ImageNet Training in Minutes

Yang You
UC Berkeley
youyang@cs.berkeley.edu

Zhao Zhang
TACC
zzhang@tacc.utexas.edu

Cho-Jui Hsieh
UC Davis
chohsieh@ucdavis.edu

James Demmel
UC Berkeley
demmel@cs.berkeley.edu

Kurt Keutzer
UC Berkeley
keutzer@cs.berkeley.edu

ABSTRACT

In this paper, we investigate large scale computers' capability of speeding up deep neural networks (DNN) training. Our approach is to use large batch size, powered by the Layer-wise Adaptive Rate Scaling (LARS) algorithm, for efficient usage of massive computing resources. Our approach is generic, as we empirically evaluate the effectiveness on two neural networks: AlexNet and ResNet-50 trained with the ImageNet-1k dataset while preserving the state-of-the-art test accuracy. Compared to the baseline of a previous study from a group of researchers at Facebook, our approach shows higher test accuracy on batch sizes that are larger than 16K. Using 2,048 Intel Xeon Platinum 8160 processors, we reduce the 100-epoch **AlexNet training time from hours to 11 minutes**. With 2,048 Intel Xeon Phi 7250 Processors, we reduce the 90-epoch **ResNet-50 training time from hours to 20 minutes**. Our implementation is open source and has been released in the Intel distribution of Caffe v1.0.7.

CCS CONCEPTS

• **Computing methodologies → Massively parallel algorithms**;

KEYWORDS

Distributed Machine Learning, Fast Deep Neural Networks Training

ACM Reference Format:
Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 2018. ImageNet Training in Minutes. In *ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3225058.3225069

1 INTRODUCTION

For deep learning applications, larger datasets and bigger models lead to significant improvements in accuracy [2], but at the cost of longer training time. Moreover, many applications, such as computational finance [14], autonomous driving [3], oil and gas exploration [15], and medical imaging [11], will almost certainly require training data-sets with billions of training elements and terabytes of data. Obtaining a working model for the above applications is often a repeated process of the training execution with varying hyper parameter settings. For example, it takes a Nvidia M40 GPU 14 days to finish just one 90-epoch ResNet-50 training execution on the ImageNet-1k dataset. This long experiment turnaround motivates the research of training time reduction of Deep Neural Networks (DNN). The 90-epoch ResNet-50 training requires 10^{18} single precision operations in total. On the other hand, the world's current fastest supercomputer can finish 2×10^{17} single precision operations per second [9]. So, if the 90-epoch ResNet-50 training can make full use of the computing capability of the fastest supercomputer, it should be able to finish in five seconds.

So far, the best results on scaling ImageNet-1k training have used the synchronous stochastic gradient descent method (synchronous SGD) to enable parallelism. The synchronous SGD algorithm has many inherent advantages, but at the root of these advantages is determinism (modulo floating point round-off). Determinism ensures that all valid parallel implementations of the algorithm match the behavior of the sequential version. This property is invaluable during DNN design and during the debugging of optimization algorithms. In parallel DNN training with synchronous SGD, larger batch size is important to keep up machine efficiency, as it assigns each processor sufficient amount of work in each iteration. For example, engaging 512 processors with a batch size of 1k would mean that each processor only gets a local batch of 2 images. In contrast, a larger batch size of 32k assigns each processor 64 images in each iteration. The latter case thus makes more efficient use of the machines, as the computation to communication ratio is higher.

Over the last two years, we have seen the focus on increasing the batch size and number of processors used in image classification training, with a resulting reduction in training time. FireCaffe [16, 17] demonstrated scaling the training of GoogleNet to 128 Nvidia K20 GPUs with a batch size of 1k and a total training time of 10.5 hours for 72 epochs. Although using a larger batch size naively can lead to significant loss in test accuracy, with the warm-up technique coupled with the linear scaling rule, researchers at Facebook [10] were able to scale the training of ResNet-50 to 256 Nvidia P100 GPUs with a batch size of 8k and a total training time of one hour. Using a more sophisticated approach to adapting the learning rate in the Layer-wise Adaptive Rate Scaling (LARS) algorithm [32], researchers were able to scale the batch size dramatically to 32k. On eight Nvidia P100 GPUs, they reported a 3.4% reduction in accuracy due to the absence of data augmentation.

Given the large batch sizes that the LARS algorithm enables, it is natural to ask: how much further can we scale out the training of DNN on the ImageNet-1k dataset? This is the investigation that

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPP 2018, August 13–16, 2018, Eugene, OR, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6510-9/18/08…$15.00
https://doi.org/10.1145/3225058.3225069

led to this paper. At a high level, we find out that the 32k batch size can efficiently scale DNN training on ImageNet-1k dataset up to thousands of processors. In particular, we are able to finish the **100-epoch training with AlexNet in 11 minutes with 58.6% top-1 test accuracy** (defined in §2.4) on 2,048 Intel Xeon Platinum 8160 processors. With 2,048 Intel Xeon Phi 7250 Processors, we are able to reduce the turnaround time of the **90-epoch ResNet-50 training to 20 minutes without losing accuracy**, inside which the top-1 test accuracy (defined in §2.4) converges to 74.9% at 64th epoch (14 minutes from starting time).

In Summary, we make the following contributions:

- We show the scaling capability of LARS up to thousands of CPUs with no loss of accuracy. Meanwhile we demonstrate that DNN can be successfully trained by CPU-based systems instead of using GPUs. We achieved the best scaling results on Intel hardware.
- We examine the generality of the LARS algorithm on both AlexNet and ResNet-50, while many other works are ResNet-50 specific.
- Empirically, we demonstrate that LARS is more robust than the recent work [10] at a batch size of 32K on large-scale computers.
- Our work has been open sourced and released in the Intel distribution of Caffe v.1.0.7.

2 BACKGROUND AND RELATED WORK

In this section, we discuss the details of data-parallel stochastic gradient descent (SGD) method, the model parallel approach, and similar work of parallelizing DNN training.

2.1 Data-Parallelism SGD

In the data parallelism method, the dataset is partitioned into P parts stored on each machine, and each machine will have a local copy of the neural network and the weights (w^j). In synchronized data parallelism, the communication happens at two places: the sum of local gradients and the broadcast of the global weights. For the first part, each worker computes the local gradient ∇w^j independently, and sends the update to the master node. The master then updates $\tilde{w} \leftarrow \tilde{w} - \eta/P \sum_{j=1}^{P} \nabla w^j$ after it gets all the gradients from workers. Here, η is the algorithm's learning rate. For the second part, the master broadcasts $\tilde{w}$ to all workers.

There is another implementation for the communication part. The **1 reduce + 1 broadcast** pattern can be replaced by **1 all-reduce** operation. In this situation, each worker computes the local gradient ∇w^j independently, and then the system conducts an all-reduce operation to send the sum of the gradients ($\sum_{j=1}^{P} \nabla w^j$) to all the machines. After that, each machine will do the weight updating ($w^j \leftarrow w^j - \eta/P \sum_{j=1}^{P} \nabla w^j$) locally. In this paper, we use the all-reduce method rather than the reduce-broadcast method. This synchronized approach is a widely-used method on large-scale systems [17].

Scaling synchronous SGD to more processors has two challenges. The first is giving each processor enough useful work to do; this has already been discussed in §1. The second challenge is the inherent problem that after processing each local batch all processors must synchronize their gradient updates via a barrier before proceeding. This problem can be partially ameliorated by overlapping computation and communication [6, 10], but the inherent synchronization barrier remains. A more radical approach to breaking this synchronization barrier is to pursue a purely asynchronous method. A variety of asynchronous approaches have been proposed [19, 24, 27, 34]. The communication and updating rules differ in the asynchronous approach and the synchronous approach. The simplest version of the asynchronous approach is a master-worker scheme. At each step, the master only communicates with one worker. The master gets the gradients ∇w^j from the j-th worker, updates the global weights, and sends the global weight back to the j-th worker. The order of workers is based on first-come-first-serve strategy. The master machine is also called as *parameter server*. The idea of a parameter server was used in real-world commercial applications by the Downpour SGD approach [7], which has successfully scaled to 16, 000 cores. However, Downpour's performance on 1, 600 cores for a globally connected network is not significantly better than a single GPU [29].

2.2 Model Parallelism

Data parallelism replicates the neural network itself on each machine while model parallelism partitions the neural network into P pieces. Partitioning the neural network means parallelizing the matrix operations on the partitioned network. Thus, model parallelism approach can get the same solution as trained by a single machine. Model parallelism has been studied in [4, 22]. However, in many cases, such as image classification, the input size (e.g. size of an image) is relatively small, and the matrix operations are not large. For example, parallelizing a 2048×1024×1024 matrix multiplication only needs one or two machines. Thus, state-of-the-art methods often use the data-parallelism approach [2, 5, 7, 28].

2.3 Other Work

A group of researchers at Facebook reported finishing the 90-epoch ResNet-50 training on ImageNet-1k dataset with 256 Nvidia P100 GPUs within one hour [10]. Their work uses a batch size of 8k. Though we are able to achieve faster training speed than the reported case, our baseline's accuracy is slightly lower than Facebook's version (76.2% vs 75.3%) due to our usage of weaker data augmentation. However, our approach has a higher accuracy with batch sizes that are larger than 16k, as shown in Figure 1.

Codreanu *et al.* [1] reported achieving 73.78% accuracy on ResNet-50 (with data augmentation) in less than 40 minutes on 512 Intel Xeon Phi 7250 Processors. There are two things worth noting: Firstly, their batch size is 8k. Secondly, that case only ran for 37 epochs. The complete 90-epoch training would take 80 minutes with 75.25% accuracy.

Akiba *et al.* [1] reported finishing the 90-epoch ResNet-50 training within 15 minutes on 1,024 Nvidia P100 GPUs. However, the baseline accuracy is missing in the report, so it is difficult to tell how much their 74.9% accuracy using the 32k batch size diverges from the baseline. Secondly, both Akiba et al. and Facebook's work [10]

[1] https://blog.surf.nl/en/imagenet-1k-training-on-intel-xeon-phi-in-less-than-40-minutes/

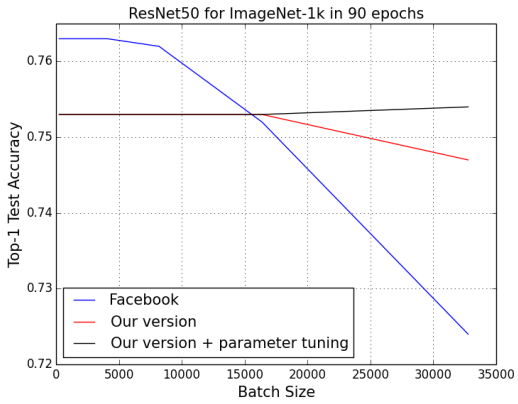


Figure 1: Top-1 Test Accuracy Comparison on Various Batch Sizes between Our Approach and Facebook's solution

are ResNet-50 specific, while we also show the generality of our approach with AlexNet. It is worth noting that our online preprint [33] is two months earlier than Akiba *et al.*

2.4 Top-1 accuracy and Top-5 accuracy

In this section, we explain the difference between the top-1 accuracy and the top-5 accuracy. Top-1 accuracy means the conventional accuracy: the model's prediction (the one with highest probability) must be exactly the expected answer. Top-5 accuracy means that any of your model's five highest probability predictions match the expected answer. For example, let us apply machine learning to object recognition using a neural network. A picture of an airplane is shown, and these are the outputs of our neural network:

- Car with 68% probability
- Train with 11% probability
- Bus with 10% probability
- Airplane with 9% probability
- Tank with 8% probability
- Gun with 2% probability
- Building with 1% probability

If we use top-1 accuracy, we count this output as false, because it predicted a car. If we use top-5 accuracy, we count this output as true, because airplane is among the top-5 guesses. In this example, the dataset has seven classes. The ImageNet-1k dataset has 1,000 classes. All the accuracy in this paper means top-1 test accuracy. Here, **test accuracy** is the prediction accuracy of the model on the validation dataset, which the model is not trained on.

3 LARGE BATCH DNN TRAINING

In this section, we discuss the benefits and challenges of large batch training, and the rational of our model selection along with a learning rate profile study. Throughout the discussion, we focus on the data-parallel synchronous SGD approach, as it is proven to be stable for DNN training at scale [10]. In contrast, the asynchronous methods using parameter server are not guaranteed to be stable in a distributed environment [5].

3.1 Benefits of Large Batch Training

The **prominent advantage of large batch training is that it can reduce the overall training time**. The idea is straightforward—by using a **large batch size** for SGD, the work for each iteration can be distributed to multiple processors. Consider the following ideal case. ResNet-50 requires 7.72 billion single-precision operations to process one 225x225 image. If we run 90 epochs on the ImageNet-1k dataset of 1.28×10^6 images, the total number of operations is $90 * 1.28 \times 10^6 * 7.72 \times 10^9 \approx 10^{18}$. Currently, the most powerful supercomputer can finish 200×10^{15} single-precision operations per second [9]. If there is an algorithm allowing us to make full use of the fastest supercomputer, we can finish the ResNet-50 training in 5 seconds.

To do so, we need to make the algorithm use more processors and load more data at each iteration, which corresponds to increasing the batch size in synchronous SGD. Ideally, if we fix total number of data accesses and grow the batch size linearly with number of processors, the number of SGD iterations will decrease linearly and the time cost of each iteration remains constant, so the total time will also reduce linearly with number of processors. A detailed analytical study on the ResNet-50 training is shown in Table 1.

In the strong scaling situation, large batch does not change the number of floating point operations (computation volume), as the number of epochs is fixed. However, large batch can reduce the overall communication volume. The reason is that larger batch size results in less iterations thus less overall communication volume, as the single iteration communication volume remains relatively constant given the fact that it is only related to the model size and the networking system. In this way, large batch size reduces the overall DNN training time in a scalable manner.

A **second benefit of large batch training is that it can keep up the high machine utilization**, which is especially important in a distributed environment.

Let us use one Nvidia M40 GPU to illustrate this benefit on a single machine. Figure 2 shows the M40 GPU performance measurements (in images/sec) for AlexNet with varying batch size from 16 to 512. Increasing the batch size from 16 to 32, the performance almost doubles. And from 128 to 512, the curve flattens, which means the M40 GPU approaches its peak performance at the batch size of 512. A large batch size, such as 8,192, can keep a 16 M40 GPU cluster at its peak performance during the training execution.

3.2 Model Selection

To scale out DNN training to many machines, a major overhead is the communication among different machines [35]. Here we define the notion of **scaling ratio** as ratio between computation and communication. For DNN models, the computation is proportional to the number of floating point operations required for processing an image. Since we focus on synchronous SGD approach, the communication is proportional to model size (or the number of parameters). Different DNN models have different scaling ratios. To generalize our study, we pick two representative models: AlexNet and ResNet-50. The reason is that they have different scaling ratios. From Table 2, we see that ResNet-50's scaling ratio is 12.5× larger than that of AlexNet. This means scaling ResNet-50 is easier

Table 1: An Analytical Scaling Performance Study with ResNet-50 as the Example. t_1 is the computation time and t_2 is communication time. We fix the number of epochs as 100. Larger batch size needs less iterations. We set batch size as 512 per machine. Then we increase the number of machines. Since $t_1 \gg t_2$ for using ImageNet-1k dataset to train ResNet-50 on GPUs [10], the single iteration time is dominant by the computation. Thus the total time will be reduced approximately linearly.

Batch	Epochs	Iters	GPUs	IterationTime	TotalTime
512	100	250k	1	t_1	250kt_1
1024	100	125k	2	$t_1 + \log(2)t_2$	125k$(t_1 + \log(2)t_2)$
2048	100	62500	4	$t_1 + \log(4)t_2$	62500$(t_1 + \log(4)t_2)$
4096	100	31250	8	$t_1 + \log(8)t_2$	31250$(t_1 + \log(8)t_2)$
8192	100	15625	16	$t_1 + \log(16)t_2$	15625$(t_1 + \log(16)t_2)$
...	...	...	...		
1.28M	100	100	2500	$t_1 + \log(2500)t_2$	100$(t_1 + \log(2500)t_2)$

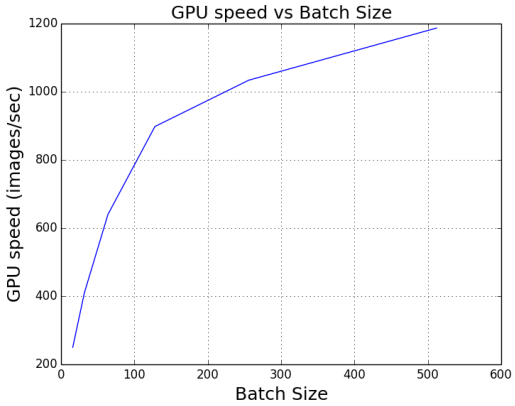


Figure 2: AlexNet Training Performance on Various Batch Sizes on a Nvidia M40 GPU. Peak performance is reached with the batch size of 512, while a 1,024 batch size runs out of memory.

than scaling AlexNet. Our experiments in Figures 10, 11, 12, and 13 confirmed this conclusion.

Table 2: Scaling Ratio for AlexNet and ResNet50.

Model	comm: parameters	comp: flops per image	comp/comm scaling ratio
AlexNet	61 million	1.5 billion	24.6
ResNet50	25 million	7.7 billion	308

3.3 Challenges of Large Batch Training

Large batch size comes with the benefits of shorter training time and high machine utilization. However, naively using synchronous SGD with large batch size usually achieves test accuracy degradation compared to smaller batch sizes with a fixed number of epochs. Unfortunately, there is no algorithm allowing us to effectively use unlimitedly large batch sizes [20]. Table 3 shows the target test accuracy by standard benchmarks. For example, when we set the batch size of AlexNet larger than 1,024 or the batch size of ResNet-50 larger than 8,192, the test accuracy will be significantly degraded, as shown in Table 4 and Figure 3, respectively.

Table 3: Standard Benchmarks for ImageNet-1k training.

Model	Epochs	Test Top-1 Accuracy
AlexNet	100	58% [17]
ResNet-50	90	75.3% [13]

Table 4: AlexNet Test Accuracy with Varying Batch Size. Current approaches (linear scaling + warmup) do not work for AlexNet with a batch size larger than 1k. We tune the warmup epochs from 0 to 10 and pick the one with highest accuracy. According to linear scaling, the optimal learning rate (LR) of batch size 4k should be 0.16. We use the poly learning rate policy, and the poly power is 2. The momentum is 0.9 and the weight decay is 0.0005.

Batch Size	Base LR	warmup	epochs	test accuracy
512	0.02	N/A	100	0.583
1k	0.02	no	100	0.582
4k	0.01	yes	100	0.509
4k	0.02	yes	100	0.527
4k	0.03	yes	100	0.520
4k	0.04	yes	100	0.530
4k	0.05	yes	100	0.531
4k	0.06	yes	100	0.516
4k	0.07	yes	100	**0.001**
...	...	...	...	...
4k	0.16	yes	100	**0.001**

For large batch training, it is essential to keep up the test accuracy with smaller batches under the constraint of the same number of epochs. Here we fix the number of epochs because: Statistically, one epoch means the algorithm touches the entire dataset once; and computationally, fixing the number of epochs means fixing the number of floating point operations. State-of-the-art techniques for large batch training to remedy the test accuracy degradation issue include:

(1) **Linear Scaling** [21]: With an increase of the batch size from B to kB, we should also increase the learning rate from η to $k\eta$.

(2) **Warmup Scheme** [10]: With a large learning rate (η), should start from a small η and increase it to the large η in the first few epochs.

The intuition of linear scaling is related to the number of iterations. Let us use B, η, and I to denote the batch size, the learning rate, and the number of iterations. If we increase the the batch

size from B to kB, then the number of iterations is reduced from I to I/k. This indicates that the frequency of weight updating is reduced by k times. Thus, we make the updating of each iteration $k\times$ more efficient by enlarging the learning rate by k times. The purpose of a warmup scheme is to avoid the situation in which the algorithm diverges at the beginning because we have to use a very large learning rate based on linear scaling. With these techniques, researchers can use the relatively large batch in a certain range (Table 5). However, we observe that these state-of-the-art approaches can only scale batch size to 1k for AlexNet and 8k for ResNet-50. With the batch size of 4k for AlexNet, we can only achieve a 53.1% test accuracy in 100 epochs (Table 4). Our target is to preserve the 58% test accuracy even when using large batch sizes, such as 32k.

Table 5: State-of-the-art Large Batch Training and Test Accuracy. Batch1 means baseline batch size. Batch2 means large batch size. Accuracy1 means baseline accuracy. Accuracy2 means large-batch accuracy.

Team	Model	Batch1	Batch2	Accuracy1	Accuracy2
Google [21]	AlexNet	128	1024	57.7%	56.7%
Amazon [23]	ResNet-152	256	5120	77.8%	77.8%
Facebook [10]	ResNet-50	256	8192	76.40%	76.26%

3.4 Scaling up Batch Size

To improve the accuracy for large batch training, a new rule of learning rate (LR) schedule was developed. As discussed in §2.1, we use $w = w - \eta\nabla w$ to update the weights. Each layer has its own weight w and gradient ∇w. Standard SGD algorithm uses the same LR (η) for all the layers. However, from our experiments, we observe that different layers may need different LRs. The reason is that the ratio between $||w||_2$ and $||\nabla w||_2$ varies significantly for different layers. From example, we observe that $||w||_2/||\nabla w||_2$ is only 20 for conv1.1 layer (Table 6). However, $||w||_2/||\nabla w||_2$ is 3,690 for fc6.1 layer. To speedup the convergence for fc6.1 layer, the users need to use a large LR. However, this large LR may lead to divergence on the conv1.1 layer. We believe this is an important reason of the optimization difficulty in large batch training.

Goyal et al [10] proposed the warmup scheme to solve this problem. The warmup scheme works well for ResNet-50 training with a batch size $\leq$ 8k. However, only using this recipe does not work for AlexNet with batch size > 10k and ResNet-50 with batch size > 8k.

Together with researchers at Nvdia, we proposed Layer-wise Adaptive Rate Scaling (LARS) algorithm [32] to improve large batch training's test accuracy. The base LR rule is defined in Equation (1). l is the scaling factor, which we set as 0.001 for AlexNet and ResNet training. γ is a tuning parameter for users. Usually γ can be chosen by linear scaling.

$$\eta = l \times \gamma \times \frac{||w||_2}{||\nabla w||_2} \quad (1)$$

In this formulation, different layers can have different LRs. In practice, we add momentum (denoted as μ) and weight decay (denoted as β) to SGD, and use the following sequence for LARS:

(1) get the local LR for each learnable parameter by $\alpha = l \times ||w||_2/(||\nabla w||_2 + \beta||\nabla w||_2)$;

Table 6: The ratios between $||w||_2$ and $||\nabla w||_2$ for different layers of AlexNet with batch size = 4k after at 1st epoch. We observe that they are very different from each other. fc is the fully connected layer and conv is the convolutional layer. x.0 is a layer's weight. x.1 is a layer's bias.

Layers	$\|\|w\|\|_2$	$\|\|\nabla w\|\|_2$	$\|\|w\|\|_2/\|\|\nabla w\|\|_2$
fc8.0	20.24	0.078445	258
fc8.1	0.316	0.006147	51
fc7.0	20.48	0.110949	184
fc7.1	6.400	0.004939	1296
fc6.0	30.72	0.097996	314
fc6.1	6.400	0.001734	**3690**
conv5.0	6.644	0.034447	193
conv5.1	0.160	0.000961	166
conv4.0	8.149	0.039939	204
conv4.1	0.196	0.000486	403
conv3.0	9.404	0.049182	191
conv3.1	0.196	0.000511	384
conv2.0	5.545	0.057997	96
conv2.1	0.160	0.000649	247
conv1.0	1.866	0.071503	26
conv1.1	0.098	0.004909	**20**

(2) get the LR for each layer by $\eta = \gamma \times \alpha$;
(3) update the gradients by $\nabla w = \nabla w + \beta w$;
(4) update acceleration term a by $a = \mu a + \eta\nabla w$;
(4) update the weights by $w = w - a$.

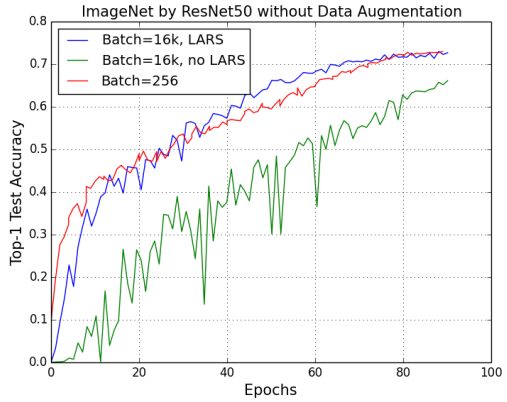
Using this approach together with the warmup technique, SGD with large batch can achieve identical test accuracy with the batch size of 32k as the baseline for AlexNet (Table 7). Technically, we change the local response normalization (LRN) to batch normalization (BN). We add BN after each convolutional layer. As shown in Figure 3, we can see that the LARS algorithm can keep up the test accuracy for ResNet-50 using 32k batch with the baseline of ~73% without data augmentation. In comparison, the current approaches of combining linear scaling and warmup has lower accuracy on ResNet-50 for batch size of 16k and 32k (68% and 56%, respectively).

Table 7: Test Accuracy of AlexNet with Batch Size of 32k using KNL Nodes on Stampede2. We use ploy learning rate policy, and the poly power is 2. The momentum is 0.9 and the weight decay is 0.0005. For a batch size of 32K, we changed local response norm in AlexNet to batch norm. Specifically, we use the refined AlexNet model by B. Ginsburg[1].

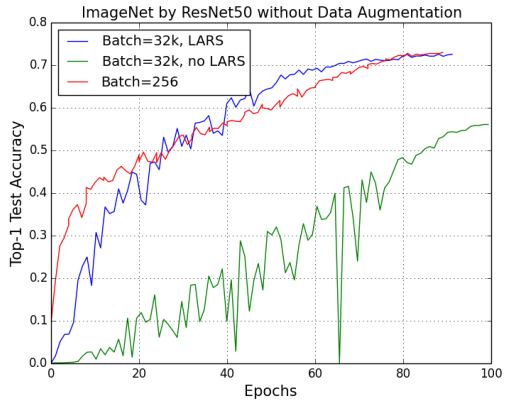
Batch Size	LR rule	warmup	Epochs	test accuracy
512	regular	N/A	100	0.583
4096	LARS	13 epochs	100	0.584
8192	LARS	8 epochs	100	0.583
32768	LARS	5 epochs	100	0.585

4 PERFORMANCE EVALUATION

In this section, we evaluate the 100-epoch AlexNet and 90-epoch ResNet-50 training on a number of platforms with different hardware. We will briefly introduce the hardware and software settings,



(a) Batch Size=16k



(b) Batch Size=32k

Figure 3: Test Accuracy Comparison between Large Batch Training, Large Batch Training with LARS, and the Baseline. The base learning rate of Batch 256 is 0.2 with poly policy (power=2). For the version without LARS, we use the state-of-the-art approach [10]: 5-epoch warmup and linear scaling for LR. For the version with LARS, we also use 5-epoch warmup. Clearly, the existing method does not work for Batch Size larger than 8K. LARS algorithm can help the large batch to achieve the same accuracy with baseline in the same number of epochs.

and present the comparison baseline and code change. In addition to performance results, we will also discuss the communication overhead analysis in details.

4.1 Hardware

Throughout this section, we run experiments on three types of hardware. The Intel Xeon Phi 7250 Processors and the Intel Xeon Platinum 8160 processors are part of the Stampede2 supercomputer hosted at Texas Advanced Computing Center[2]. The eight Nvidia P100 GPU cluster is locally hosted.

[2]portal.tacc.utexas.edu/user-guides/stampede2

We perform the large batch scaling efficiency study on the eight Nvidia P100 GPU cluster. Each P100 GPU has the performance of 10.6 teraflops and has 16 GB memory.

The Intel Xeon Phi 7250 Processor (referred as KNL) is the latest version of Intel's general-purpose accelerator. It is a self-hosted platform running CentOS 7 on our testbed. Each processor has 68 physical cores, and four hardware threads per core. All cores are running at 1.4 GHz clock rate. On Stampede2, 3,696 out of the total 4,200 KNL nodes are configured in the following way: On each node, there is one processor with 96 GB DDR4 RAM, 16 GB MCDRAM, and a 200 GB local Solid State Drive, of which 144 GB is available. The memory is configured as cache-quadrant mode, where MCDRAM is used as an L3 cache.

The Intel Xeon Platinum 8160 processors (referred as SKX) are part of Intel Xeon Scalable Processors collection. Each SKX node in Stampede2 has two such processors with 48 physical cores in total. Each cores is documented with a 2.1 GHz clock rate, however, the clock rate varies from 1.4 GHz to 3.7 GHz depending on the instruction set and the number of active cores. Our measured runtime clock rate for AlexNet and ResNet-50 are 2.1 GHz and 2.0 GHz, respectively. Each SKX node is equipped with 192 GB RAM and a 200 GB Solid State Drive, of which 144 GB is available. There are 1,600 SKX nodes (3,200 processors) on Stampede2.

4.2 Software

On Nvidia GPUs, we used the Nvidia distribution of Caffe [3]. And on Intel processors, we used two variants of the official Caffe [18]: 1) our customized parallel implementation that uses MPI [12] for the communication across nodes and 2) the Intel distribution of Caffe v1.0.3 [4], which supports multi-node training by Intel Machine Learning Scaling Library (MLSL) v2017.1.016 [5].

4.3 Data and Baseline

Throughout the performance evaluation, we use the ImageNet-1k [8] dataset. The dataset has 1.28 million images for training and 50,000 images for testing. There are two top-1 test accuracy baseline for ResNet-50 in 90 epochs: the case without data augmentation is 73% while the case with data augmentation is 75.3%. The top-1 test accuracy baseline for AlexNet in 100 epochs is about 58%.

4.4 Scaling Efficiency of Large Batches

As discussed in §3.1, using large batch size can reduce the communication volume with less iterations, thus yielding higher scaling efficiency than small batch size. Here, we present an analytical study and the empirical performance evaluation to validate this hypothesis.

Communication often is the major bottleneck for efficient scaling for applications across many processors (Table 8). On a distributed system, communication means moving the data over the network (e.g. master machine broadcast its data to all the worker machines). In DNN training, communication across nodes is in the form of a all-reduce (sum of local gradients). These communication patterns have a higher scaling overhead than the matrix

[3]https://github.com/NVIDIA/caffe
[4]https://github.com/intel/caffe
[5]https://github.com/intel/MLSL

computations (i.e. the matrix computations on each machine can be finished independently). In particular, the all-reduce on N nodes have the scaling factor of $O(\log N)$ or $O(N)$ depending on the network topology [25, 26, 30, 31]. And the scaling factor of broadcast is $O(\log N)$. In contrast, the matrix computation in DNN training can be distributed almost evenly to N nodes with the scaling factor of $O(1/N)$.

For finishing the same number of epochs, the communication overhead is lower in the large batch version than in the small batch version, as the large batch version sends fewer messages (latency overhead) and moves less data (bandwidth overhead). For synchronous SGD, the algorithm needs to conduct an all-reduce operation (sum of gradients on all machines) in each iteration. The number of messages sent is linear with the number of iterations. And for each iteration, the communication volume is constant regardless of the batch size, as the gradients has identical size as the model weights ($|W|$).

Let us use the following notations for the analytical evaluation:

- E the number of epochs
- n the total number of images in the training dataset
- B the batch size

Then the number of iterations is $E \times n/B$. Holding E and n constant, with the large batch size, the program finishes with less iterations. By fixing E, the number of epochs, it is fixing the total number of floating point operations. Meanwhile, the number of iterations is consistent with the communication frequency of the training process. Let us denote $|W|$ as the neural network model size. Then we can get the communication volume is $|W| \times E \times n/B$.

Thus, the large batch version transfers less data than the small batch version to finish the same number of floating point operations. In summary, the number of floating point operations remain constant when the number of epochs is fixed. **The larger batch size increases the computation-communication ratio because it reduces the communication frequency**. As a result, the larger batch size makes the algorithm more scalable on distributed systems.

Table 8: Communication unit is much slower than computation unit because time-per-flop (γ) $\ll$ 1/ bandwidth (β) $\ll$ latency (α). For example, $\gamma = 0.9 \times 10^{-13}$s for NVIDIA P100 GPUs.

Network	α (latency)	β (1/bandwidth)
Mellanox 56Gb/s FDR IB	0.7×10^{-6}s	0.2×10^{-9}s
Intel 40Gb/s QDR IB	1.2×10^{-6}s	0.3×10^{-9}s
Intel 10GbE NetEffect NE020	7.2×10^{-6}s	0.9×10^{-9}s

For the empirical performance evaluation, we use the ImageNet-1k training with AlexNet-BN on eight P100 GPUs in this experiment. The baseline's batch size is 512 and is referred as the small batch size. The large batch size is 4k. In this example, we focus on the the communication across GPUs. Firstly, the experiment results confirm that the large batch size achieves the same test accuracy as the small batch size in 100 epochs, as shown in Figure 4. Fixing the number of epochs implies fixing the number of floating point operations. Thus, large batch size achieves the same test accuracy as the small batch size in fixed number of floating point operations (Figure 5).

We observed a 3x reduction in training time with the large batch size compared to that of the small batch size, as shown in Figure 6. The number of iterations in large-batch training is much less than small-batch training (Figure 7). The number of messages is equal to the number of iterations. Our experimental results also confirmed that large batch will reduce the accumulated latency overhead (Figure 8) and bandwidth overhead (Figure 9).

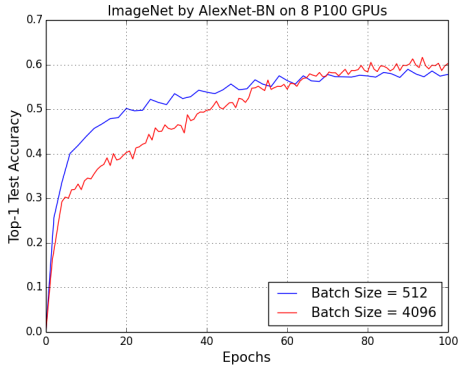


Figure 4: Test Accuracy Comparison between the Small Batch Size and the Large Batch Size. The 512 small batch size is the baseline.

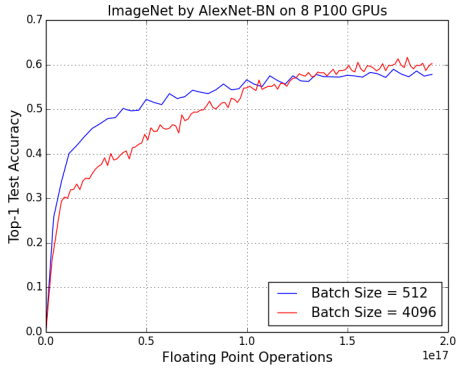


Figure 5: Increasing the batch size does not increase the number of floating point operations. Large batch can achieve the same accuracy in the fixed number of floating point operations.

4.5 ImageNet training with AlexNet

In this experiment, we use the AlexNet training case to evaluate our approach's effectiveness in scaling DNN training at large scale.

Previously, Nvidia reported that using one DGX-1 station they were able to finish 90-epoch ImageNet-1k training with AlexNet in two hours[6]. However, they used half-precision or FP16, whose

[6] www.nextplatform.com/2016/04/06/dgx-1-nvidias-deep-learning-system-newbies

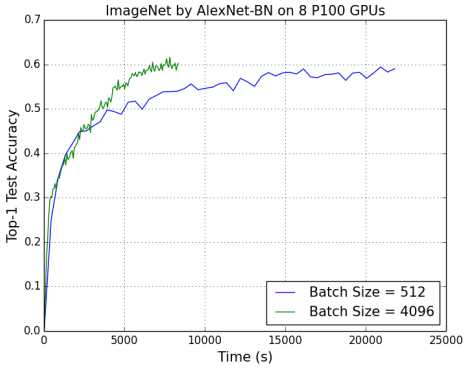


Figure 6: Time-to-solution Comparison between the Small Batch Size and the large Batch Size. To achieve the 58% accuracy, the large batch size of 4k only needs about two hours while the smaller batch size of 512 needs about six hours.

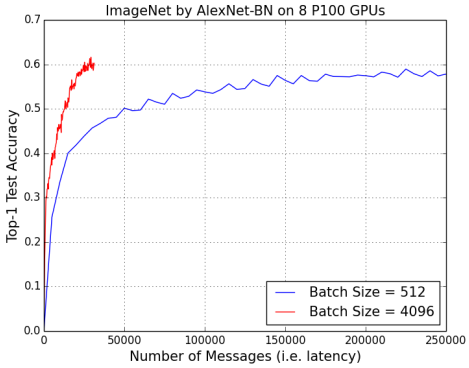


Figure 8: When we fix the number of epochs and increase the batch size, we need much less iterations. The number of iterations is inear with the number of messages the algorithm sent.

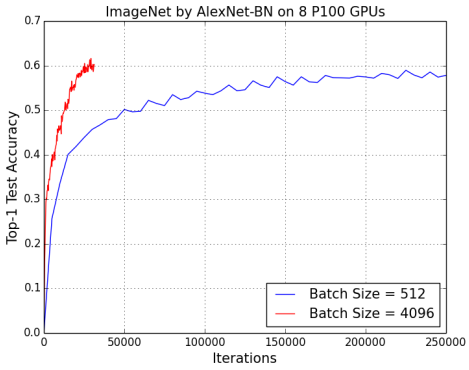


Figure 7: When we fix the number of epochs and increase the batch size, we need much less iterations.

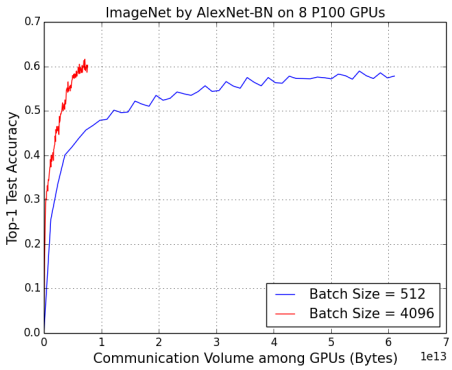


Figure 9: The number of iterations is linear with the number of messages the algorithm sent. Let us denote $|W|$ as the neural network model size. Then we can get the communication volume is $|W| \times E \times n/B$. Thus, the larger batch version needs to move much less data than the smaller batch when they finish the number of floating point operations.

cost is half of the standard single-precision operation. We run the AlexNet training with standard single-precision. It takes 6 hours 9 minutes with the batch size of 512 on one NVIDIA DGX-1 station. With our approach, using the large batch size of 4k achieves similar test accuracy as the small batch size case (Line 2 in Table 7), and it finishes in two hours and ten minutes on the same station. Thus, using large batch size can significantly speedup DNN training on GPU cluster.

Then we scale the same AlexNet training case with a batch size of 32k, and run it on multiple scales of the KNL nodes and the SKX nodes on the Stampede2 supercomputer. Figure 10 and 11 show the strong scaling performance on each type of nodes with the ideal scaling curve relative to the performance of 128 nodes in each case.

Despite the 11-minute training time on 1,024 SKX nodes (2,048 Intel Xeon Platinum 8160 processors), the AlexNet training does not scale well beyond 512 nodes in both cases. The inefficient scaling performance is due to its low scaling ratio, as defined in §3.2. On the other hand, on 512 KNL nodes (512 Intel Xeon Phi 7250 processors), the AlexNet training finished in 24 minutes.

The minute-level training performance is remarkable given the current practice, a comparison against known performance is presented in Table 9.

Table 9: Time-to-solution Comparison against Other Published Performance

Batch Size	epochs	Accuracy	hardware	time
256	100	58.7%	CPU + K20 GPU	144h
512	100	58.8%	DGX-1 station	6h 10m
4096	100	58.4%	DGX-1 station	2h 19m
32768	100	58.5%	512 KNLs	24m
32768	100	58.6%	1024 CPUs	11m

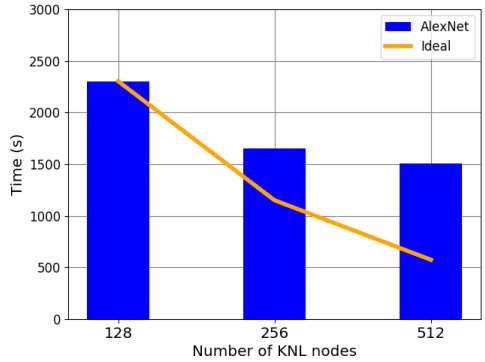


Figure 10: Strong Scaling Performance of AlexNet with 32k Batch Size on KNL Nodes

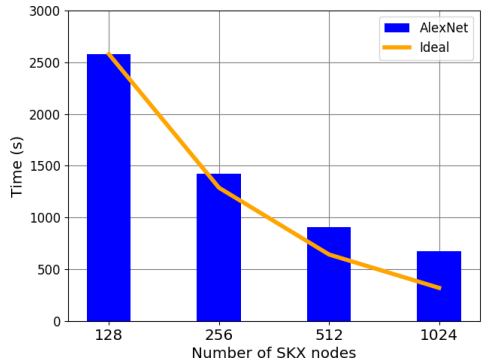


Figure 11: Strong Scaling Performance of AlexNet with 32k Batch Size on SKX Nodes

4.6 ImageNet training with ResNet-50

In this experiment, we use the ResNet-50 training case to evaluate the effectiveness of our approach in scalable DNN training. We use the ResNet-50 training on the ImageNet-1k dataset for 90 epochs with the batch size of 32k as the test case, run it at multiple scales on the KNL and SKX nodes on Stampede2, then compare the test accuracy and time-to-solution to the published results.

Figure 12 and 13 show the strong scaling performance of the ResNet-50 case at various scales. Compared to AlexNet, ResNet-50 scales more efficiently to 1,024 KNL nodes and 1,600 SKX nodes. This is because ResNet-50 has a relatively higher scaling ratio. In particular, the ResNet-50 case finishes in 32 minutes on 1,600 SKX nodes (3,200 Intel Xeon Platinum 8160 processors) and 20 minutes on 2,048 KNL nodes (2,048 Intel Xeon Phi 7250 processors).

A comprehensive result comparison against existing results is presented in Table 10. Codreanu *et al.* reported their experience on using Intel KNL clusters to speed up ImageNet-1k training by a blogpost[7]. They reported a 73.78% accuracy (with data augmentation) in less than 40 minutes on 512 KNL nodes with the batch size of 8k. However, this case only ran for 37 epochs. The complete 90-epoch training would take 80 minutes with a 75.25% accuracy.

[7]https://blog.surf.nl/en/imagenet-1k-training-on-intel-xeon-phi-in-less-than-40-minutes/

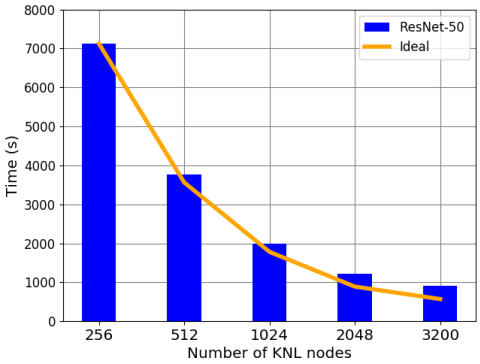


Figure 12: Strong Scaling Performance of ResNet-50 with 32k Batch Size on KNL Nodes

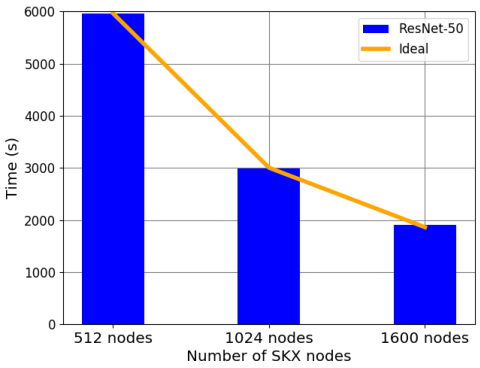


Figure 13: Strong Scaling Performance of ResNet-50 with 32k Batch Size on SKX Nodes

Table 10: ResNet-50 Result Comparison. DA means Data Augmentation

Batch	DA	epochs	accuracy	hardware	time
256	NO	90	73.0%	DGX-1 station	21h
256	YES	90	75.3%	16 KNLs	45h
8k	NO	90	72.7%	DGX-1 station	21h
8k	NO	90	72.7%	256 P100 GPUs	1h
8k	YES	90	75.3%	256 P100 GPUs	1h
16k	YES	90	75.3%	1024 SKX nodes	52m
16k	**YES**	**90**	**75.3%**	**1600 SKX nodes**	**31m**
32k	NO	90	72.6%	512 KNL nodes	1h
32k	YES	90	75.4%	512 KNL nodes	1h
32k	YES	90	75.4%	1024 SKX nodes	48m
32k	**YES**	**90**	**74.2%**	1600 SKX nodes	32m
32k	**YES**	**90**	**75.4%**	**2048 KNL nodes**	**20m**

Based on the original ResNet-50 model [13], we added data augmentation to our baseline. Our baseline is 75.3% top-1 test accuracy in 90 epochs. We failed to reproduce the reported 76.24% top-1 test accuracy form a group of Facebook researchers, as the model is not open sourced. The model we used is available upon request. The test accuracy comparison is shown in Table 11. Although our baseline's accuracy is lower than Facebook's, we achieve a correspondingly higher accuracy with batch sizes that are greater than 10k.

Table 11: Comparison by 90-epoch ResNet50 Accuracy. DA means Data Augmentation

Batch Size	256	8K	16K	32K	64K	DA
MSRA	75.3%	75.3%	—	—	—	weak
IBM	—	75.0%	—	—	—	—
SURFsara	—	75.3%	—	—	—	—
Facebook	76.3%	76.2%	75.2%	72.4%	66.0%	heavy
Our	73.0%	72.7%	72.7%	72.6%	70.0%	no
Our	75.3%	75.3%	75.3%	75.4%	73.2%	weak

5 CONCLUSION

In conclusion, we explore the large batch size approach to enable scalable DNN training on large scale computers. We examine the benefits and the challenges of this approach, and incorporate the LARS algorithm as the solution. We evaluate the implementation with the ImageNet-1k dataset and two neural network models of AlexNet and ResNet-50 at scale for the efficiency, test accuracy, training speed, and solution generality. Our solution is able to keep up with the baseline test accuracy for both test cases within the same number of epochs. We are able to reduce the ImageNet-1k training time from hours to minutes: With 1,024 SKX nodes, the AlexNet case finished in 11 minutes. While with 2,048 KNL nodes, the ResNet-50 case finished in 20 minutes. Our solution is general to be effective for both the AlexNet and ResNet-50 cases. We showcase large scale computers' capability in accelerating DNN training with massive computing resource with standard ImageNet-1k based benchmark. We believe this is a pilot use case to motivate future DNN based research on large scale computers across domains in both industry and academia.

6 ACKNOWLEDGEMENT

The Layer-wise Adaptive Rate Scaling (LARS) algorithm (simulated on single node) was developed by Y. You, B. Ginsburg, and I. Gitman when Y. You was an intern at NVIDIA [32].

REFERENCES

[1] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. 2017. Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes. *arXiv preprint arXiv:1711.04325* (2017).

[2] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. 2015. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595* (2015).

[3] Carol Reiley. 2016. Deep Driving. (2016). https://www.technologyreview.com/s/602600/deep-driving/.

[4] Bryan Catanzaro. 2013. Deep learning with COTS HPC systems. (2013).

[5] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).

[6] Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul, and Pradeep Dubey. 2016. Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709* (2016).

[7] 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.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 248–255.

[9] Jack Dongarra, Martin Meuer, Horst Simon, and Erich Strohmaier. 2017. Top500 supercomputer ranking. (2017). https://www.top500.org/lists/2017/06/

[10] 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).

[11] Hayit Greenspan, Bram van Ginneken, and Ronald M Summers. 2016. Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. *IEEE Transactions on Medical Imaging* 35, 5 (2016), 1153–1159.

[12] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing* 22, 6 (1996), 789–828.

[13] 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.

[14] JB Heaton, NG Polson, and JH Witte. 2016. Deep learning in finance. *arXiv preprint arXiv:1602.06561* (2016).

[15] Júlio Hoffimann, Youli Mao, Avinash Wesley, and Aimee Taylor. 2017. Sequence Mining and Pattern Analysis in Drilling Reports with Deep Natural Language Processing. *arXiv preprint arXiv:1712.01476* (2017).

[16] Forrest N. Iandola, Khalid Ashraf, Matthew W. Moskewicz, and Kurt Keutzer. 2015. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. *CoRR* abs/1511.00175 (2015). http://arxiv.org/abs/1511.00175

[17] Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, and Kurt Keutzer. 2016. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2592–2600.

[18] 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*. ACM, 675–678.

[19] Peter H Jin, Qiaochu Yuan, Forrest Iandola, and Kurt Keutzer. 2016. How to scale distributed deep learning? *arXiv preprint arXiv:1611.04581* (2016).

[20] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

[21] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997* (2014).

[22] Quoc V Le. 2013. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 8595–8598.

[23] Mu Li. 2017. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. Ph.D. Dissertation. Intel.

[24] 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.

[25] Rolf Rabenseifner. 2004. Optimization of collective reduction operations. In *International Conference on Computational Science*. Springer, 1–9.

[26] Rolf Rabenseifner and Jesper Larsson Träff. 2004. More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems. In *PVM/MPI*. Springer, 36–46.

[27] 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.

[28] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs.. In *Interspeech*. 1058–1062.

[29] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. On parallelizability of stochastic gradient descent for speech DNNs. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 235–239.

[30] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

[31] Robert A Vandegeijn. 1994. On global combine operations. *J. Parallel and Distrib. Comput.* 22, 2 (1994), 324–328.

[32] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Scaling SGD Batch Size to 32K for ImageNet Training. (2017).

[33] Yang You, Zhao Zhang, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2017. ImageNet Training in 24 Minutes. *arXiv preprint arXiv:1709.05011* (2017).

[34] Sixin Zhang, Anna E Choromanska, and Yann LeCun. 2015. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems*. 685–693.

[35] Sixin Zhang, Anna E Choromanska, and Yann LeCun. 2015. Deep learning with elastic averaging SGD. In *Advances in Neural Information Processing Systems*. 685–693.