Analysis of the Time-To-Accuracy Metric and Entries in the DAWNBench Deep Learning Benchmark

Cody Coleman[*], **Daniel Kang**[*], **Deepak Narayanan**[*], **Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, Matei Zaharia**
Stanford DAWN Project

Abstract

Researchers have proposed hardware, software, and statistical optimizations to improve the computational performance of deep learning. While some of these optimizations keep model semantics *exactly* the same (e.g., using a NVIDIA K80 GPU vs. a P100 GPU), many also modify the semantics of the training procedure (e.g., reduced precision) and impact the final model's validation accuracy. Due to a lack of standard evaluation criteria that considers these trade-offs, it is difficult to compare optimizations. DAWNBENCH and MLPERF are benchmarks that use time-to-accuracy to compare the behavior of different techniques that improve deep learning computational performance. In this work, we investigate time-to-accuracy and analyze DAWNBENCH entries, including recent rolling submissions. We show that time-to-accuracy is stable and provide evidence that models optimized for time-to-accuracy generalize about as well as pre-trained models. Additionally, we find that the best entries, which can train a model on ImageNet to 93% top-5 accuracy in under 19 minutes, severely underutilize modern hardware.

1 Introduction

Optimizations in deep learning, including advances in software systems [1, 2, 3, 4, 5], hardware [6, 7, 8, 9], communication methods [10, 11, 12, 13], and training algorithms [14, 15, 16, 17, 18, 19, 20], are difficult to compare because of a lack of a standard evaluation criteria. Some optimizations preserve the statistical efficiency (number of iterations) of the underlying algorithm and model (e.g., replacing an NVIDIA K80 GPU with a P100 GPU can give a 4× speedup [21]) and are thus easy to evaluate. However, many optimizations sacrifice statistical efficiency for hardware efficiency (time needed for each iteration). On the hardware side, large minibatch training [15, 8], and reduced precision [3, 10, 22] can help better use hardware units and speed up "proxy" metrics such as time to process an epoch, but can prevent models from converging to the same accuracy. On the statistical side, techniques such as the Adam optimizer [17] were shown to accelerate the minimization of training loss ("time-to-training-loss") but can lead to models with lower final accuracy [23]. Proxy metrics like time-per-epoch or time-to-training-loss do not reason about both runtime and accuracy, making it hard to evaluate different optimizations.

To rectify this lack of standard evaluation criteria, DAWNBENCH[2] and MLPERF[3], two recent benchmarks targeting computational performance, use end-to-end training time to a specified validation accuracy level (called *time-to-accuracy*, or TTA) as their main metric. While several papers have used TTA for evaluation [24, 25, 15, 26, 27, 28], DAWNBENCH was the first multi-entrant *benchmark* to standardize TTA as "an objective means of normalizing across differences in computation frameworks, hardware, optimization algorithms, hyperparameter settings, and other factors that affect real-world performance" [29]. During the competition, Google, Intel, fast.ai, and others submitted optimized implementations that achieved a two orders of magnitude improvement in training time for ImageNet

[*] Equal Contribution

[2] `https://dawn.cs.stanford.edu/benchmark/`

[3] `https://mlperf.org/`

Preprint. Work in progress.

and an order of magnitude improvement in cost over the initial seed entries, which were created from multi-GPU examples in MXNet [30] and TensorFlow [1]. Building on DAWNBENCH's success, MLPERF [31] has also adopted TTA as its primary metric, while expanding the set of tasks evaluated.

Despite the improvements seen in DAWNBENCH and the now widespread use of TTA, many questions remain about TTA as a metric. For example: is TTA stable? Do models optimized for TTA still generalize well? And how efficient are these models, i.e., how close are we to fully using modern hardware, given all the work that has gone into preparing entries for these competitions?

We analyze TTA using the latest DAWNBENCH submissions, and find that:

- For image classification tasks, TTA is stable with a low coefficient of variation, i.e., multiple runs of entries reach the target accuracy in nearly the same time across runs. However, certain optimizations, such as cyclic learning rates [32] and progressive resizing [33, 34] increase the coefficient of variance, and sometimes prevent convergence to the same accuracy.
- Models tuned for TTA generalize as well as comparable pre-trained models to unseen data.
- Hardware is underutilized by the top submissions. Our experiments show that both hardware and statistical efficiency for deep learning workloads do not scale well, with up to a 2× gap from perfect linear scaling, and that entries underutilize modern hardware by up to 10× (§ 4). Specifically, hardware utilization for entries that run a ResNet-50 model on NVIDIA's V100 GPUs is low due to memory-bound layers, high communication overhead, and poor utilization of Tensor Cores.

2 Benchmark Summary

Overview. DAWNBENCH evaluates the time and cost (in USD) of image classification on ImageNet and CIFAR10, and question answering on SQuAD. DAWNBENCH had four metrics: training time to a specified validation accuracy, cost of training to that accuracy for submissions that use hardware in the public cloud, average latency of performing inference on a single item (image or question), and average cost of inference for 10,000 items.

Entries were required to submit a description of their submission and validation accuracy after every epoch. While source code was optional, every submission for image classification included a link to all code needed to reproduce runs, assuming access to the appropriate hardware. For question answering on SQuAD, some submissions did not include code until well after the DAWNBENCH deadline; because of this and the general lack of submissions, we focus exclusively on ImageNet and CIFAR10 *training* submissions in this paper.

Summary of Entries. The entries spanned a range of hardware platforms, software platforms, and models. We summarize the entries in Table 1. Notably, the entries spanned GPUs, TPUs, and CPUs on the hardware side, TensorFlow [1], PyTorch [35], and Caffe [5] on the software side, and used both standard and learned model architectures.

ImageNet. DAWNBENCH set a top-5 accuracy target of 93% for ImageNet. The fastest entry overall trained ResNet-50 on 16 multi-GPU servers (with 8 V100 GPUs) in 18 minutes and 6 seconds, which is faster than many published large-scale results [15, 24, 36, 37]. While faster results exist, they either do not achieve the same high level of accuracy [25] or use a private on-premise cluster [27] with faster network interface cards compared to the public cloud. The fastest TPU submission trained ResNet-50 on half of a TPUv2 pod in 30 minutes, and the fastest CPU submission trained ResNet-50 on 128 machines with 36 cores each in 3 hours and 26 minutes. The cheapest entry trained ResNet-50 on 4 multi-GPU (with 8 V100 GPUs) servers on Amazon Web Services for $48.48, followed by an AmoebaNet [38] model that trained for $49.30 on a Google Cloud TPU.

CIFAR10. DAWNBENCH set a top-1 accuracy target of 94% for CIFAR10. The fastest entry trained a custom Wide ResNet [39] architecture in less than 3 minutes on 8 NVIDIA V100 GPUs. The cheapest entry trained the same custom Wide ResNet for $0.26 on 1 NVIDIA V100 GPU.

Achieving these fast TTAs for both ImageNet and CIFAR10 come from a range of optimizations, including mixed-precision training [40], large minibatch training [15], progressive resizing of images [33, 34], novel model architectures [38], and faster hardware [21, 8].

Hardware	# of entries (ImageNet)	# of entries (CIFAR10)	Framework	# of entries (ImageNet)	# of entries (CIFAR10)
GPU	6	6	TensorFlow	8	2
TPU	6	0	PyTorch	4	4
CPU	3	0	Caffe	3	0

Table 1: Overview of hardware platform and software framework for each DAWNBench submission.

Entry name	Coeff. of variation	Frac. of runs
Wide ResNet-34, 8xV100	3.8%	50%
Wide ResNet-34, 1xV100	2.9%	70%
ResNet-18, 1xV100	1.4%	90%

Table 2: Coefficient of variation and fraction of runs that reached the desired target accuracy of the top single server blade entries for image classification on CIFAR10 (10 runs).

3 Analysis of Time-to-Accuracy

We evaluate the TTA metric using publicly available code from DAWNBench's top submissions. We show that: a) TTA is generally stable with a low coefficient of variation over several runs with fixed hyperparameters, b) models optimized for TTA generalize as well as standard models.

3.1 Variability of time-to-accuracy

We measured the stability of TTA by running several of the top training time entries for CIFAR10 and ImageNet several times, with the same hyperparameters. We use the coefficient of variation, defined as the ratio of the standard deviation to the mean, as a measure of stability. We do not use the same random seed, so the data may be traversed in different orders, possibly leading to convergence to different local minima. We note that model training does not contain any data-dependent optimizations, which means time-per-epoch is stable. As shown in Table 3, the coefficient of variation of TTA is at most 4.5% for entries that do not use novel statistical optimizations, but 12.2% for all entries. This indicates that TTA is largely stable to the randomness inherent to training deep networks, but that some statistical optimizations can change the number of iterations needed to reach the target accuracy, thus increasing variance.

In addition, several entries failed to consistently achieve the given accuracy threshold. In particular, the cyclic learning rates used by the top two CIFAR10 entries appear to make validation convergence less robust [32]. Cyclic learning rates prevent validation convergence on ImageNet and were not included in any top submissions [41].

3.2 Generalization of Optimized Models

We scraped and labeled a set of 2,864 images from Flickr. The images were scraped based on the WordNet keywords associated with each class in the ImageNet dataset. The top five images based on relevance were shown to a human labeler and labeled correct or incorrect. To ensure no overlap with ImageNet, only images posted after January 1st, 2014 were used. While the images only spanned 886 (out of 1000) classes, we believe they reflect a reasonable distribution.

For both the pre-trained ResNet-50 (PyTorch provided) and DAWNBench entries optimized for TTA, we computed the top-5 accuracy on the Flickr images. The results are summarized in Table 4. As shown, the models optimized for TTA achieve nearly the same accuracy or higher than the pre-trained ResNet-50, indicating that optimizing for TTA does not sacrifice generalization performance.

4 Hardware Utilization and Scaling

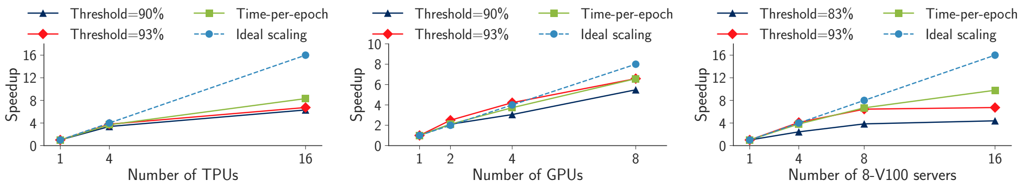
In this section, we analyze how well submissions scale to multiple servers. We also perform a roofline analysis [42] that shows that despite near state-of-the-art performance, many submissions still *severely* underutilize the available hardware, and subsequently analyze why some of these entries underutilize the NVIDIA V100 GPU.

Entry name	Coeff. of variation	Frac. of runs	Entry name	Coeff. of variation	Frac. of runs
ResNet-50, 8xV100	5.3%	80%	ResNet-50, 1xTPU	4.5%	100%
ResNet-50, 32xV100	11.2%	60%	AmoebaNet-D, 1xTPU	2.3%	100%
ResNet-50, 64xV100	9.2%	100%	ResNet-50, 1/2 TPU Pod	2.5%	100%
ResNet-50, 128xV100	12.2%	100%			

Table 3: Coefficient of variation and fraction of runs that reached the target accuracy of the top entries for image classification on ImageNet (5 runs). We also include the coefficient for 4 runs of 1/2 a TPU Pod for ResNet-50.

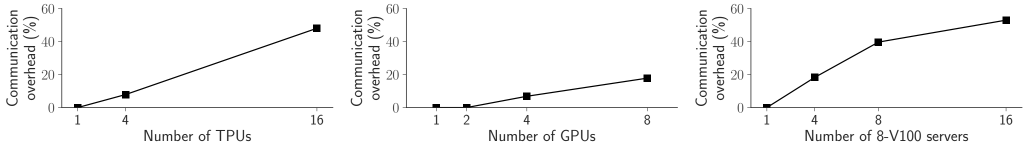
Model	Accuracy (unseen data)	Model	Accuracy (unseen data)
ResNet-18 (p)	89.5%	ResNet-50, 8xV100	91.9%
ResNet-50 (p)	92.2%	ResNet-50, 32xV100	91.3%
ResNet-152 (p)	93.2%	ResNet-50, 64xV100	91.5%
ResNet-50, 1xTPU	92.6%	ResNet-50, 128xV100	91.3%
		AmoebaNet-D, 1xTPU	91.3%

Table 4: Performance of pre-trained models and models optimized for TTA on unseen data. The models optimized for TTA perform nearly as well as or better than the pre-trained model. (p) refers to pretrained. We expect the pretrained ResNet-18 and ResNet-152 to be lower and upper bounds on generalization performance.



(a) AmoebaNet on TPUs, TPU pod. **(b)** ResNet-50, 8-V100 server. **(c)** ResNet-50, 8-V100 servers.

Figure 1: Speedup with respect to a single worker vs. number of workers for ImageNet entries, one on a TPU pod, another among V100 GPUs on a single server with 8 V100 GPUs, and a third on multiple 8-V100 servers. As the number of workers increases, the scaling performance drops off (over 2× gap from ideal scaling).



(a) AmoebaNet on TPUs, TPU pod. **(b)** ResNet-50, 8-V100 server. **(c)** ResNet-50, 8-V100 servers.

Figure 2: Percentage of time spent communicating vs. worker count for three ImageNet models, one on a TPU pod, another among V100 GPUs on a 8-V100 server, and a third on multiple 8-V100 servers. Within a 8-V100 server, communication overhead is low (17.82%), but cross-server communication is expensive (53%).

4.1 Distributed Training

Distributed training has become common because of advances in large minibatch training [15], where learning rates are slowly increased over the course of training, and learning rates are scaled linearly with the global minibatch size. Large minibatch training enables much faster time-per-epoch when parallelized while achieving similar accuracies to small minibatches.

DAWNBENCH entries used minibatch sizes an order of magnitude larger than the minibatch size used in the original ResNet paper [43] for distributed training. Figure 1b shows the speedups when scaling the training of a ResNet-50 model on up to 8 GPUs in a 8-V100 server. Both time-per-epoch and TTA for the ResNet-50 model scale much better on the 8-V100 server, partially because of less communication overhead at high worker counts (Figure 2b). Figure 1c shows the speedups when scaling ResNet-50 training up to 16 servers which have 8 V100 GPUs each. Time-per-epoch and TTA scale worse than *within* a 8-V100 server. Figure 2 shows the communication overhead as the number of workers increase; we see that within a 8-V100 server, communication overhead is much lower (17.82%) compared to across servers (53%), since NVIDIA's nvlink interconnect has far higher bandwidth than the 25Gbps provided by Amazon EC2 across `p3.16xlarge` instances. For cross-machine communication, we observed that the peak network bandwidth used (5.95 Gbps) is

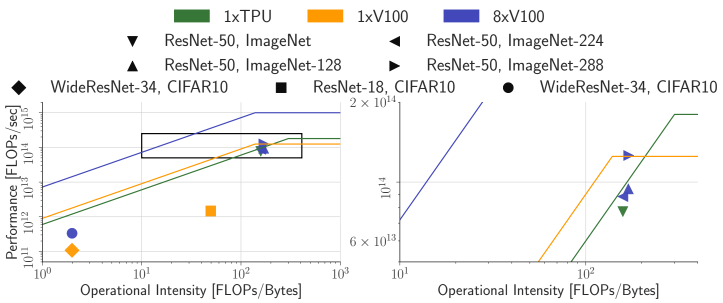


Figure 3: Roofline models for the various entries submitted to DAWNBENCH. All entries under-utilize the hardware resources, by up to 10×. The right figure is a zoomed-in version of the box in the left figure.

actually significantly lower than the advertised bandwidth, indicating an inefficiency in the framework and/or cloud infrastructure as well.

While several DAWNBENCH entries corroborated the original findings that used ResNet-50 [15], AmoebaNet, a learned architecture from Google, also used large minibatches to utilize multiple TPUs. However, we found that the time-per-epoch does not scale linearly with the number of workers for the AmoebaNet model in the TPU pod, which consists of 64 TPUs. We show the speedup relative to one worker for two different accuracy thresholds, as well as the average time to compute a single epoch, for different numbers of workers for an AmoebaNet model trained in a TPU Pod on the ImageNet dataset in Figure 1a. We see that TTA scales even worse, since a greater number of epochs are needed to converge to the same accuracy target.

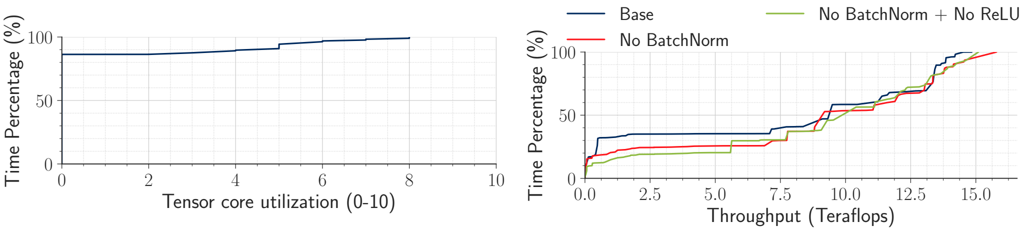
Discussion. These results suggest that despite the work in scaling image classification to multiple nodes [15, 28], communication remains a bottleneck, for large machine counts and for certain models in public cloud deployments. The work by Goyal et al. [15] shows far better scaling than we have observed in the DAWNBENCH entries; we believe this is due to the fact that the results presented in this paper use faster V100 GPUs (compared to P100 GPUs), and slower network interfaces (up to 25 Gigabits/second on AWS compared to 50 Gigabits/second in a private Facebook cluster).

To address this, highly optimized communication libraries like Horovod [44] have been developed. Other work [45] has explored techniques to reduce the amount of data sent over the network. However, these techniques need to be evaluated on more models and in more hardware settings for widespread adoption. Integration into widely-used deep learning frameworks like PyTorch and TensorFlow would also help with usability. Additionally, exploring parallelization schemes other than data parallelism that do not require all-to-all communication patterns among workers could be helpful.

4.2 Single-worker analysis

Roofline Analysis. To further study the hardware performance of various entries, we use the roofline model [42], which can highlight causes of performance bottlenecks. The roofline model plots computational throughput (in floating point operations per second) against the operational intensity of the application (number of floating-point operations performed per DRAM byte accessed). Applications with high operational intensity are "compute-bound" (the flat line in Figure 3) and bottlenecked on the device's computation units, while applications with low operational intensity are "memory-bound" (the slanting line in Figure 3). Each point in Figure 3 represents a DAWNBENCH entry. For entries which use progressive image resizing [33, 34], where different image sizes are used through training, we show each image size used. Operational intensities and throughputs are approximated by instrumenting model code, and using command-line utilities like `nvprof`.

We observe that all entries *severely* underutilize the available compute resources — each plotted point achieves a throughput significantly lower than the peak device throughput. All analyzed CIFAR10 models operate in low operational intensity regimes, partially because of the small input size (32×32) and small associated operations like matrix multiplications and convolutions. The ImageNet models do a better job of utilizing the underlying hardware, but still are as much as a factor of 10× away from peak device throughput for the GPUs. The TPUs are much better utilized, with a utilization of 45% for the ResNet-50 model, and 31% for the AmoebaNet model (not pictured in Figure 3).



(a) Tensor Core utilization

(b) Per-kernel throughput

Figure 4: Left: CDF of tensor core utilization for ResNet50 model trained with `fp16` precision. About 85% of time is spent on kernels that don't use the NVIDIA Tensor Cores *at all*, and no kernel achieves full utilization. **Right:** CDF of per-kernel throughput for ResNet50 models trained with `fp32` precision. The CDF is computed using percentage of time spent executing each GPU kernel. A standard ResNet-50 model spends ~40% time in low-throughput kernels (< 6 TFlops). Removing BatchNorm from the ResNet50 model decreases the percentage of time in low-throughput kernels to ~30%; removing the ReLU layers decreases this further.

Single-V100 PyTorch entries. To investigate the underutilization on the V100 GPUs, we measure the Tensor Core utilization of each GPU kernel and `fp32` throughput in PyTorch's implementation of ResNet-50. The V100s are advertised to deliver 15.7 Teraflops of `fp32` arithmetic, while the Tensor Cores are advertised to deliver 125 Teraflops of half-precision arithmetic [46]. Figure 4a shows that the GPU kernels that take the majority of the `fp16` precision time use the Tensor Cores poorly, with a time-averaged Tensor Core utilization of 0.71 (on a scale of 1-10 as reported by `nvprof`).

Training the same model even with standard `fp32` precision only achieves a throughput of 7.6 Tflops (compared to advertised device throughput of 15.7 Tflops). We observe that this is largely due to memory-bound kernels like batch norm and ReLU, that take a significant percentage of total runtime. This is illustrated in Figure 4b, which show that a non-trivial number of kernels underutilize the GPU, and that removing batch norm and ReLU layers improves `fp32` throughput by about 20%.

Discussion. Compilers for deep learning like TVM [47] and XLA [48] try to automatically generate code given a higher-level description of the deep learning computation being performed. Optimizations like loop and kernel fusion can help reduce the impact of memory-bound kernels by reducing the number of DRAM reads and writes made. For example, consider a program that performs the forward pass of a batch norm followed by the forward pass of a ReLU. Naively, this code could be written as two `for` loops – one to compute the batch norm, and another to compute the ReLU. If `y` is the output of the batch norm, and `z` is the output of the ReLU, then a DRAM write is performed for every `y[i]` and `z[i]`, and a DRAM read of `y[i]` is performed to compute every `z[i]`.

However, for training, we could optimize this to save on DRAM reads of `y[i]` by writing the intermediate result to a local variable instead. For inference, we could optimize away the writes to `y[i]` entirely (since the intermediate `y[i]` are not needed for a subsequent backward pass). Saving on DRAM reads and writes is highly useful on GPUs since compute throughput is roughly two orders of magnitude faster than DRAM read and write throughput.

In addition, we believe that co-design of model architectures with modern hardware could be useful as well. For example, as we have shown, the batch norm and ReLU operations are memory-bound. It may be possible to develop alternatives to these operations that are less memory-bound, but still provide similar statistical effects, resulting in faster end-to-end training.

5 Conclusion

By analyzing the top DAWNBENCH entries, we evaluated both TTA as a metric for measuring deep learning system performance on image classification, and the many optimizations that led to considerable speedups for ImageNet and CIFAR10 training time. Reproducing and repeating runs of the top entries revealed TTA as a stable metric with a low coefficient of variation despite the randomness in training. Even though TTA is sensitive to the threshold, the high accuracy target prevented optimizations that hurt final validation convergence and generalization, but provided room for hardware, software, and statistical optimizations. All the entries, however, still underutilized the available compute resources, leaving opportunities for further research.

Acknowledgements

We thank Jeremy Howard, the Google Cloud TPU team (including Sourabh Bajaj, Frank Chen, Brennan Saeta, and Chris Ying), and the many other teams that submitted to DAWNBENCH. We thank Juan Manuel Camacho, Shoumik Palkar, Kexin Rong, Keshav Santhanam, Sahaana Suri, Pratiksha Thaker, and James Thomas for their assistance in labeling. We also thank Amazon and Google for cloud credits. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Intel, Microsoft, NEC, Teradata, SAP, and VMware—as well as DARPA under No. FA8750-17-2-0095 (D3M), industrial gifts and support from Toyota Research Institute, Keysight Technologies, Hitachi, Northrop Grumman, NetApp, and the NSF under grants DGE-1656518 and CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv:1410.0759*, 2014.

[3] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *OSDI*, volume 14, pages 571–582, 2014.

[4] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[5] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[6] Doug Burger. Microsoft unveils Project Brainwave for Real-time AI. *Microsoft Research, Microsoft*, 22, 2017.

[7] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Networks. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[8] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.

[9] Dexmont Pena, Andrew Forembski, Xiaofan Xu, and David Moloney. Benchmarking of CNNs for Low-Cost, Low-Power Robotics Applications. 2017.

[10] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. Understanding and Optimizing Asynchronous Low-precision Stochastic Gradient Descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 561–574. ACM, 2017.

[11] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. Addressing the Straggler Problem for Iterative Convergent Parallel ML. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 98–111. ACM, 2016.

[12] Feng Niu, Benjamin Recht, Christopher Re, and Stephen Wright. Hogwild: A Lock-free Approach to Parallelizing Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[13] Ce Zhang and Christopher Ré. Dimmwitted: A Study of Main-memory Statistical Analytics. *Proceedings of the VLDB Endowment*, 7(12):1283–1294, 2014.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[15] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and < 0.5 MB Model Size. *arXiv preprint arXiv:1602.07360*, 2016.

[17] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *ICLR*, 2015.

[18] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast Large-scale Optimization by Unifying Stochastic Gradient and Quasi-Newton Methods. In *International Conference on Machine Learning*, pages 604–612, 2014.

[19] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting. *arXiv preprint arXiv:1706.06197*, 2017.

[20] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the Importance of Initialization and Momentum in Deep Learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

[21] J Murphy. Deep Learning Benchmarks of NVIDIA Tesla P100 PCIe, Tesla K80, and Tesla M40 GPUs, 2017.

[22] Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149*, 2015.

[23] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

[24] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, page 1. ACM, 2018.

[25] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[26] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pages 583–598, 2014.

[27] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[28] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[29] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. *NIPS ML Systems Workshop*, 2017.

[30] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[31] MLPerf. `https://mlperf.org/`, 2018.

[32] Leslie N Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates. *arXiv preprint arXiv:1708.07120*, 2017.

[33] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced Deep Residual Networks for Single Image Super-Resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, volume 1, page 3, 2017.

[34] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *arXiv preprint arXiv:1710.10196*, 2017.

[35] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[36] Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. Scale out for large minibatch sgd: Residual network training on imagenet-1k with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291*, 2017.

[37] Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. Powerai ddl. *arXiv preprint arXiv:1708.02188*, 2017.

[38] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized Evolution for Image Classifier Architecture Search. *arXiv preprint arXiv:1802.01548*, 2018.

[39] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. *arXiv preprint arXiv:1605.07146*, 2016.

[40] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, et al. Mixed Precision Training. *arXiv preprint arXiv:1710.03740*, 2017.

[41] Jeremy Howard. Training ImageNet in 3 hours for $25; and CIFAR10 for $0.26. `http://www.fast.ai/2018/04/30/dawnbench-fastai/`, 2018.

[42] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[44] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[45] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018.

[46] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. *arXiv preprint arXiv:1803.04014*, 2018.

[47] TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA, 2018. USENIX Association.

[48] Tensorflow xla overview. `https://www.tensorflow.org/performance/xla`, 2017.