DAWNBench: An End-to-End Deep Learning Benchmark and Competition

Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, Matei Zaharia
Stanford DAWN Project
http://dawn.cs.stanford.edu/benchmark

Abstract

Despite considerable research on systems, algorithms and hardware to speed up deep learning workloads, there is no standard means of evaluating end-to-end deep learning performance. Existing benchmarks measure proxy metrics, such as time to process one minibatch of data, that do not indicate whether the system as a whole will produce a *high-quality* result. In this work, we introduce DAWNBench, a benchmark and competition focused on *end-to-end* training time to achieve a *state-of-the-art* accuracy level, as well as inference time with that accuracy. Using time to accuracy as a target metric, we explore how different optimizations, including choice of optimizer, stochastic depth, and multi-GPU training, affect end-to-end training performance. Our results demonstrate that optimizations can interact in non-trivial ways when used in conjunction, producing lower speed-ups and less accurate models. We believe DAWNBench will provide a useful, reproducible means of evaluating the many trade-offs in deep learning systems.

1 Introduction

Deep learning methods are effective but computationally expensive, leading to a great deal of work to optimize their computational performance. Researchers have proposed new software systems [29, 1, 8, 12, 9, 46], training algorithms [33, 48, 45, 28, 44, 18, 27, 50, 42, 43, 14], communication methods [39, 49, 11, 9, 12, 22] and hardware [20, 30, 37, 19, 7, 21] to decrease this cost. However, despite these significant advances, it is difficult to measure or compare the utility of these results due to a lack of standard evaluation criteria. Most existing benchmarks for deep learning performance [16, 10, 5, 2, 41, 8, 3] only measure proxy metrics such as the time to process one minibatch of data. In reality, deep learning performance is far more complex. In some settings, techniques such as using larger batch sizes [18, 30], reduced precision [21, 9, 25, 11] and asynchronous updates [39, 12, 9, 49] can stop an algorithm from converging to a good result, or increase the time to do so. Moreover, these approaches interact in nontrivial ways and may require updating the underlying optimization algorithm [33, 18, 36], further affecting end-to-end performance.

This lack of standard evaluation criteria results in a set of poorly-understood trade-offs. For example, minimal effort back propagation delivers a 3.1x speed up over back propagation on MNIST [43]. Using 8-bit precision gives a 3x speed up on MNIST [11]. Does combining minimal effort back propagation with 8-bit precision give a 9.3x speed up? Would that speed translate to a larger model on a dataset like ImageNet, and combine with accurate, large minibatch SGD [18] to train an ImageNet model in 7 minutes? Currently, these questions can only be answered via tedious and time-consuming experimentation. Moreover, which previous techniques should one build on when evaluating the efficiency of a new optimization?

To provide an objective means of quantifying end-to-end deep learning performance, we introduce DAWNBench, an open benchmark and competition for end-to-end deep learning training and inference. Instead of simply measuring time per iteration or throughput, DAWNBench measures *end-to-end*

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Tasks	Metrics	Datasets
Image classification	Training time Training cost	ImageNet CIFAR10
Question answering	Inference latency Inference cost	SQuAD

Table 1: Dimensions evaluated in the first version of DAWNBench. All metrics are for a near-state-of-the-art accuracy.

performance of training (e.g., time, cost) and inference (e.g., latency, cost) at a specified accuracy level. This provides an objective means of normalizing across differences in computation frameworks, hardware, optimization algorithms, hyperparameter settings, and other factors that affect real-world performance. Our initial release of DAWNBench provides end-to-end learning and inference tasks including image classification on CIFAR10 [34] and ImageNet [40], and question answering on SQuAD [38], and reference implementations for each task.

In this paper, we present DAWNBench's benchmark specification and goals, and show how one can use DAWNBench to perform a simple performance study that measures the impact of choice of optimizer, batch size, multi-GPU training, and stochastic depth [26]. Over time, and with community input, we plan to expand DAWNBench's scope to include both additional benchmark tasks (e.g., segmentation, machine translation, video classification) and metrics (e.g., energy, sample complexity).

2 Benchmark Overview and Goals

DAWNBench evaluates deep learning systems on different tasks based on several metrics, using multiple datasets. The benchmark allows innovation in software, algorithms, communication methods, etc. By only specifying the task, DAWNBench also allows experimentation of new model architectures and hardware. In the initial release, we seed entries for two tasks: image classification on CIFAR10 and ImageNet, and question answering on SQuAD, and evaluate on four metrics: training time to a specified validation accuracy, cost (in USD) of training to a specified validation accuracy using public cloud instances, average latency of performing inference on a single item (image or question), and average cost of inference for 10,000 items (Table 1). We also provide reference implementations and seed entries, implemented in two popular deep learning frameworks, PyTorch and TensorFlow. These reference implementations were collected and adapted from official repositories on Github, and produce accuracy numbers on par with those reported in the original research papers [24], while also conforming to the various performance recommendations published with these frameworks [17].

DAWNBench is not the first benchmark to compare and evaluate deep learning systems on standard tasks; previous efforts include Baidu DeepBench [5], Fathom [2], and TensorFlow Benchmark [16]. Our benchmark differs in two key aspects. First, DAWNBench focuses on end-to-end performance. Many prior benchmarks use the time needed to train on a single minibatch of data as the key metric, while disregarding the resulting accuracy of the trained model [2, 16, 10, 41, 4]. Other benchmarks [2, 5] focus on timing individual low-level operations (e.g. convolutions, matrix multiplications) utilized in deep learning computations. DAWNBench, instead, measures time to a pre-specified (and high) level of accuracy, taking into account both hardware and statistical performance.

Second, we view DAWNBench as open and evolving. Prior benchmarks were often defined at a snapshot in time, using fixed model architectures and tasks. However, advances in deep learning have quickly invalidated architectures. For example, recent approaches in machine translation using recurrent layers have been quickly superseded by convolutional and attention layers [13, 47]. Deep learning is a rapidly evolving field, and needs benchmarks that adapt at the same pace. We plan to allow DAWNBench to grow and evolve, and hope to foster a community discussion around advances in the field, including new tasks.

3 Early Results

To demonstrate the utility of an end-to-end benchmark, we present DAWNBench results that aim to answer three questions: (1) Is training time to a specified validation accuracy a useful metric to evaluate deep learning systems? (2) Do different optimizations in deep learning actually compose? (3) Using DAWNBench's seed entries, what trade-offs between training time, training cost, and inference latency do different software frameworks and hardware present?

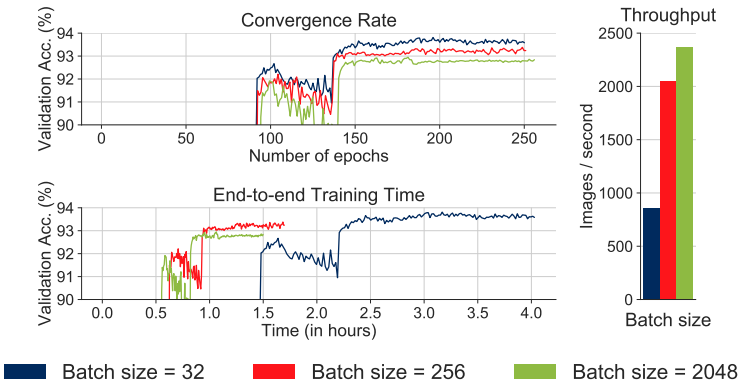


Figure 1: Effect of minibatch size on convergence rate, throughput, and end-to-end training time of a ResNet56 CIFAR10 model on a P100. Learning rates are tuned as per [18].

We follow the same basic training procedure as the original ResNet paper [24] for all experiments in this section. For a batch size of 128, we use SGD with a weight decay of 0.0005 and momentum of 0.9. We use the weight initialization procedure from [23] and use Batch Normalization [28] without dropout. We warm-up training with a learning rate of 0.01 for five epochs, then proceed with an initial learning rate of 0.1 for 90 epochs; we decay the learning rate by a factor of 10 every 45 epochs thereafter, and terminate training after 185 epochs. We follow the same data augmentation process as [24]. For larger batch sizes, we linearly scale the initial learning rate according to [18], using a base learning rate of 0.1 (corresponding to a batch size of 128).

Unless stated otherwise, the K80 machines used in this section are Google Cloud Engine instances based on the `n1-standard-8` instance type (8 vCPUs, 30 GB memory), with Haswell CPUs and Google's standard HDD for persistent storage. The P100 machines used have 512 GB of memory and 28 CPU cores, and are in a private cluster. We used TensorFlow 1.2 and PyTorch 0.1.12, compiled from source with CUDA 8.0 and CuDNN 5.1. DAWNBench does not currently include distributed training results, but we plan to explore this as future work, and also welcome outside submissions. All experiments in this section compute validation accuracy on the CIFAR10 test set (10,000 images) [34].

3.1 Evaluating Time to Accuracy for Various Minibatch Sizes

To illustrate the value of DAWNBench's end-to-end performance metric, we study how minibatch size impacts *both* the convergence rate and hardware performance (FLOPS) of a deep learning workload. Prior work [32, 18, 15, 6, 35] has shown that picking a minibatch size too small or too large can lead to poor convergence, i.e. minibatch size affects convergence. Additionally, larger minibatch sizes better saturate hardware execution units [6, 15]. In choosing the minibatch size that minimizes total time to a target accuracy, we must balance these two factors.

As we show in Figure 1, for a ResNet56 model trained on the CIFAR10 dataset on a Nvidia P100 GPU, a minibatch size of 32 produces the best convergence rate (least number of epochs to highest accuracy), and a minibatch size of 2048 produces the best throughput (number of images processed divided by total time taken). A minibatch size of 256 represents a reasonable trade-off between convergence rate and throughput. A minibatch size of 256 reaches an accuracy of 93.38%, which is only 0.43% less than the maximum accuracy achieved with a minibatch size of 32, in 1.9x less time. Benchmarks that focus exclusively on convergence rate and throughput are unable to surface these practical trade-offs for optimizations even as simple as modifying the minibatch size.

3.2 Composing Optimizations

To show how DAWNBench can be used for performance studies in practice, we run a small study that evaluates the effect of composing several different optimizations for training Convolutional Neural Networks (CNNs) on the CIFAR10 dataset. Evaluating deep learning systems is challenging, in part because it's unclear how different optimizations interact with each other. Given the complexity of deep learning systems and plethora of hyperparameters involved, seemingly complementary optimizations can interact in non-trivial ways and stop an algorithm from converging to a good result, or increase the time to do so.

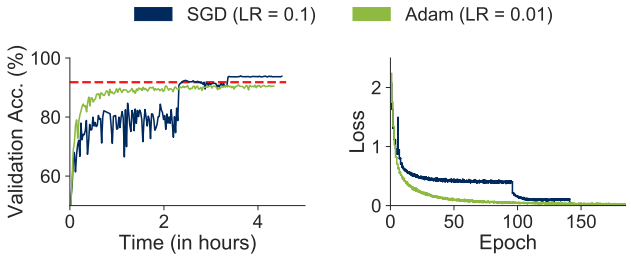


Figure 2: Impact of different optimizers while training a ResNet56 model on CIFAR10. The graph on the left plots the top-1 validation accuracy with respect to time, and the graph on the right plots a rolling average of the loss with respect to the epoch. We see that Adam initially outperforms SGD with momentum, but falls short around epoch 100 (~2 hours in the graph on the left), and does not reach a validation accuracy of 93%.

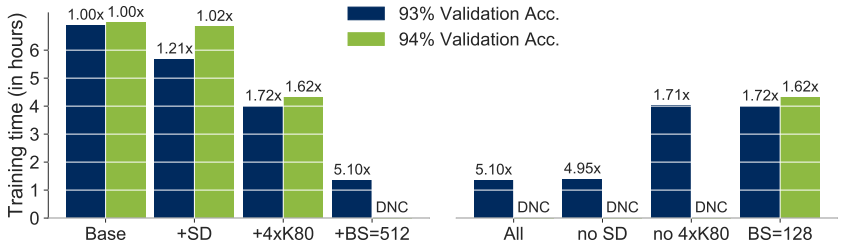


Figure 3: Factor analysis for training a ResNet110 model on CIFAR10 with stochastic depth (SD), 4 Nvidia K80 GPUs on a single node (4xK80), and a minibatch size (BS) of 512. Cumulatively enabling each optimization reduces the time to 93% top-1 accuracy, but combined, the model does not converge (DNC) to the 94% accuracy threshold. By removing the larger minibatch size, the model reaches the higher accuracy target.

Here, we demonstrate that different optimizations in fact *do not compose* while training a ResNet110 model on CIFAR10. Our goal here is not to provide blanket statements about any particular training method. Instead, we showcase the trade-offs and interactions between methods and their non-trivial impact on end-to-end accuracy and performance. We consider the following three optimizations,

- Adam, an adaptive optimizer for gradient descent, which reports considerable speed-ups over other adaptive optimizers when training CNNs on CIFAR10 [33].
- Single-node multi-GPU training, using 4 GPUs in two different settings: 1) With the same minibatch size (128) as that used in the baseline approach, but distributed across the 4 GPUs 2) With effectively a minibatch size multiplied by 4, where every GPU is given a minibatch size of 128. For the larger minibatch size, we tune the learning rate as per [18].
- Stochastic Depth [26], a technique that can be thought of as a form of regularization similar to dropout. Entire layers are randomly dropped during training, helping to regularize against co-adaptation, while still using the full network to perform inference. Stochastic Depth reports improvements in both training time and accuracy.

Adam. After evaluating each optimization in isolation, we discovered Adam with multiple initial learning rates (0.0001, 0.001, 0.01, 0.1) was unable to come close to the maximum validation accuracy we were able to achieve with our baseline. Figure 2 shows the best performance using Adam (learning rate = 0.01) compared to SGD with momentum using the standard learning rate schedule from [24]. Despite the better training loss and faster initial convergence on both the training and validation dataset, Adam's promising performance plateaued well below SGD. Because Adam failed to reach a reasonable validation accuracy in isolation, we eliminated it from the remainder of our factor analysis. This shows that training loss is only a proxy for accuracy and generalization, and that "time to an accuracy threshold" is a valuable metric when evaluating deep learning systems.

Single-node Multi-GPU Training. We observe that the other two optimizations each provide a reasonable speed-up, but the net speed-up from applying all the optimizations at once is less than the product of the individual speed-ups as shown Figure 3. All speed-ups and accuracy thresholds are relative to the performance of ResNet110 on 1 Nvidia K80 GPU with a minibatch size of 128 and initial learning rate of 0.1. The top accuracy ResNet110 achieved was 94.33%. Using this as a baseline, we chose two threshold points, 93% and 94% top-1 accuracy, representing different points in the trade-off space between

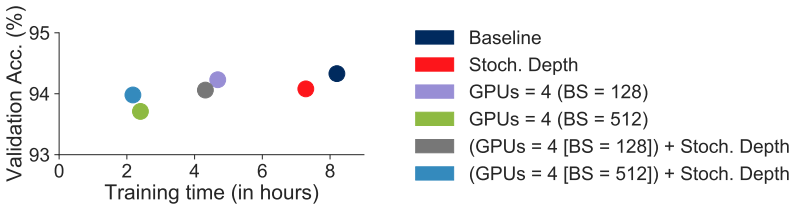


Figure 4: Comparison of maximum top-1 validation accuracy and training times of different combination of optimizations. While reducing training time, each optimization decreases the maximum accuracy.

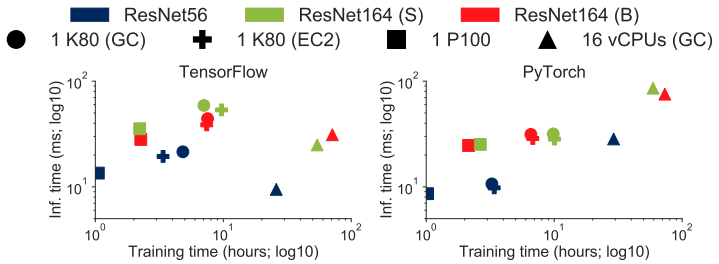


Figure 5: Inference time vs. training time to 93% validation accuracy, for different hardware, frameworks, and model architectures in DAWNBench's seed entries. ResNet164 (S) uses a simple building block, while (B) uses a bottleneck building block. Amazon EC2 instances use a `p2.xlarge` instance type (4vCPUs, 61 GB memory).

training time and accuracy. 94% represents human-level accuracy on the CIFAR10 dataset [31]. 93%, while considerably lower, helps demonstrate both the magnitude of speed-up possible when accuracy constraints are relaxed, and the difficulty of achieving human-level or super human-levels of accuracy.

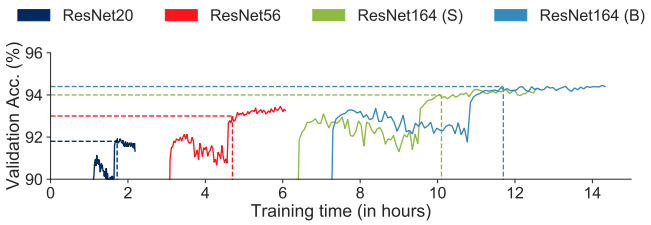
Scaling the number of GPUs while keeping the minibatch size the same produces a 1.68x and 1.59x speed-up to reach 93% and 94% accuracy respectively, compared to the baseline. A larger batch size of 512 reached an accuracy of 93% much faster (speed-up of 4.95x over baseline), but failed to reach the higher threshold. Interestingly, the speed-up in terms of time to 93% is larger than the increase in throughput alone (3.57x). In addition to scaling the learning rate according to [18], we also tried a batch size of 512 with multiple initial learning rates (0.1, 0.2, 0.3) with the same learning rate decay schedule. Surprisingly, the learning rate of 0.1 outperformed the linearly scaled learning rate of 0.4 on CIFAR10 in terms of maximum accuracy. We thus used a learning rate of 0.1 for all experiments with a batch size of 512 in Figure 3.

Stochastic Depth. Adding stochastic depth to ResNet110 provided a speed-up for both thresholds. Stochastic depth was much faster at reaching the lower threshold (1.21x speed-up) than the higher threshold (1.02x speed-up). While the total training time speed-up was close to the 1.25x speed-up reported in [26], the number of epochs to reach the higher threshold was larger for stochastic depth than the baseline, which led to the decreased speed-up in terms of time to accuracy.

Multi-GPU Training and Stochastic Depth Combined. By combining stochastic depth and multi-GPU training, we see that the maximum accuracy reached by applying all the optimizations is about a percent lower than the maximum accuracy reached by the baseline (Figure 4). We note that there are many possible ways to tune learning rates, batch size and optimization algorithms that may achieve better performance here. For this experiment, we chose to follow the standard recommendations in the literature [24, 18, 26] to illustrate what a practitioner following these approaches may achieve, and we performed parameter sweeps where possible to explore alternative settings. Without standard ways to run and tune different optimizations jointly, composing optimizations may affect the convergence rate of the algorithm, making performance hard to reason about.

3.3 Comparing Hardware and Software Frameworks

We seeded DAWNBench with single-GPU and CPU results for TensorFlow and PyTorch, using reference implementations of models when possible. We show some of the variability present across DAWNBench's metrics even from simple factors such as the model, software framework, and hardware type, in Figure 5. This figure presents training time to 93% validation accuracy, and single-image inference la-



Accuracy threshold	Time to threshold	Fastest model
91.8%	1h 43m	ResNet20
93%	4h 42m	ResNet56
94%	10h 6m	ResNet164 (simple)
94.4%	11h 42m	ResNet164 (bottleneck)

Figure 6: Validation accuracy vs. training time for different ResNet architectures on CIFAR10. Horizontal lines indicate accuracy thresholds of 91.8%, 93%, 94%, and 94.4%. ResNet20, ResNet56, ResNet164 (with simple building blocks), and ResNet164 (with bottleneck building blocks) are fastest to the corresponding accuracy thresholds.

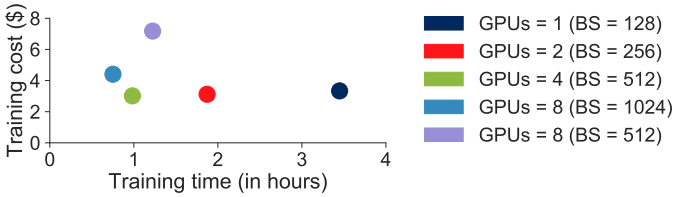


Figure 7: Training cost vs. training time for ResNet56 on the CIFAR10 dataset, using different numbers of GPUs, with an accuracy threshold of 92.5%. The cost of training stays roughly the same, regardless of the number of GPUs used, until 8 GPUs. Training time scales almost linearly with the inverse of the number of GPUs.

tency for various ResNet architectures for the CIFAR10 dataset, on different hardware platforms (1 K80 GPU on two cloud providers [Google Compute Engine and Amazon EC2], 1 P100 GPU on a private cluster, and a 16vCPU machine on Google Cloud with Broadwell microarchitecture) and frameworks.

As the figure illustrates, TensorFlow is faster than PyTorch on CPUs, but slightly slower on GPUs, both for training and inference. This is partly due to data format: TensorFlow supports both NCHW and NHWC layouts (N: Number of Samples, C: Number of Channels, H: Height, W: Width), which give better performance on GPUs and CPUs respectively, while PyTorch only supports NCHW. K80 performance is similar on both cloud providers, but with spot pricing for GPU instances, Amazon is cheaper. Training and inference time are proportional to the depth of the model, as expected.

We also see that for different target validation accuracies, different ResNet architectures are "optimal" (that is, fastest to reach target validation accuracy). Figure 6 shows how the top-1 validation accuracy varies with training time for different ResNets for CIFAR10 on a Nvidia K80 GPU. For lower accuracy thresholds, shallower architectures reach the threshold faster.

We also demonstrate how training cost relates to training time for ResNet56 on CIFAR10 as the number of GPUs used changes (Figure 7). Prior to running this experiment, we expected training cost to increase as the number of GPUs increases (and training time decreases), leading to a Pareto frontier between cost and time. In practice, we did not observe this to be the case for CIFAR10 experiments run on 1 to 4 GPUs – training time scales perfectly linearly with the inverse of the number of GPUs used, and hence cost remains constant despite training time going down. However, when scaling this computation to 8 GPUs, we did see an increase in the cost of training, because training time does not decrease enough to counter the doubling in instance cost per unit time.

4 Conclusion

DAWNBench is a benchmark that measures the *end-to-end* time to train a model with *state-of-the-art* accuracy, and inference time with that accuracy. By focusing on time to accuracy, we show how different optimizations can interact; using different optimizations in conjunction can prevent models from converging, or increase the time to do so. Reasoning about deep learning systems in this way exposes valuable trade-offs between training time, training cost, and inference time. We intend to keep DAWNBench up to date with new tasks and goals to foster a quantitatively-informed dialogue around progress in deep learning systems.

Acknowledgments

We thank Toby Boyd, Doug Burger, Soumith Chintala, Ramesh Illikal, Mario Srouji, Jian Zhang, Bill Zhao, Xiaoqiang Zheng, the anonymous reviewers, and the many members of the Stanford InfoLab for their valuable feedback on this work. 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 – Intel, Microsoft, Teradata, and VMware – as well as industrial gifts and support from Toyota Research Institute, Juniper Networks, Keysight Technologies, Hitachi, Facebook, Northrop Grumman, NetApp, and the NSF under grants DGE-1656518, DGE-114747, and CNS-1651570.

References

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

[2] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2016. Fathom: Reference Workloads for Modern Deep Learning Methods. In *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 1–10.

[3] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. 2015. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435* (2015).

[4] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. 2015. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435* (2015).

[5] Baidu. 2017. DeepBench: Benchmarking Deep Learning operations on different hardware. (Aug. 2017). `https://github.com/baidu-research/DeepBench`

[6] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, 437–478.

[7] Microsoft Research Blog. 2017. Microsoft unveils Project Brainwave for real-time AI. `https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/`. (2017). Accessed: 2017-09-04.

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

[9] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, USA, 571–582. `http://dl.acm.org/citation.cfm?id=2685048.2685094`

[10] Soumith Chintala. 2017. Convnet-benchmarks: Easy Benchmarking of All Publicly Accessible Implementations of Convnets. (Sept. 2017). `https://github.com/soumith/convnet-benchmarks` original-date: 2014-07-12T03:18:46Z.

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

[12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large Scale Distributed Deep Networks. In *Advances in neural information processing systems*. 1223–1231.

[13] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 1243–1252. `http://proceedings.mlr.press/v70/gehring17a.html`

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 315–323.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[16] Google. 2017. TensorFlow Benchmarks. (2017). https://www.tensorflow.org/performance/benchmarks

[17] Google. 2017. TensorFlow Performance Guidelines. (2017). https://www.tensorflow.org/performance/performance_models

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

[19] Graphcore. 2017. Accelerating Next Generation Machine Intelligence. https://www.graphcore.ai/technology. (2017). Accessed: 2017-09-04.

[20] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *SIGARCH Comput. Archit. News* 44, 3 (June 2016), 243–254. https://doi.org/10.1145/3007787.3001163

[21] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[22] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. 2016. Addressing the straggler problem for iterative convergent parallel ML. In *SoCC*. 98–111.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[25] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[26] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. 2016. Deep networks with stochastic depth. In *European Conference on Computer Vision*. Springer, 646–661.

[27] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. (2016). https://arxiv.org/abs/1602.07360

[28] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. 448–456.

[29] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM '14)*. ACM, New York, NY, USA, 675–678. https://doi.org/10.1145/2647868.2654889

[30] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 1–12. `https://doi.org/10.1145/3079856.3080246`

[31] Andrej Karpathy. 2011. Lessons learned from manually classifying CIFAR-10. `http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/`. (2011).

[32] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On large-batch training for deep learning: Generalization gap and sharp minima. *ICLR* (2017).

[33] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[34] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[35] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. 2014. Efficient mini-batch training for stochastic optimization. In *SIGKDD*. ACM, 661–670.

[36] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. 2016. Asynchrony begets Momentum, with an Application to Deep Learning. In *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*. IEEE, 997–1004.

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

[38] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv preprint arXiv:1606.05250* (2016).

[39] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*. 693–701.

[40] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. 115, 3 (2015), 211–252. `https://doi.org/10.1007/s11263-015-0816-y`

[41] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking state-of-the-art deep learning software tools. In *Proceedings of the International Conference on Cloud Computing and Big Data*. IEEE.

[42] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. 2014. Fast large-scale optimization by unifying stochastic gradient and quasi-Newton methods. In *International Conference on Machine Learning*. 604–612.

[43] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. 2017. meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*,

Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 3299–3308. http://proceedings.mlr.press/v70/sun17c.html

[44] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*. 1139–1147.

[45] T Tieleman and G Hinton. 2014. RMSprop Gradient Optimization. *http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf* (2014).

[46] Leonard Truong, Rajkishore Barik, Ehsan Totoni, Hai Liu, Chick Markley, Armando Fox, and Tatiana Shpeisman. 2016. Latte: A Language, Compiler, and Runtime for Elegant and Efficient Deep Neural Networks. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '16)*. ACM, New York, NY, USA, 209–223. https://doi.org/10.1145/2908080.2908105

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *arXiv preprint arXiv:1706.03762* (2017).

[48] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer. 2017. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2017-07). 446–454. https://doi.org/10.1109/CVPRW.2017.60

[49] Ce Zhang and Christopher Ré. 2014. Dimmwitted: A study of main-memory statistical analytics. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1283–1294.

[50] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. (2017). arXiv:1707.01083 http://arxiv.org/abs/1707.01083