiply, 32-bit accumulate). Recent research indicates that, typically, very little is lost (in terms of training accuracy) when half precision is used in place of single precision, and performance gains can be made by doing arithmetic in reduced precision. In this work we demonstrate that making layer-by-layer choices as to the arithmetic/data precision can lead to further performance improvement. In our study of 25,200 CNNs we demonstrate an average speedup (over purely half precision) of 1.27x and speedups as high as 3.64x by appropriately combining single and half precision arithmetic and data types on a layer-by-layer basis.**

I. Introduction

Deep learning approaches, especially those using convolutional neural networks (CNNs), have produced state-of-the-art results on a variety of tasks, including object recognition, object detection, and semantic segmentation [1], [2], [3]. The explosion of use-cases for CNNs to tackle complex tasks was in part due to the advent of GPUs, which, in the late 2000's, provided a 10-20x speedup in training over CPU implementations [4], [1]. Because of the explosion of interest in deep learning approaches, hardware developers such as NVIDIA are now incorporating specialized hardware into chip designs that target deep learning applications and are working with developers of deep learning frameworks to make that hardware accessible by the masses [5], [6].

For example, recent research in deep learning has indicated that, in many cases, reducing precision the mathematical operations of CNNs from single precision (32 bit floating point, or FP32) to half precision (16 bit floating point, or FP16) has relatively little impact on training accuracy, but can significantly speed up training and inference [7]. To exploit this deep learning result in hardware and produce even more performance improvements, NVIDIA's new Volta architecture includes tensor cores, which perform a fused multiply-accumulate operation in mixed precision (FP16 multiply, FP32 accumulate), and their associated deep learning library, cuDNN, can exploit the tensor cores. Since many popular deep learning frameworks rely on cuDNN, the Volta tensor cores can be immediately used by the community to potentially speed up training.

Among the challenges of successfully deploying deep learning networks to novel datasets is the design of a suitable neural network, e.g. hyper-parameter optimization or neural architecture search. Some hyper-parameters determine the shape of the loss surface; for example, the choice of network topology (number, type, and sequence of layers) and the topology's internal parameters (e.g., size of convolutional kernels, number of kernels, number of neurons in fully-connected layers, etc.) and the training dataset determine a loss surface and impact which features (both coarse and fine-grained) it is possible for the network to learn. Other hyper-parameters control the way in which the internal weights (or learnable parameters) are updated or learned; for example, the choice of optimization function (e.g., Stochastic Gradient Descent, ADAM), batch size, learning rate, learning rate decay schedule, momentum, weight decay, and others all impact the ability to find optimal values on the network's loss surface. Traditionally, hyper-parameter optimization has been carried out by hand and has been spread out over years and distributed across many independent research efforts; for example, reflect on the slow evolution of networks tuned to perform well on the ImageNet dataset.

Recently, CNN hyper-parameter and topology optimization approaches, including MENNDL [8], RAvENNA [9], and other derivatives [10], [11] have turned to high-performance computing (HPC) to expedite and automate this long, often tedious process, especially when the application space is novel or unique. These types of approaches, when utilizing leadership HPC resources, can evaluate/train thousands to millions of convolutional networks with widely varying hyper-parameters. Since hyper-parameter optimization requires the evaluation of many networks, any speedup gained during the training

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

process can significantly affect the overall performance of the optimization method, i.e., reducing the training/evaluation time of each individual network increases the number of networks that can be evaluated in the same amount of wall clock time resulting in a more thorough sampling of the hyper-parameter space. Therefore, we would like to exploit custom hardware such as the tensor cores on the NVIDIA Voltas in order to achieve training/evaluation speedup with the specific goal in mind of improving hyper-parameter optimization. However, because the implementations utilizing half-precision within each deep learning library are still relatively immature and are undergoing active development, it is not always clear when it is useful to utilize half precision arithmetic and when the highly optimized single precision implementations should be used, especially for CNNs with non-standard hyper-parameters or topologies.

For this study, we collected a dataset consisting of 25,200 CNN networks with varying hyper-parameter settings; these networks comprise a typical initial population of networks to evaluate for hyper-parameter optimization approaches. We evaluated the performance of each of those networks using both single and half precision, and we collected timing information at the layer level for both forward and backward passes of the training process. We demonstrate that choosing precision at the layer level rather than at the (whole) network level can provide a speedup in network training. In particular, we demonstrate that making layer-by-layer choices for precision leads to an average speedup of **1.27x** and speedups as high as **3.64x** by appropriately combining single and half precision operations.

II. Background and Related Work

Though convolutional neural networks (CNNs) were proposed and used as early as the late 1980's [12], the real explosion in the use of CNNs came in 2012 with dominating performance on the ImageNet dataset challenge [1]. This result was made possible in part by the efficient use of GPUs for training CNNs. The deep learning community has fully embraced GPU implementations, with most deep learning software packages being supported by GPU-enabled libraries such as cuDNN [13]. We anticipate that hardware improvements both for GPUs and other types of custom hardware implementations such as Google's tensor processing unit [6] will continue to accelerate deep learning performance in the coming years.

A key result for convolutional neural networks that has driven recent hardware developments is the use of reduced precision calculations. As many of the applications of deep learning use data that can be represented as 8-bit integers (i.e., images), performing operations at 32-bit precision is likely unnecessary for many problems. Thus, the use of half precision operations (i.e., 16-bit floating point) is attractive and has been shown to not sacrifice performance on a variety of problems [7]. As a result of this evidence, manufacturers of deep learning accelerators quickly exploited this opportunity and began to incorporate half precision operations into their hardware and software stacks [14]. More recently, NVIDIA has developed custom logic on their hardware targeting the multiply and accumulate operation inside the convolution operation. This new hardware feature, known as a tensor core, utilizes half precision arithmetic towards the goal of greatly increasing the capability of their GPUs for deep learning [5], and a 2-6x speedup of some applications has been demonstrated [15]. However, this previous work only studied a handful of possible networks and did not explore the many combinations of hyper-parameters that can have dramatic effects on computational performance. In this work, we perform a statistical study of computational performance across a large set of randomly generated networks.

III. Experiments

In this section we describe a series of experiments which explore the performance benefits of using half precision (FP16) data types and arithmetic over traditional single precision (FP32). We first demonstrate the overall impact on performance of using exclusively half precision versus single precision by comparing the average (total) time per mini-batch of training (including forward pass and backward pass); this comparison is made on 25,200 different neural networks. We dive deeper into the performance by breaking this down layer-by-layer during both the forward and backwards passes. Finally, we demonstrate performance gains by optimally selecting the precision on a per layer basis.

A. Framework and Hardware

Our experiments are all conducted using `nvcaffe`, NVIDIA's fork of BVLC Caffe [16], running on NVIDIA Volta V100 (16 GB) GPUs. The generation of our initial dataset was conducted on the pre-accepted (i.e., still under construction) Summit supercomputer at Oak Ridge National Lab. Subsequent experiments (which also reproduced and confirmed our initial observations) were conducted on an NVIDIA DGX-1 using the latest `nvcaffe` image available from the NVIDIA container registry (`nvcr.io/nvidia/caffe:18.11-py2`). Our experiments on Summit used 4,200 nodes of the machine, each equipped with six Volta GPUs for a total of 25,200 GPUs.

In this work, we exploit a particular feature of `nvcaffe`, which has the ability to specify numerical precision. Using `nvcaffe`, numerical precision can be specified in two ways. First, precision for the network as a whole can be specified using the options:

`default_forward_type`,
`default_forward_math`,
`default_backward_type`, and
`default_backward_math`.

These can be set to either `FLOAT` or `FLOAT16` to indicate single or half precision, respectively. Second, precision for an individual layer can be specified using similar options at the layer level: `forward_type`, `forward_math`, `backward_type`, and `backward_math` again with the options of `FLOAT` or `FLOAT16`. We will utilize both ways

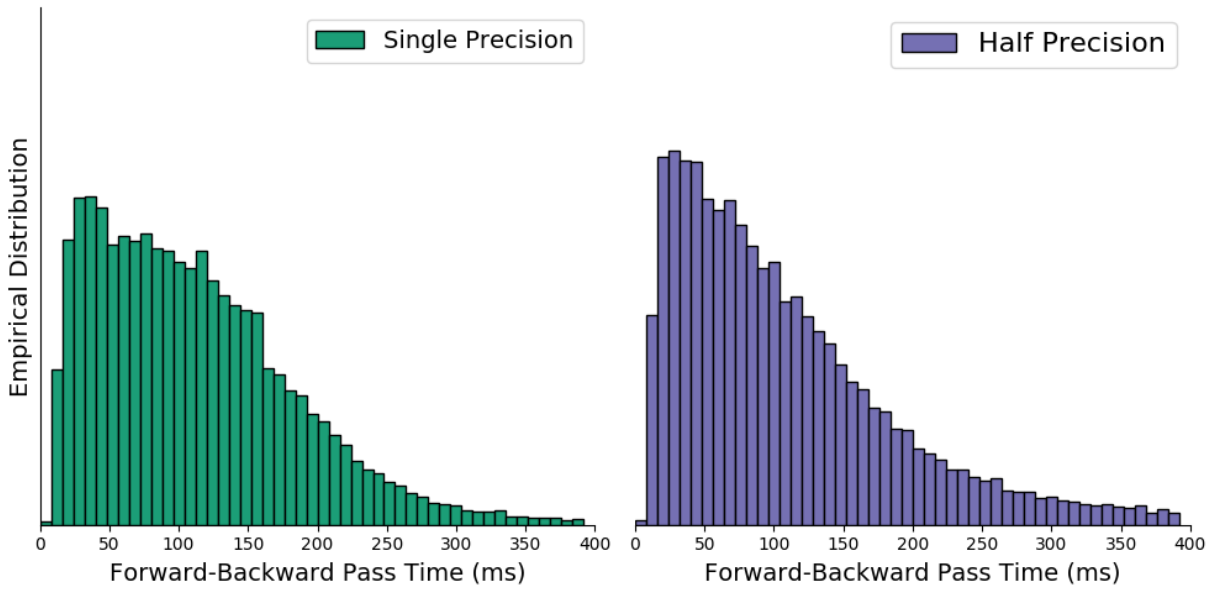

Fig. 1. The run time distribution when running each of the 25,200 (random) networks using entirely single precision arithmetic and data types (left, green) and, the corresponding distribution when running the same 25,200 networks using entirely half precision arithmetic and data types (right, purple). The time being measured is the (average) time for a single forward-backward pass (training on a single mini-batch).

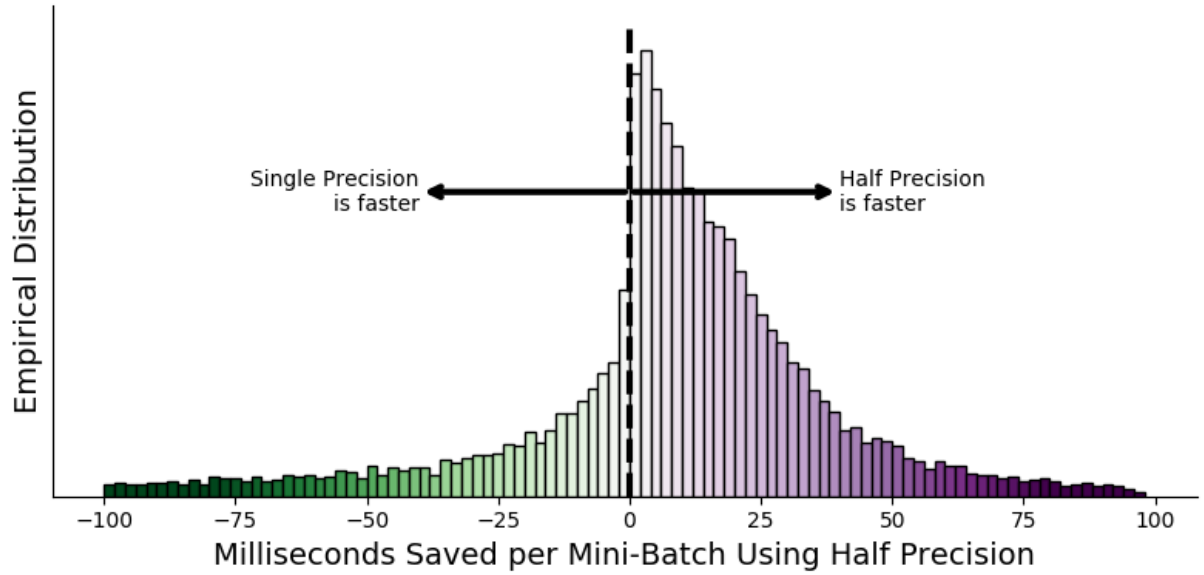

Fig. 2. The per network difference between single precision and half precision mini-batch times ($t_{\text{single}} - t_{\text{half}}$). Positive values indicate that half precision is faster (expected) and negative values indicate networks where single precision is faster (unexpected).

of specifying the precision in the following experiments. In all of our experiments we ensure that, at the layer level, forward types match forward math (i.e., either both are FLOAT or FLOAT16, no mixing and matching) and likewise that backward types match backward math. For `nvcaffe`, forward type and backward type refer to how the data is stored in memory; forward math and backward math refer to the arithmetic precision used in calculation.

With an eye towards hyper-parameter optimization, we generated an initial population consisting of 25,200 distinct neural networks. These 25,200 (one per GPU) are random variations of the traditional AlexNet topology–a total of 5 convolution layers, 3 (Max) pooling layers, 3 fully connected layers, interspersed with a variety of layers such as ReLU, batch normalization, and dropout; for a complete description of the sequence of layers, see Figure 8. The variations af-

fected the number of kernels, kernel sizes, and strides in all five convolution layers, the kernel sizes and strides in all three max-pooling layers, and the number of output neurons in the first two fully connected (inner product) layers; the third and final fully connected layer had a fixed number of outputs for our specific classification task. One important note is that in the fully connected layers the number of output neurons is always a multiple of eight; and, in the convolutional layers the number of learned kernels is always a multiple of four. These constraints were placed on the random selection in order to give the half precision implementation the best opportunity to provide speedup through the tensor cores. In total, the architecture consists of 26 layers (not all of which contain learned parameters), and we randomly alter 23 network hyper-parameters. Training (or evaluating the fitness of) these AlexNet variations is the first (random) phase of hyper-parameter optimization (what may happen in both MENNDL [8] or RAvENNA [9]) when AlexNet is used as a seed network. In subsequent generations of the evolutionary algorithm the network topology may expand (become deeper) or contract (become shallower). In our experience with scientific datasets, which are typically smaller and noisier than most standard, benchmark datasets, the networks typically contract resulting in shallower, faster (because they are less computationally expensive), and more accurate networks. As such, AlexNet provides a suitable starting point for understanding how different hyper-parameter settings affect half or single precision performance. For each of the following experiments we conduct we re-evaluate these same 25,200 networks.

B. Single vs. Half Precision Performance

NVIDIA has advertised up to 3x performance improvement using Volta V100 GPUs instead of Pascal P100 GPUs [17]. One of the reasons for the speedup is the availability of tensor cores and the ability to do half precision arithmetic. In our first experiment, our goal is to quantify the speedup achieved by using half precision (and the tensor cores) over single precision. Intuition and previous research on this topic indicate that half precision is expected be faster. To quantify the difference we evaluate the speed (using `caffe time`)–a detailed benchmarking suite built into `caffe`–of each of the 25,200 networks using (entirely) single precision and then repeat the experiment using (entirely) half precision.

Figure 1 shows the distribution of run times (average forward-backward time per mini-batch) in both single precision (left) and half precision (right). Since it is expected that, for any given network, the half precision implementation will be faster than the corresponding single precision implementation, we expect the distribution of half precision run times to be concentrated farther to the left than those for single precision. Generally, we do observe this trend with the exception that the half precision tail appears to be somewhat thicker than the single precision tail. It may not be obvious to some why these distributions are so wide, as each network has the same topology (i.e., that of AlexNET). However, if you consider a convolution layer, doubling the number of kernels will double the amount of computation required within that layer and thus impact the size of the data being fed into subsequent layers. Extending this argument to all the hyper-parameters makes it easy to see that even for a fixed topology, we can anticipate radically different run times for networks with different hyper-parameters.

Missing from Figure 1 is the impact on individual networks by shifting from single to half precision. Figure 2 shows the distribution of the number of milliseconds saved (per mini-batch of training) by switching from single precision to half precision: $(t_{\text{single}} - t_{\text{half}})$. One would expect that, except for minor variations due to system noise, half precision would never be significantly slower than single precision; this would be manifest by the distribution being entirely or nearly entirely non-negative. However, we observe that a significant portion of the distribution is negative (and sometimes significantly so) indicating that a number of the networks are significantly faster using single precision instead of half precision. These cases run completely contrary to intuition since half precision arithmetic should always be simpler (acting on fewer bits), hence faster, than single precision arithmetic. There may be any number of reasons for this surprising observation: there may be bugs (or naive implementations) in the software framework, these may be a natural consequence of design choices of the framework, or consequences of special purpose hardware (e.g., extra data movement in order to take advantage of specialized half precision cores). While we could delve into the causes we chose to take a more familiar path to data scientists and attempt to mitigate the problem without making fundamental changes to the framework's source code which have detrimental impacts on portability/deployability of codes.

C. Fine-Grained Layer-by-Layer Performance

To investigate the differences in performance between single and half precision, we probed further into network performance by examining layer-by-layer performance metrics. In particular, for each of these 25,200 networks, we use `caffe time` to measure the average forward time **per layer** (instead of per network) and the average backward time **per layer**–the time required to process one mini-batch. The timing is conducted both in single precision (32-bit floating point arithmetic and data types) and in half precision (16-bit floating point arithmetic and data types). We exclude results from a few networks that could not be profiled because they would consume more memory than available on the GPU (i.e., the computation fits in memory when entirely done in single or half precision but not in mixed precision). After compiling these results we have timing data for 122,020 convolution layers (various layer input sizes and layer hyper-parameters), 73,212 max pooling layers, 73,212 fully connected layers, 170,828 ReLU layers, 97,616 batch normalization layers, 48,808 dropout layers, as well as the data and loss layers for each network. The shape of a layer's input (batch size, number of channels, height and width) as well as the layer's hyper-parameters–including data types and math types–all impact the time spent in the layer.

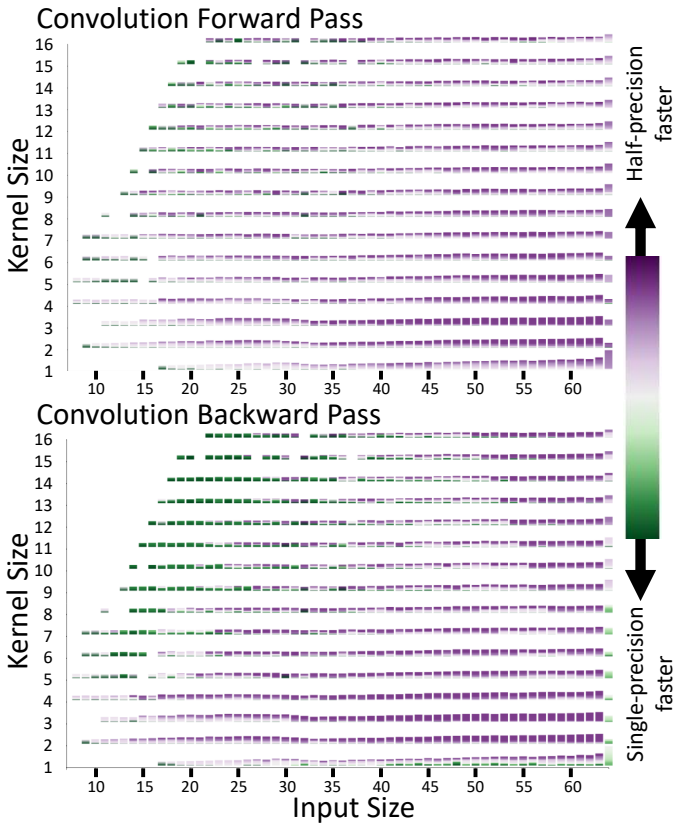

Fig. 3. A high level overview of performance in convolution layers broken down by layer input size and kernel size. For each input-size kernel-size pair, the bin represents the frequency at which half precision is faster and magnitude of the improvement in speed. Darker purple represents a larger difference in run time (faster in half precision) whereas darker green represents a larger difference in run time (faster in single precision). The height of the bin is proportional the the square root of the number of examples in the dataset. Note that the magnitude of the difference in run time skews heavily towards the purple (i.e. when half precision is faster, it can be as much as 100s of milliseconds faster whereas when single precision is faster it is typically only a few milliseconds to 10s of milliseconds faster).

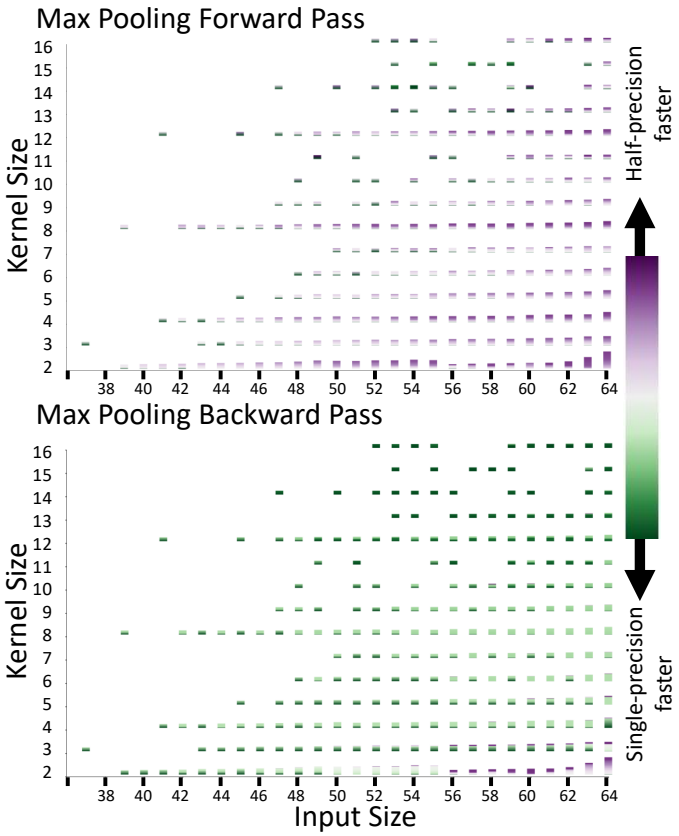

Fig. 4. A high level overview of performance in max pooling layers broken down by layer input size and kernel size. For each input-size kernel-size pair, the bin represents the frequency at which half precision is faster and magnitude of the improvement in speed. Darker purple represents a larger difference in run time (faster in half precision) whereas darker green represents a larger difference in run time (faster in single precision). The height of the bin is proportional the the square root of the number of examples in the dataset. Note that the magnitude of the difference in run time in forward passes is very small, only a few milliseconds; however, in the backwards pass, the magnitude skews heavily towards the green (i.e. up to 100s of milliseconds faster in single precision or up to 10 ms faster in half precision).

Our goal is to understand and hopefully predict whether a layer will be faster in single or half precision. Figures 3 and 4 bin layers by type (convolution or pooling) and count the frequency at which single precision outperforms half precision (green) and the frequency at which half precision outperforms single precision (purple) as a function of the input size and kernel size; the size of the bin is proportional to the square root of the number of corresponding examples in the dataset. Some combinations of input size and kernel size were not sampled during this random network generation process, resulting in some gaps in the data.

Figures 3 and 4 capture a number of trends. For example, Figure 3 shows that, except when the input size is very small, convolution layers are almost always faster using half precision; and, Figure 4 demonstrates that except when the kernel size is very small (e.g. kernel size at most three) the backward pass through pooling layers is almost always faster in single precision than in half precision. In light of the clear trends that are visible, we expect that standard machine learning classification tools will be quite successful at predicting which data/math types will be fastest at a given layer of the network.

D. Optimizing with Mixed Precision

It is clear from the results in the previous section that with the current state of the hardware and software stack, under some conditions there are certain layers that perform significantly better in single precision than in half precision and vice versa. As such, we predict that there will be a speedup for certain networks by dynamically choosing for each layer whether half or single precision should be used, rather than simply using all half precision or all single precision calculations for the entire network. We make the decision on whether to use half or single precision based on an initial forward and backward pass through the network in both single and half precision. During these two, quick, exploratory runs, we

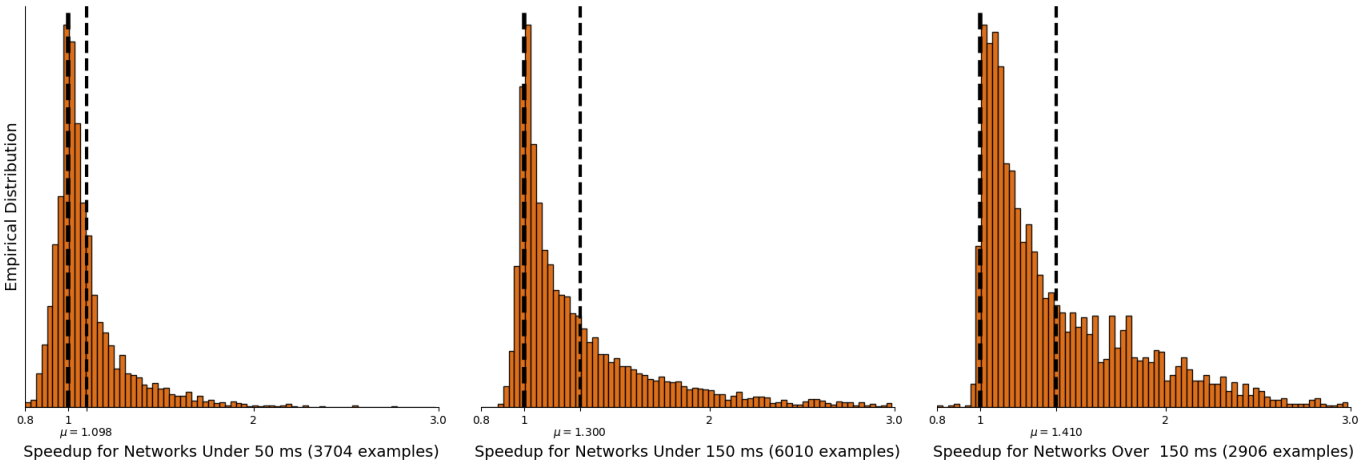

Fig. 5. Distribution of **observed** network speedups obtained using *fine-grained mixed precision*. The slowest networks, those requiring at least 150 ms per forward-backward pass, saw, on average, the largest (relative) speedup of 1.410x. Networks requiring between 50 and 150 ms per forward-backward pass saw, on average, a (relative) speedup of 1.300x; and the fastest networks, those with forward-backward time less than 50 ms, saw the smallest average (relative) speedup of 1.098x.

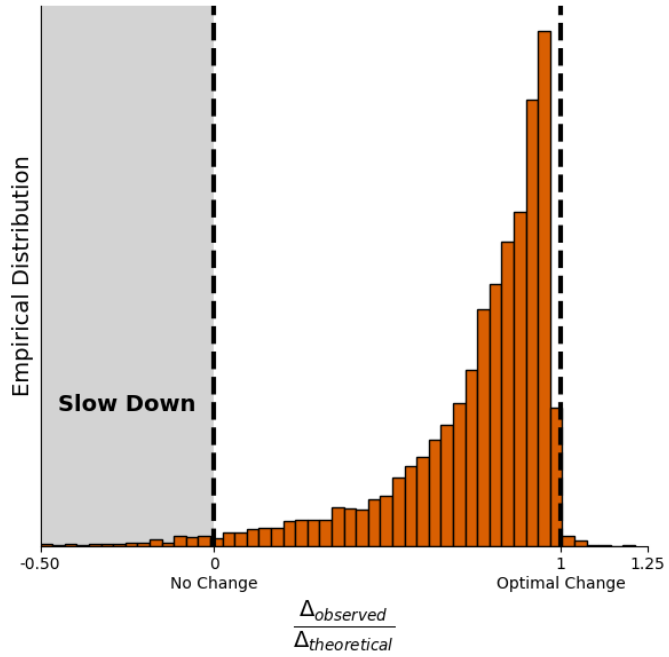

Fig. 6. Distribution showing the fraction of expected speedup we achieve. $\Delta_{observed} = t_{\text{half}} - t_{\text{mixed}}$ where both times are observed/measured. $\Delta_{theoretical} = t_{\text{half}} - t_{\text{ideal}}$ where t_{ideal} is the run time if every layer hit the theoretically optimal speedup.

measure performance layer-by-layer both in the forward pass as well as the backward pass exactly as in the previous section. The forward (similarly, backward) times per layer sum to the total network forward (backward) pass time. Thus, these per layer times take into account the conversions between single and half precision, if those conversions occur. We default to using half precision, but, when we observe a performance improvement of at least 1 ms in single precision, then we alter that layer to use single precision instead. We call this approach of choosing precision by layer *fine-grained mixed precision*. It is worth noting that the switch from all half to some half and some single precision should not adversely affect the accuracy of the networks.

Out of the 25,200 networks we looked at, we expected that approximately half of them could benefit from fine-grained mixed precision. In other words, for approximately half of the networks, we expected that by making the right layer-by-layer precision assignments we would see improved performance over both the exclusively single precision network and the exclusively half precision network. In this section, we restrict our attention to the networks that we expect to benefit from fine-grained mixed precision. Figure 5 shows the **observed speedup** using *fine-grained mixed precision* over all half precision evaluations for three types of networks: networks requiring less than 50 ms to evaluate (left, 3704 networks total), networks requiring between 50 ms and 150 ms to evaluate (center, 6010 networks total), and networks requiring over 150 ms to evaluate (right, 2906 networks total). The fastest training networks (under 50 ms per mini-batch) typically see only very modest speedups, on average 1.098x, and there are many cases where moving to fine-grained mixed precision can slow down training. We expect that this slow-down for fast networks is due to the high relative cost associated with the overhead from the type conversions. However, for the slower networks (networks that required more than 50 ms to evaluate), the average speedup was significantly higher: 1.3x on average for networks requiring less than 150 ms and 1.41x on average for networks requiring more than 150 ms. Perhaps more importantly, these network types were significantly less likely to have degraded performance as a result of fine-grained mixed precision. Overall, we saw an average speedup of 1.27x with the most-improved network seeing a 3.64x speedup.

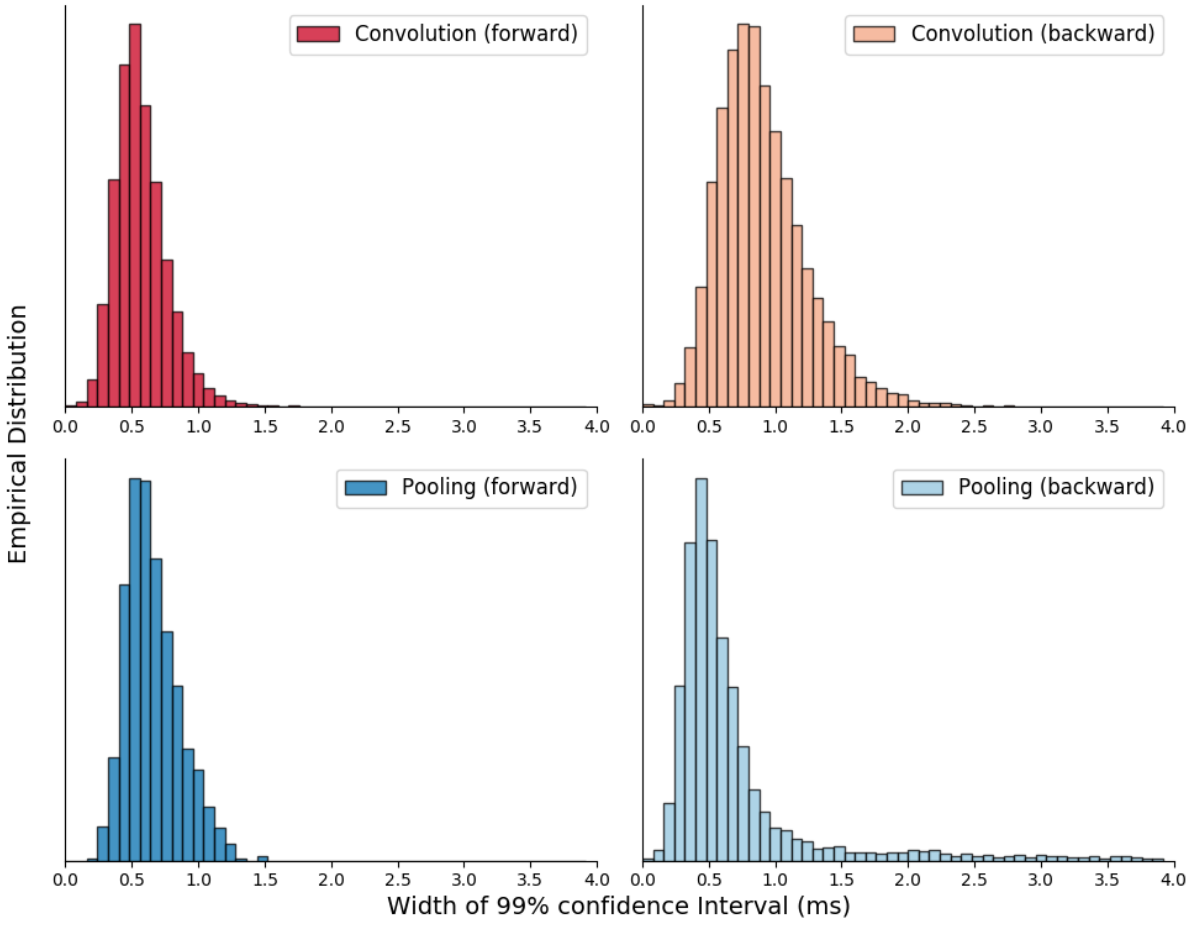

Fig. 7. Distributions of the absolute size of a 99% confidence interval around the mean time through convolution and pooling layers (forward and backward passes). Note that the width of the vast majority of the confidence intervals is less than about 1.5 milliseconds.

We would also like to understand how close our measured results are to those predicted by assuming the best run time (single or half precision) for each layer, which we define to be our theoretically optimal run time based on our initial calculations. Figure 6 shows a distribution of how nearly we achieve the theoretically optimal run time.

Since we observe random fluctuations in run time of up to ± 5 ms for networks overall, we exclude networks for which our expected improvement in run time is less than 10 ms–i.e., for those networks we consider it to be optimal to run in purely half precision (or purely single precision), these account for approximately half of the 25,200 networks. Figure 6 shows that most of our networks get close to the predicted speedup, which means that the cost of switching precision (maintaining multiple copies of data, type conversions, etc) is often small compared to the speedup obtained when we make the dynamic changes.

To probe further into how these speedups are achieved, we benchmarked a subset containing 4000 networks 16 times so that we could compute confidence intervals around average run times in layers to provide approximate error bars. Figure 7 shows the distribution of sizes of a 99% confidence interval (confidence around mean run time through the layer) in the most impactful (in terms of total run time) layers. As can be seen in this figure, the variation in layer run time due to noise is relatively minimal (usually less than 1.5 milliseconds) when compared to differences in network performance for single vs. half precision.

We visualize, in Figures 8 and 9, two examples of how the network time is broken down, by layer, in both forward and backward passes (using either entirely half precision (purple), entirely single precision (green), or our fine-grained mixed precision (orange)). Figure 8 provides an example confirming what we expect to see, that half precision is never slower than single precision. However, Figure 9 shows a surprising distribution. We selected the network in that figure because we expected a large speedup using fine-grained mixed precision.

The predicted mixed precision evaluation time should theoretically be the minimum of the half and single precision results, but due to noise and because of type conversion and

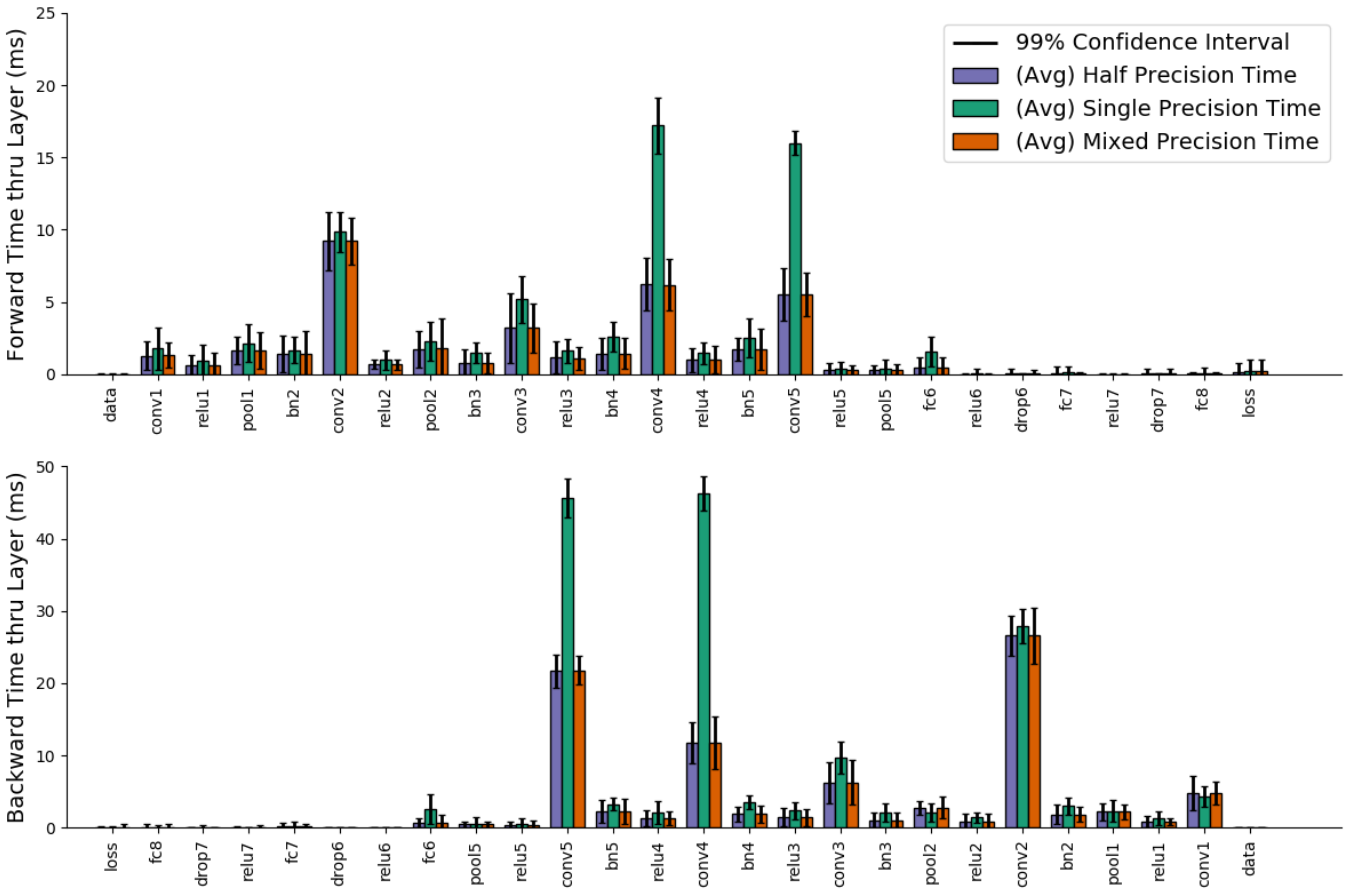

Fig. 8. Per-layer run times (forward and backward pass) when network is entirely half precision (purple)–expected to be fastest–entirely single precision (green), or when precision is specified layer-by-layer (i.e. mixed, orange/brown) to be the faster of half and single precision. In this case, the results are as expected with half precision being consistently faster, or at least as fast, as single precision–implying that the mixed precision model is actually entirely in half precision.

associated memory costs, the mixed precision performance may be slightly worse than expected. It is clear from this figure why neither all half nor all single precision performance should be utilized for all network types. In particular, on the backward pass, half precision performance is particularly bad for one of the pooling layers (pool2), while single precision performance is particularly bad for one of the convolution layers (conv2). By switching to fine-grained mixed precision for this network, we can save approximately 250 ms per mini-batch while training.

IV. Impact and Implications

In this work we demonstrate that using all half precision calculations in network forward-backward passes does not always result in speedup (over single precision) and can actually hurt performance. We propose an approach for using both single and half precision calculations within a single forward-backward pass of the network. In this approach, which we call fine-grained mixed precision, we do a preliminary measurement to determine whether single or half precision is more suitable for each layer type and then use the precision type that is faster for each layer when it can provide significant speedup. Overall we observed an average speedup (over baseline entirely half precision calculations) of 1.27x and, for slower training networks an average speedup of 1.41x with individual networks potentially seeing significantly more speedup (as high as 3.64x).

We recognize that these results rely on the current state of the NVIDIA software stack, as well as the implementations of the deep learning framework. However, we anticipate that, as the software continues to be refined, there will still be differences in performance between single and half precision that could be exploited to obtain performance gains. Furthermore, as new hardware implementations are introduced, it is likely that performance differences will persist–though the specifics of where they appear may likely change. As such, our proposed approach that will utilize benchmark timing results to decide how to configure layer-by-layer precision during hyper-parameter optimization in order to speed up individual network training and allow for overall improvements in the hyper-parameter optimization process, will continue to be an useful component moving forward.

Though a single network speedup of up to 1.41x could be

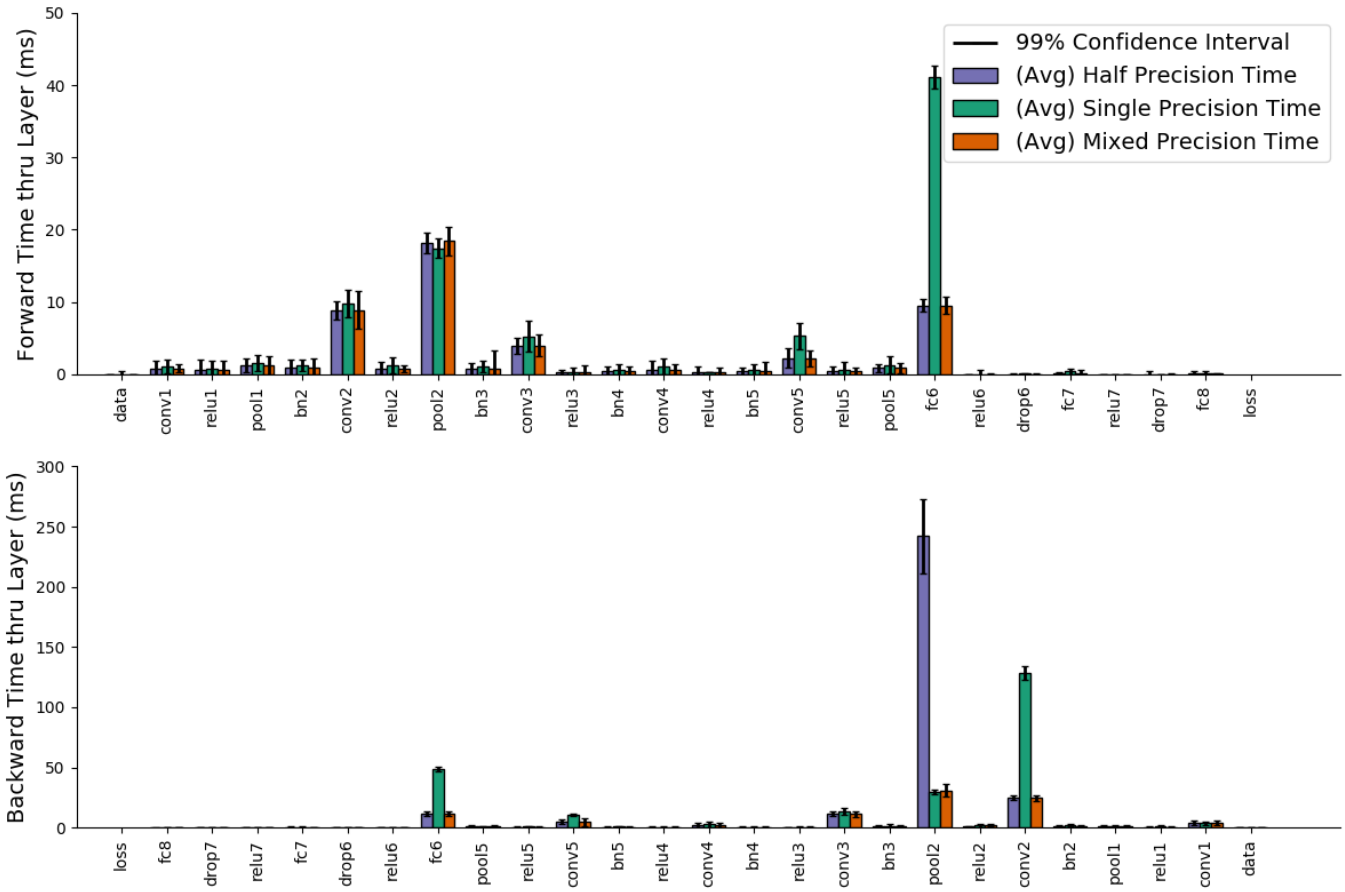

Fig. 9. Per-layer run times (forward and backward pass) when network is entirely half precision (purple)–expected to be fastest–entirely single precision (green), or when precision is specified layer-by-layer (i.e. mixed, orange/brown) to be the faster of half and single precision. In this case, most layers are faster in half precision as expected; however, the backwards pass of `pool2` is significantly slower when using half precision than using single precision.

important for a long-training network run (e.g., by reducing a training time of ten days to a training time of seven days), our ultimate goal is to incorporate this fine-grained mixed precision into a hyper-parameter optimization method to be deployed on high-performance computers. In hyper-parameter optimization on HPC, we expect to evaluate potentially hundreds of thousands of networks to find an optimal or close-to-optimal set of network hyper-parameters or network topology for a given application. In this case, the speedup provided by fine-grained mixed precision can increase the number of networks evaluated in a given time period resulting in better exploration of the search space and potentially finding better solutions in the same period of time as using pure single or pure half precision calculations. It is worth noting that even if the networks that see the most speedup because of this process are the least accurate networks produced, we must still evaluate those networks to determine what their accuracy is for the hyper-parameter optimization process. Thus, this approach benefits the hyper-parameter optimization process regardless.

Additionally, in this work we are explicitly benchmarking the layers using both half and single precision for each network. However, the forward and backward pass for each layer should simply be a function of its hyper-parameters and input size. Thus, after benchmarking a representative set of networks, we should be able to create a model that can predict forward and backwards pass times simply from the hyper-parameters. Once this model is created for a fixed hardware and software stack, the computation for selecting optimal precision in our fine-grained mixed precision approach would be negligible.

V. Acknowledgements

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Robinson Pino, program manager, under contract number DE-AC05-00OR22725.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[3] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528.

[4] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron *et al.*, "Theano: Deep learning on gpus with python," in *NIPS 2011, BigLearning Workshop, Granada, Spain*, vol. 3. Citeseer, 2011, pp. 1–48.

[5] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & precision," *arXiv preprint arXiv:1803.04014*, 2018.

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 2017, pp. 1–12.

[7] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[8] S. R. Young, D. C. Rose, T. Johnston, W. T. Heller, T. P. Karnowski, T. E. Potok, R. M. Patton, G. Perdue, and J. Miller, "Evolving deep networks using hpc," in *Proceedings of the 3rd Workshop on Machine Learning in HPC Environments*. ACM, 2017, p. 7.

[9] T. Johnston, S. R. Young, D. Hughes, R. M. Patton, and D. White, "Optimizing convolutional neural networks for cloud detection," in *Proceedings of the Machine Learning on HPC Environments*. ACM, 2017, p. 9.

[10] M. Coletti, "Ramifications of evolving misbehaving convolutional neural network kernel and batch sizes," in *Proceedings of the Machine Learning on HPC Environments*, 2018, p. 8.

[11] D. A. Martinez-Gonzalez and W. Brewer, "Deep learning evolutionary optimization for regression of rotorcraft vibrational spectra," in *Proceedings of the Machine Learning on HPC Environments*, 2018, p. 8.

[12] W. Zhang, K. Itoh, J. Tanida, and Y. Ichioka, "Parallel distributed processing model with local space-invariant interconnections and its optical architecture," *Applied optics*, vol. 29, no. 32, pp. 4790–4797, 1990.

[13] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[14] M. Harris, "New features in cuda 7.5," Jul 2015. [Online]. Available: https://devblogs.nvidia.com/new-features-cuda-7-5/

[15] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed Precision Training," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=r1gs9JgRZ

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Kara, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[17] "Tensor cores in nvidia volta gpu architecture." [Online]. Available: https://www.nvidia.com/en-us/data-center/tensorcore/