# ImageNet Training by CPU: AlexNet in 11 Minutes and ResNet-50 in 48 Minutes

**Yang You**[1], **Zhao Zhang**[2], **Cho-Jui Hsieh**[3], **James Demmel**[1], **Kurt Keutzer**[1]
UC Berkeley[1], TACC[2], UC Davis[3]
{youyang, demmel, keutzer}@cs.berkeley.edu; zzhang@tacc.utexas.edu; chohsieh@ucdavis.edu

## Abstract

Since its creation, the ImageNet 1-k benchmark set has played a significant role as a benchmark for ascertaining the accuracy of different deep neural net (DNN) models on the classification problem. Moreover, in recent years it has also served as the principal benchmark for assessing different approaches to DNN training. Finishing a 90-epoch ImageNet-1k training with ResNet-50 on a NVIDIA M40 GPU takes 14 days. This training requires $10^{18}$ single precision operations in total. On the other hand, the world's current fastest supercomputer can finish $2 \times 10^{17}$ single precision operations per second. If we can make full use of the computing capability of a supercomputer for DNN training, we should be able to finish the 90-epoch ResNet-50 training in five seconds. Over the last two years a number of researchers have focused on how to close this significant performance gap through scaling DNN training to larger numbers of processors. Most successful approaches to scaling the training of ImageNet have used the approach of synchronous stochastic gradient descent. However, to scale synchronous stochastic gradient descent one must also increase the batch size used in each iteration.

Thus, for many researchers, the focus on scaling DNN training has translated into a focus on developing approaches that enable increasing the batch size in data-parallel synchronous stochastic gradient descent without losing accuracy over a fixed number of epochs. As a result, we have seen the batch size and number of processors successfully utilized increase from 1K batch/128 processors to 8K batch/256 processors over the last two years. The recently published LARS algorithm increased batch size further to 32K for some DNN models. Following up on this work, we wished to confirm that LARS could be used to further scale the number of processors efficiently used in DNN training and, and as a result, further reduce the total training time. In this paper we present the results of this investigation: using LARS we were able to efficiently utilize 1024 CPUs to finish the 100-epoch ImageNet training with AlexNet in 11 minutes, and we finished 90-epoch ImageNet training with ResNet-50 in 48 minutes (batch size = 32K). Furthermore, when we increase the batch size to above 20K, our accuracy is much higher than Facebook's on corresponding batch sizes (Figure 1). Our source code is available upon request. It will also be released in Intel Caffe.

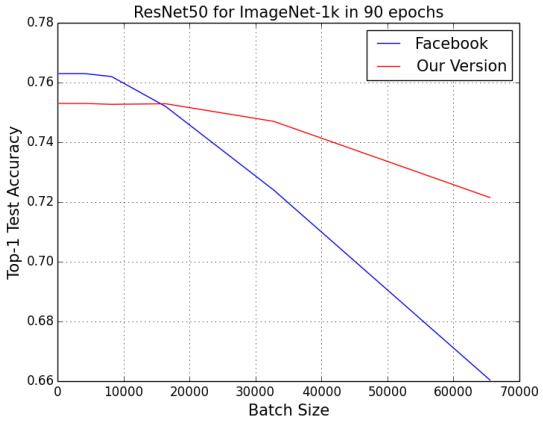



Figure 1: Because we use weaker data augmentation, our baseline's accuracy is slightly lower than Facebook's version (76.2% vs 75.4%). However, at very large batch sizes our accuracy is much higher than Facebook's accuracy. Facebook's accuracy is from their own report (Goyal et al. 2017). Our accuracy scaling efficiency is much higher than Facebook's version

## Introduction

For deep learning applications, larger datasets and bigger models lead to significant improvements in accuracy (Amodei et al. 2015), but at the cost of longer training times. Moreover, many applications such as computational finance, autonomous driving, oil and gas exploration, and medical imaging, will almost certainly require training data-sets with billions of training elements and terabytes of data. This highly motivates the problem of accelerating the training time of Deep Neural Nets (DNN). For example, finishing 90-epoch ImageNet-1k training with ResNet-50 on a NVIDIA M40 GPU takes 14 days. This training requires $10^{18}$ single precision operations in total. On the other hand, the world's current fastest supercomputer can finish $2 \times 10^{17}$ single precision operations per second (Dongarra et al. 2017). Thus, if we can make full use of the computing capability of a supercomputer for DNN training, we should be able to finish the 90-epoch ResNet-50 training in five seconds. So far, the best results on scaling ImageNet training have used synchronous stochastic gradient descent (syncronous SGD).

The synchronous SGD algorithm has many inherent advantages, but at the root of these advantages is *sequential consistency*. Sequential consistency implies 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. Continuing to scale the synchronous SGD model to more processors requires ensuring that there is sufficient useful work for each processor to do during each iteration. This, in turn, requires increasing the batch size used in each iteration. For example engaging 512 processors in synchronous SGD on a batch size of 1K would mean that each processor only processed a local batch of 2 images. If the batch size can batch size can be scaled to 32K then each processor processes a local batch of 64, and the computation to communication ratio can be more balanced.

As a result, over the last two years we have seen a focus on increasing the batch size and number of processors used in the DNN training for ImageNet-1K, with a resulting reduction in training time. In the following discussion we briefly review relevant work where all details of batch size, processors, DNN model, runtime, and training set are defined in the publications. All of the following refer to training on ImageNet.

FireCaffe (Iandola et al. 2015) (Iandola et al. 2016) demonstrated scaling the training of GoogleNet to 128 Nvidia K20 GPUs with a batch size of 1K for 72 epochs and a total training time of 10.5 hours. Although large batch size can lead to a significant loss in accuracy, using a warm-up scheme coupled with a linear scaling rule, researchers at Facebook (Goyal et al. 2017) were able to scale the training of ResNet 50 to 256 Nvidia P100's 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 a method they named the Layer-wise Adaptive Rate Scaling (LARS) algorithm (You, Gitman, and Ginsburg 2017), researchers were able to scale the batch size to very large sizes, such as 32K, although only 8 Nvidia P100 GPUs were employed. A 3.4% reduction in accuracy was attributed to the absence of data augmentation.

Given the large batch sizes that the LARS algorithm enables, it was natural to ask: how much further can we scale the training of DNNs on ImagNet? This is the investigation that led to this paper. In particular, we found that using LARS we could scale DNN training on ImageNet to 1024 CPUs and finish the 100-epoch training with AlexNet in 11 minutes. Further, we finish the 90-epoch ImageNet training with ResNet50 on 1024 CPUs in 48 minutes.

**Notes.** This paper is focused on training large-scale deep neural networks on $P$ machines/processors. We use $w$ to denote the parameters (weights of the networks), $w^j$ to denote the local parameters on $j$-th worker, $\tilde{w}$ to denote the global parameter. When there is no confusion we use $\nabla w^j$ to denote the stochastic gradient evaluated at the $j$-th worker. All the accuracy means top-1 test accuracy. There is no data augmentation in all the results.

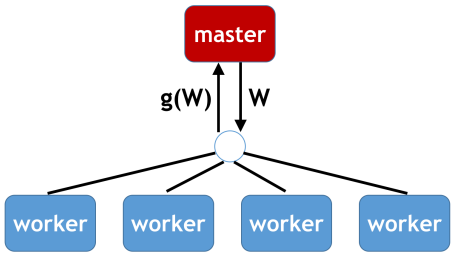
# Background and Related Work

## Data-Parallelism SGD

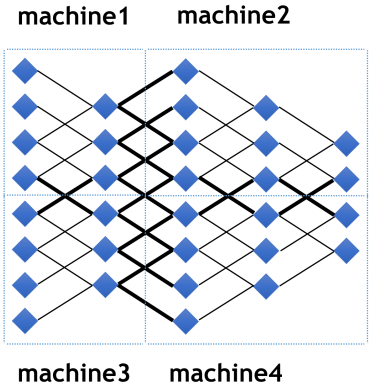
In data parallelism method, the dataset is partitioned into $P$ parts stored on each machine, and each machine will have a local copy of the neural network and the weights ($w^j$). In synchronized data parallelism, the communication includes two parts: sum of local gradients and broadcast of the global weight. For the first part, each worker computes the local gradient $\nabla w^j$ 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. For the second part, the master broadcasts $\tilde{w}$ to all workers. This synchronized approach is a widely-used method on large-scale systems (Iandola et al. 2016). Figure 2-(a) is an example of 4 worker machines and 1 master machine.

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. 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 communication and communication (Das et al. 2016) (Goyal et al. 2017), but the inherent synchronization barrier remains. A more radical approach to breaking this synchronization barrier is to pursue a purely asynchronous approach. A variety of asynchronous approaches have been proposed (Recht et al. 2011) (Zhang, Choromanska, and LeCun 2015a) (Jin et al. 2016) (Mitliagkas et al. 2016). 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 $\nabla w^j$ 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 (Dean et al. 2012), which has succesfully 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 (Seide et al. 2014b).

**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 can get the same solution as the single-machine case. Figure 2-(b) shows an example of using 4 machines to parallelize a 5-layer DNN. Model paralleism has been studied in (Catanzaro 2013; Le 2013). However, since the input size (e.g. size of an image) is relatively small, the matrix operations are not large. For example, parallelizing a $2048 \times 1024 \times 1024$ matrix multiplication only needs one or two machines. Thus, state-of-the-art methods often use data-parallelism (Amodei et al. 2015; Chen et al. 2016; Dean et al. 2012; Seide et al. 2014a).




(a) Data Parallelism




(b) Model Parallelism

Figure 2: (a) is an example of data parallelism. Each worker sends its gradients $\nabla w^j$ to the master, and the master updates its weights by $\tilde{w} \leftarrow \tilde{w} - \eta/P \sum_{i=1}^{P} \nabla w^j$. Then the master sends the updated weights $\tilde{w}$ to all the workers. (b) is an example of model parallelism. A five layer neural network with local connectivity is shown here, partitioned across four machines (blue rectangles). Only those nodes with edges that cross partition boundaries (thick lines) will need to have their state communicated between machines (e.g. by MPI (Gropp et al. 1996)). Even in cases where a node has multiple edges crossing a partition boundary, its state is only sent to the machine on the other side of that boundary once.

## Intel Knights Landing System

Intel Knights Landing (KNL) is the latest version of Intel's general-purpose accelerator. The major distinct features of KNL that can benefit deep learning applications include the following: **(1) Self-hosted Platform.** The traditional accelerators (e.g. FPGA, GPUs, and KNC) rely on CPU for control and I/O management. KNL does not need a CPU host. It is self-hosted by an operating system like CentOS 7. **(2) Better Memory.** KNL's measured bandwidth is much higher than that of a 24-core Haswell CPU (450 GB/s vs 100 GB/s). KNL's 384 GB maximum memory size is large enough to handle a typical deep learning dataset. Moreover, KNL is equipped with Multi-Channel DRAM (MCDRAM). MCDRAM's measured bandwidth is 475 GB/s. MCDRAM has three modes: a) Cache Mode: KNL uses it as the last level cache; b) Flat Mode: KNL treats it as the regular DDR; c) Hybrid Mode: part of it is used as cache, the other is used as the regular DDR memory. **(3) Configurable NUMA.** The basic idea is that users can partition the on-chip processors and cache into different groups for better memory efficiency and less communication overhead. This is very important for complicated memory-access applications like DNN training.

Since its release, KNL has been used in some HPC (High Performance Computing) data centers. For example, National Energy Research Scientific Computing Center (NERSC) has a supercomputer with 9,668 KNLs (Cori Phase 2). Texas Advanced Computing Center (TACC) has a supercomputer with 4,200 KNLs (Stampede 2).

In this paper, we have two chip options: (1) 1024 Intel Skylake CPUs or (2) 512 Intel KNLs. Using 1024 CPUs, we finish the 100-epoch AlexNet in 11 minutes and 90-epoch ResNet-50 in 48 minutes. Using 512 KNLs, we finish the 100-epoch AlexNet in 24 minutes and 90-epoch ResNet-50 in 60 minutes.

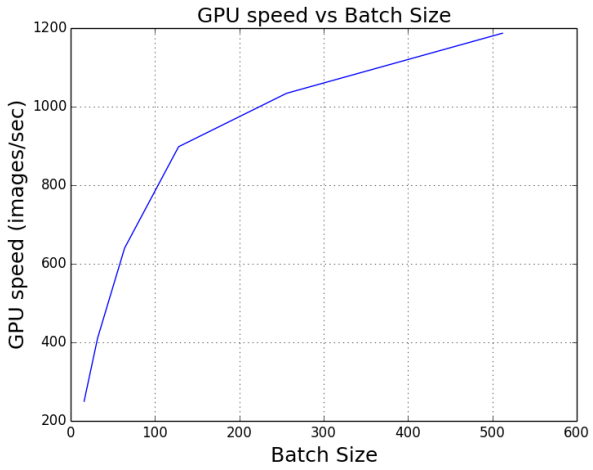



Figure 3: In a certain range, large batch improves the performance of system (e.g. GPU). The data in this figure is collected from training AlexNet by ImageNet dataset on NVIDIA M40 GPUs. Batch=512 per GPU gives us the highest speed. Batch=1024 per GPU is out of memory.

## Large-Batch DNN Training

### Benefits of Large-Batch Training

The asynchronous methods using parameter server are not guaranteed to be stable on large-scale systems (Chen et al. 2016). As discussed in (Goyal et al. 2017), data-parallelism synchronized approach is more stable for very large DNN training. The idea is simple—by using a **large batch size** for SGD, the work for each iteration can be easily 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 for ImageNet dataset, the number of operations is 90 * 1.28 Million * 7.72 Billion ($10^{18}$). Currently, the most powerful supercomputer can finish $200 \times 10^{15}$ single-precision operations per second (Dongarra et al. 2017). If there is an algorithm allowing us to make full use of the supercomputer, we can finish the ResNet-50 training in 5 seconds.

Table 1: Train neural networks by ImageNet dataset. $t_{comp}$ is the computation time and $t_{comm}$ is communication time. We fix the number of epochs as 100. Larger batch size needs much less iterations. Let us set batch size=512 per machine. Then we increase the number of machines. Since $t_{comp} \gg t_{comm}$ for using ImageNet dataset to train ResNet-50 networks and GPUs (Goyal et al. 2017), the single iteration time can be close to constant. Thus total time will be much less.

| Batch Size | Epochs | Iterations | GPUs | Iteration Time | Total Time |
|---|---|---|---|---|---|
| 512 | 100 | 250,000 | 1 | $t_{comp}$ | $250,000 \times t_{comp}$ |
| 1024 | 100 | 125,000 | 2 | $t_{comp}$ + log(2)$t_{comm}$ | $125,000 \times (t_{comp}$ + log(2)$t_{comm})$ |
| 2048 | 100 | 62,500 | 4 | $t_{comp}$ + log(4)$t_{comm}$ | $62,500 \times (t_{comp}$ + log(4)$t_{comm})$ |
| 4096 | 100 | 31,250 | 8 | $t_{comp}$ + log(8)$t_{comm}$ | $31,250 \times (t_{comp}$ + log(8)$t_{comm})$ |
| 8192 | 100 | 15,625 | 16 | $t_{comp}$ + log(16)$t_{comm}$ | $15,625 \times (t_{comp}$ + log(16)$t_{comm})$ |
| ... | ... | ... | ... | | |
| 1,280,000 | 100 | 100 | 2500 | $t_{comp}$ + log(2500)$t_{comm}$ | $100 \times (t_{comp}$ + log(2500)$t_{comm})$ |

Table 2: Standard Benchmarks for ImageNet training.

| Model | Epochs | Test Top-1 Accuracy |
|---|---|---|
| AlexNet | 100 | 58% (Iandola et al. 2016) |
| ResNet-50 | 90 | 75.3% (He et al. 2016) |

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. Let us use one NVIDIA M40 GPU to illustrate the case of a single machine. In a certain range, larger batch size will make the single GPU's speed higher (Figure 3). The reason is that low-level matrix computation libraries will be more efficient. For ImageNet training with Alexthe Net model the, optimal batch size per GPU is 512. If we want to use many GPUs and make each GPU efficient, we need a larger batch size. For example, if we have 16 GPUs, then we should set the batch size to $16 \times 512 = 8192$. 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 (Table 1).

## Model Selection

To scale up the algorithm to many machines, a major overhead is the communication among different machines (Zhang, Choromanska, and LeCun 2015b). Here we define **scaling ratio**, which means the 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 ResNet50. The reason is that they have different scaling ratios. From Table 5, we observe that ResNet50's scaling ratio is 12.5$\times$ larger than that of AlexNet. This means scaling ResNet50 is easier than scaling AlexNet. Generally, ResNet50 will have a much higher weak scaling efficiency than AlexNet.

In the fixed-epoch situation, large batch does not change the number of floating point operations (computation volume). However, large batch can reduce the communication volume. The reason is that the single-iteration communication volume is only related to the model size and network system. Larger batch size means less number of iterations and less overall communication. Thus, large batch size can improve the algorithm's scalability.

## Difficulty of Large-Batch Training

However, synchronous SGD with larger batch size usually achieves lower accuracy than when used with smaller batch sizes, if each is run for the same number of epochs, and currently there is no algorithm allowing us to effectively use very large batch sizes. (Keskar et al. 2016). Table 2 shows the target accuracy by standard benchmarks. For example, when we set the batch size of AlexNet larger than 1024 or the batch size of ResNet-50 larger than 8192, the test accuracy will be significantly decreased (Table 4 and Figure 4).

For large-batch training, we need to ensure that the larger batches achieve similar test accuracy with the smaller batches by running 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 approaches for large batch training include two techniques:

(1) **Linear Scaling** (Krizhevsky 2014): If we increase the batch size from $B$ to $kB$, we should also increase the learning rate from $\eta$ to $k\eta$.

(2) **Warmup Scheme** (Goyal et al. 2017): If we use a large learning rate ($\eta$). We should start from a small $\eta$ and increase it to the large $\eta$ in the first few epochs.

The intuition of linear scaling is related to the number of iterations. Let us use $B$, $\eta$, 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 means that the frequency of weight updating 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 3). However, we observe that state-of-the-art approaches can only scale batch size to 1024 for AlexNet and 8192 for ResNet-50. If we increase the batch size to 4096 for AlexNet, we only achieve 53.1% in 100 epochs (Table 4). Our target is to achieve 58% accuracy even when using large batch sizes.

### Scaling up Batch Size

In this paper, we use LARS algorithm (You, Gitman, and Ginsburg 2017) together with warmup scheme (Goyal et al. 2017) to scale up the batch size. Using these two approaches, synchronous SGD with a large batch size can achieve the same accuracy as the baseline (Table 6). To scale to larger batch sizes (e.g. 32k) for AlexNet, we need to change the local response normalization (LRN) to batch normalization (BN). We add BN after each Convolutional layer. Specifically, we use the refined AlexNet model by B. Ginsburg[1]. From Figure 4, we can clearly observe the effects of LARS. LARS can help ResNet-50 to preserve the high test accuracy. The current approaches (linear scaling and warmup) has much lower accuracy for batch size = 16k and 32k (68% and 56%). The target accuracy is about 73%.

## Experimental Results

### Experimental Settings.

The dataset we used in this paper is ImageNet-1k (Deng et al. 2009). The dataset has 1.28 million images for training and 50,000 images for testing. Without data augmentation, the top-1 testing accuracy of our ResNet-50 baseline is 73% in 90 epochs. For versions without data augmentation, we achieve state-of-the-art accuracy (73% in 90 epochs). With data augmentation, our accuracy is 75.4%. In the original paper, the accuracy of ResNet-50 is 75.3% after 90 epochs (He et al. 2016). We use the same network as the original ResNet-50 paper. For the KNL implementation, we have two versions:

(1) We wrote our KNL code based on Caffe (Jia et al. 2014) for single-machine processing and use MPI (Gropp et al. 1996) for the communication among different machines on KNL cluster.

(2) We use Intel Caffe, which supports multi-node training by Intel MLSL (Machine Learning Scaling Library).

We use the TACC Stampede 2 supercomputer as our hardware platform[2]. All GPU-related data are measured based on B. Ginsburg's nvcaffe[3]. The LARS algorithm is opened source by NVIDIA Caffe 0.16. We implemented the LARS algorithm based on NVIDIA Caffe 0.16.

[1]https://github.com/borisgin/nvcaffe-0.16/tree/caffe-0.16/models/alexnet_bn

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

[3]https://github.com/borisgin/nvcaffe-0.16

### ImageNet training with AlexNet

Previously, NVIDIA reported that using one DGX-1 station they were able to finish 90-epoch ImageNet training with AlexNet in 2 hours. However, they used half-precision or FP16, whose cost is half of the standard single-precision operation. We run 100-epoch ImageNet training with AlexNet with standard single-precision. It takes 6 hours 9 minutes for batch size = 512 on one NVIDIA DGX-1 station. Because of LARS algorithm (You, Gitman, and Ginsburg 2017), we are able to have the similar accuracy using large batch sizes (Table 6). If we increase the batch size to 4096, it only needs 2 hour 10 minutes on one NVIDIA DGX-1 station. Thus, using large batch can significantly speedup DNN training.

For the AlexNet with batch size = 32K, we scale the algorithm to 512 KNL chips (about 32K processors or cores). The batch size per KNL is 64, so the overall batch size is 32678. We finish the 100-epoch training in 24 minutes. When we use 1024 CPUs (batch size per CPU is 32), we finish 100-epoch AlexNet training in 11 minutes. To the best of our knowledge, this is the fastest 100-epoch ImageNet training with AlexNet. The overall comparison is in Table 7.

### ImageNet training with ResNet-50

Facebook (Goyal et al. 2017) finishes the 90-epoch ImageNet training with ResNet-50 in one hour on 32 CPUs and 256 NVIDIA P100 GPUs (similar to 32 DGX-1 stations). After scaling the batch size to 32K, we are able to more KNLs. We use 512 KNL chips and the batch size per KNL is 64. We finish the 90-epoch training in 48 minutes on 1024 CPUs (batch size = 32K). The version of our CPU chip is Intel Xeon Platinum 8160 (Skylake). The version of our KNL chip is Intel Xeon Phi Processor 7250. Note that we are not affiliated to Intel or NVIDIA, and we do not have any *a priori* preference for GPUs or KNL. The overall comparison is in Table 8.

Codreanu *et al.* reported their experience on using Intel KNL clusters to speed up ImageNet training by a blogpost[4]. They reported that they achieved 73.78% accuracy (with data augmentation) in less than 40 minutes on 512 KNLs. Their batch size is 8k. However, Codreanu *et al.* only finished 37 epochs. If they conduct 90-epoch training, the time is 80 minutes with 75.25% accuracy. In terms of absolute speed (images per second or flops per second), Facebook and our version are much faster than Codreanu *et al.* Since both Facebook and Codreanu used data augmentation, Facebook's 90-epoch accuracy is higher than that of Codreanu.

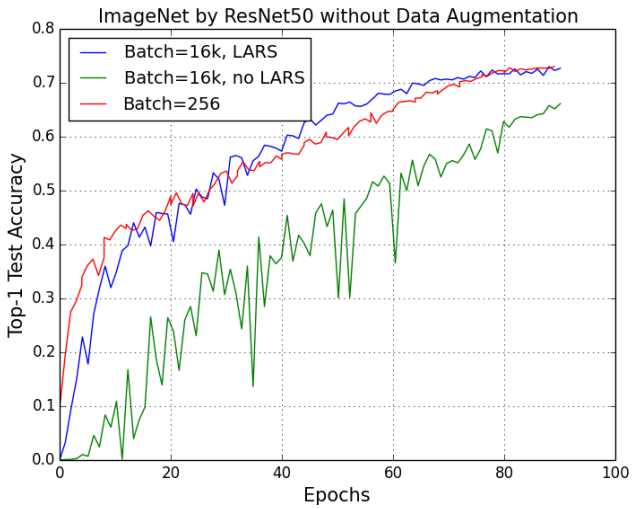
### ResNet-50 with Data Augmentation

Based on the original ResNet50 model (He et al. 2016), we added data augmentation to our baseline. The top-1 val accuracy of original ResNet50 is 75.3% in 90 epochs. Our baseline achieves 75.4% top-1 val accuracy in 90 epochs. Because we do not have Facebook's model file, we failed to reproduce full match their results of 76.24% top-1 accuracy. The model we used is available upon request. Codreanu *et al.* reported they achieved 75.81% top-1 accuracy in 90

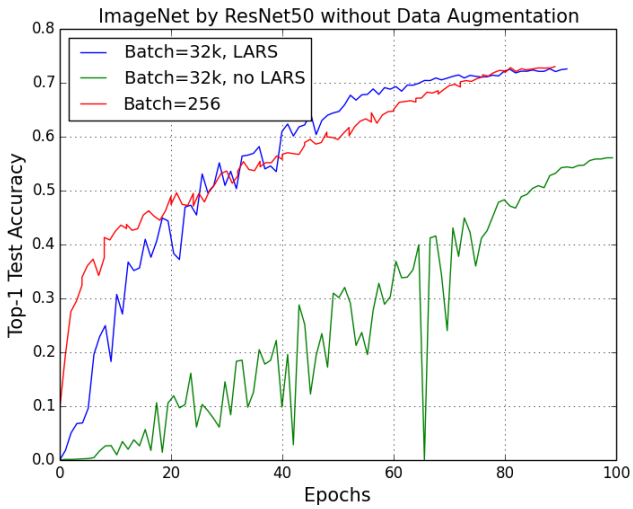
[4]https://blog.surf.nl/en/imagenet-1k-training-on-intel-xeon-phi-in-less-than-40-minutes/

Table 3: State-of-the-art large-batch training and test accuracy.

| Team | Model | Baseline Batch | Large Batch | Baseline Accuracy | Large Batch Accuracy |
|---|---|---|---|---|---|
| Google (Krizhevsky 2014) | AlexNet | 128 | 1024 | 57.7% | 56.7% |
| Amazon (Li 2017) | ResNet-152 | 256 | 5120 | 77.8% | 77.8% |
| Facebook (Goyal et al. 2017) | ResNet-50 | 256 | 8192 | 76.40% | 76.26% |




(a) Batch Size=16k




(b) Batch Size=32k

Figure 4: 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 (Goyal et al. 2017): 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.

epochs; however, they changed the model parameters (not only hyper-parameters). The overall comparison is in Table 9. We observe that our scaling efficiency is much higher than Facebook's version. Even though our baseline's accuracy is lower than Facebook's, we achieve a correspondingly higher accuracy when we increase the batch size above 10K. The accuracy-epoch curve of our version is shown in Figure 5.

## NVIDIA P100 GPU and Intel KNL

Because state-of-the-art models like ResNet50 are computational intensive, our comparison is focused on the computational power rather than memory efficiency. Since deep learning applications mainly use single-precision operations, we do not consider double-precision here. The peak performance of P100 GPU is 10.6 Tflops[5]. The peak performance of Intel KNL is 6 Tflops[6]. Based on our experience, the power of one P100 GPU is roughly equal to two KNLs. For example, we used 512 KNLs to match Facebook's 256 P100 GPUs. However, using more KNLs still requires the larger batch size.

[5]http://www.nvidia.com/object/tesla-p100.html

[6]https://www.alcf.anl.gov/files/HC27.25.710-Knights-Landing-Sodani-Intel.pdf

## Scaling Efficiency of Large Batch

To scale up deep learning, we need to a communication-efficient approach. Communication means moving data. On a shared memory system, communication means moving data between different level of memories (e.g. from DRAM to cache). On a distributed system, communication means moving the data over the network (e.g. master machine broadcast its data to all the worker machines). Communication often is the major overhead when we scale the algorithm on many processors. Communication is much slower than computation (Table 10). Also, communication costs much more energy than computation (Table 11).

Let us use the example ImageNet training with AlexNet-BN on 8 P100 GPUs to illustrate the idea here. The baseline's batch size is 512. Large batch is 4096. In this example, we focus the the communication among different GPUs. Firstly, our target is to make large batch achieve the same accuracy with small batch by using the same number of epochs (Figure 6). Fixing the number of epochs means fixing the number of floating point operations (Figure 7). If the system is not overloaded, large batch is much faster than small batch for using the same hardware (Figure 8). For finishing the same number of epochs, large batch's communication overhead is lower than small batch. Specifically, large batch

Table 4: Current approaches (linear scaling + warmup) do not work for AlexNet with batch size larger than 1024. 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 4096 should be 0.16. We use 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 |
| 1024 | 0.02 | no | 100 | 0.582 |
| 4096 | 0.01 | yes | 100 | 0.509 |
| 4096 | 0.02 | yes | 100 | 0.527 |
| 4096 | 0.03 | yes | 100 | 0.520 |
| 4096 | 0.04 | yes | 100 | 0.530 |
| 4096 | 0.05 | yes | 100 | 0.531 |
| 4096 | 0.06 | yes | 100 | 0.516 |
| 4096 | 0.07 | yes | 100 | **0.001** |
| ... | ... | ... | ... | ... |
| 4096 | 0.16 | yes | 100 | **0.001** |

Table 5: Scaling Ratio for AlexNet and ResNet50.

| Model | communication # parameters | computation # flops per image | comp/comm scaling ratio |
|---|---|---|---|
| AlexNet | # 61 million | # 1.5 billion | 24.6 |
| ResNet50 | # 25 million | # 7.7 billion | 308 |

sends less message (latency overhead) and move less data (bandwidth overhead) than small batch. For Sync SGD, the algorithm needs to conduct an all-reduce operations (sum of gradients on all machines). The number of messages sent is linear with the number of iterations. Also, because the gradients has the same size with the weights ($|W|$). Let us use $E$, $n$, and $B$ to denote the number of epochs, the total number of pictures in the training dataset, and the batch size, respectively. Then we can get the number of iterations is $E \times n/B$. Thus, when we increase the batch size, we need much less number of iterations (Table 1 and Figure 9). The number of iterations is equal to the number of messages the algorithm sent (i.e. latency overhead). 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, large batch needs to move much less data than small batch when they finish the number of floating point operations (Figures 10 and 11). In summary, large batch does not change the number of floating point operations when we fix the number of epochs. Large batch can increase the computation-communication ratio because it reduces the communication overhead (reduce latency and move less data). Large batch makes the algorithm more scalable on distributed systems.

Table 6: ImageNet Dataset with AlexNet Model. 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 batch size=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 |

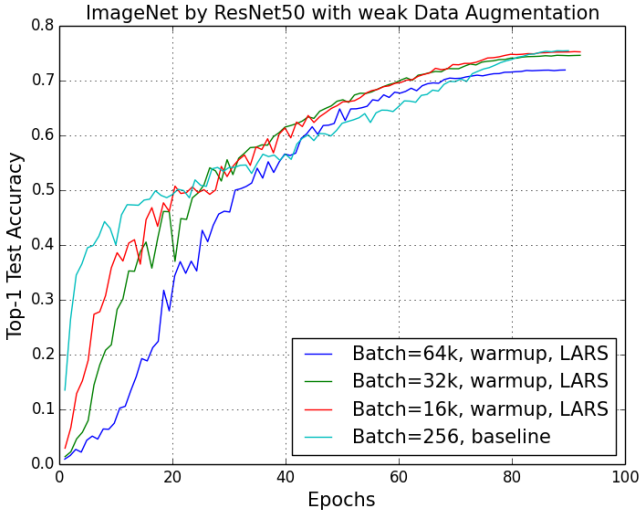



Figure 5: We use the same model and data augmentation with the original ResNet50 (He et al. 2016), which achieves 75.3% accuracy. Our baseline achieves 75.4% accuracy. We can scale the batch size to 32k, which only lost 0.6% accuracy. The accuracy loss is larger than 1% for the batch size larger than 32K, which is not acceptable.

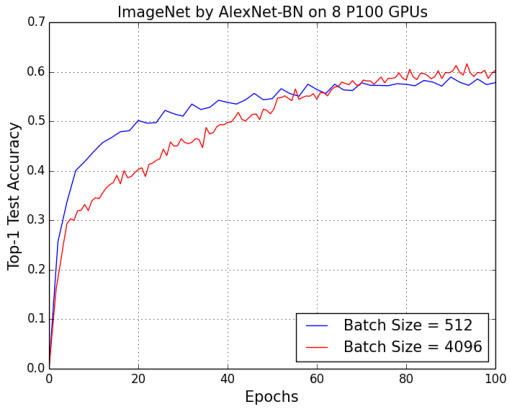



Figure 6: From this figure, we observe that we can achieve the target accuracy in the same number of epochs by using large batch size. Batch Size = 512 is the baseline.

Table 7: For batch size=32K, we changed local response norm in AlexNet to batch norm.

| Batch Size | epochs | Peak Top-1 Accuracy | hardware | time |
|---|---|---|---|---|
| 256 | 100 | 58.7% | 8-core CPU + K20 GPU | 144h |
| 512 | 100 | 58.8% | DGX-1 station | 6h 10m |
| 4096 | 100 | 58.4% | DGX-1 station | 2h 19m |
| 32K | 100 | 58.5% | 512 KNLs | 24m |
| 32K | 100 | 58.6% | 1024 CPUs | 11m |

Table 8: ResNet50 Results. We use the same data augmentation with the original ResNet-50 model (He et al. 2016).

| Batch Size | Data Augmentation | epochs | Peak Top-1 Accuracy | hardware | time |
|---|---|---|---|---|---|
| 256 | NO | 90 | 73.0% | DGX-1 station | 21h |
| 256 | YES | 90 | 75.4% | 16 KNLs | 45h |
| 8192 | NO | 90 | 72.7% | DGX-1 station | 21h |
| 8192 | NO | 90 | 72.7% | 32 CPUs + 256 P100 GPUs | 1h |
| 8192 | NO | 90 | 75.3% | 32 CPUs + 256 P100 GPUs | 1h |
| 32K | NO | 90 | 72.6% | 512 KNLs | 1h |
| 32K | YES | 90 | 74.7% | 512 KNLs | 1h |
| 32K | YES | 90 | 74.7% | 1024 CPUs | 48m |

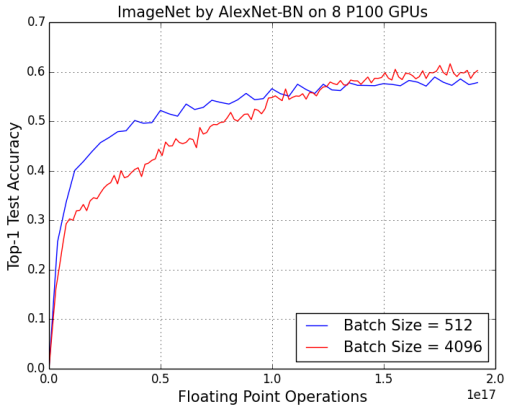



Figure 7: Increasing the batch size does not increase the number of floating point operations. Large batch can achieve the same accuracy in the fixed number of floating point operations.

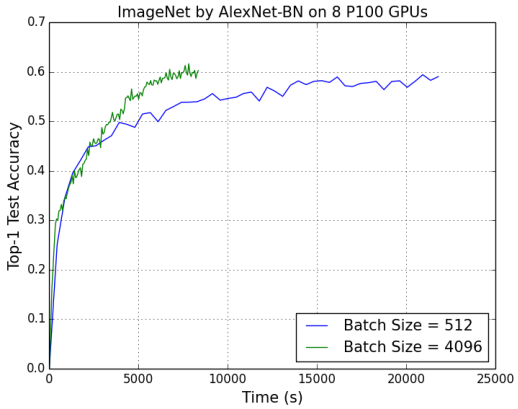



Figure 8: When we have enough computational powers, large batch is much faster than the small batch. To achieve 58% accuracy, large batch (batch size = 4096) only needs about two hours while small batch (batch size = 512) needs about six hours. Large batch and small batch finish the same number of floating point operations.

## Conclusion

In recent years the ImageNet 1K benchmark set has played a significant role as a benchmark for assessing different approaches to DNN training. The most successful results on accelerating DNN training on ImageNet have used a synchronous SGD approach. To scale this synchronous SGD approach to more processors requires increasing the batch size. Using a warm-up scheme coupled with a linear scaling rule, researchers at Facebook (Goyal et al. 2017) were able to scale the training of ResNet 50 to 256 Nvidia P100's 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 a method they named the Layer-wise Adaptive Rate Scaling (LARS) algorithm (You, Gitman, and Ginsburg 2017), researchers were able to scale the batch size to 32K; however, the potential for scaling to larger number of processors was not demonstrated in that work, and only 8 Nvidia P100 GPUs were employed. Also, data augmentation was not used in that work, and accuracy was impacted. In this paper we confirmed that the increased batch sizes afforded by the LARS algorithm could lead to increased scaling. In particular, we scaled synchronous SGD batch size to 32K and using 1024 Intel Skylake CPUs we were able to finish the 100-epoch ImageNet training with AlexNet in 11 minutes. We were able to finish 90-epoch ImageNet training with ResNet-50 in 48 minutes on 1024 CPUs (batch size = 32K). We also explored the impact of data augmentation in our work.

Table 9: Overall Comparison by 90-epoch ResNet50 Top-1 Val Accuracy.

| Batch Size | 256 | 8K | 16K | 32K | 64K | note |
|---|---|---|---|---|---|---|
| MSRA | 75.30% | 75.27% | — | — | — | weak data augmentation |
| IBM | — | 75.00% | — | — | — | — |
| SURFsara | — | 75.25% | — | — | — | Changed the network |
| Facebook | 76.30% | 76.20% | 75.20% | 72.40% | 66.04% | Heavy data augmentation |
| Our version | 73.00% | 72.70% | 72.70% | 72.60% | 70.00% | no data augmentation |
| Our version | 75.40% | 75.27% | 75.30% | 74.70% | 72.00% | weak data augmentation |

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

| Network | $\alpha$ (latency) | $\beta$ (1/bandwidth) |
|---|---|---|
| Mellanox 56Gb/s FDR IB | $0.7 \times 10^{-6}$s | $0.2 \times 10^{-9}$s |
| Intel 40Gb/s QDR IB | $1.2 \times 10^{-6}$s | $0.3 \times 10^{-9}$s |
| Intel 10GbE NetEffect NE020 | $7.2 \times 10^{-6}$s | $0.9 \times 10^{-9}$s |

Table 11: Energy table for 45nm CMOS process (Horowitz ). Communication costs much more energy than computation.

| Operation | Type | Energy (pJ) |
|---|---|---|
| 32 bit int add | Computation | 0.1 |
| 32 bit float add | Computation | 0.9 |
| 32 bit register access | Communication | 1.0 |
| 32 bit int multiply | Computation | 3.1 |
| 32 bit float multiply | Computation | 3.7 |
| 32 bit SRAM access | Communication | 5.0 |
| 32 bit DRAM access | Communication | 640 |

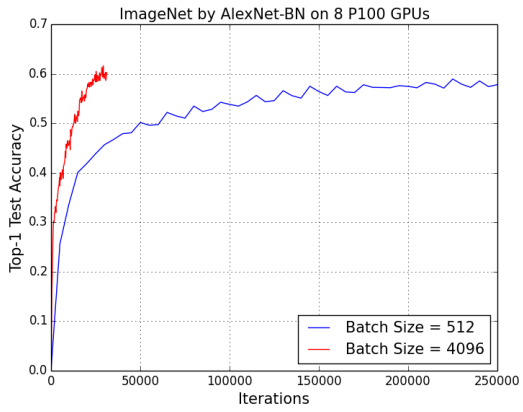



Figure 9: When we fix the number of epochs and increase the batch size, we need much less iterations.

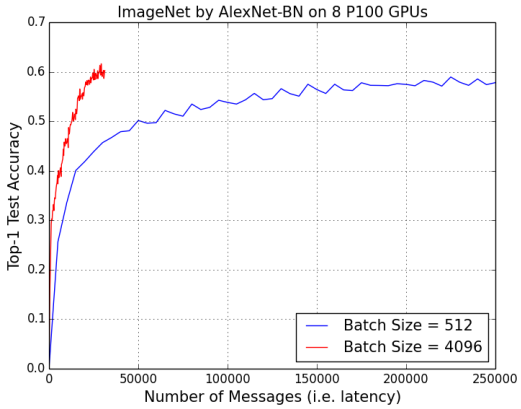



Figure 10: When we fix the number of epochs and increase the batch size, we need much less iterations. The number of iterations is linear with the number of messages the algorithm sent.

## Acknowledgement

The large batch training algorithm was developed jointly with I.Gitman and B.Ginsburg done during Yang You's internship in NVIDIA in the summer 2017. The work presented in this paper was supported by the National Science Foundation, through the Stampede 2 (OAC-1540931) award. JD and YY are supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-SC0010200; by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research under Award Numbers DE-SC0008700; by DARPA Award Number HR0011-12- 2-0016, ASPIRE Lab industrial sponsors and affiliates Intel, Google, HP, Huawei, LGE, Nokia, NVIDIA, Oracle and Samsung. Other industrial sponsors include Mathworks and Cray. In addition to ASPIRE sponsors, KK is supported by an auxiliary Deep Learning ISRA from Intel. CJH also thank XSEDE and Nvidia for independent support.

## References

Amodei, D.; Anubhai, R.; Battenberg, E.; Case, C.; Casper, J.; Catanzaro, B.; Chen, J.; Chrzanowski, M.; Coates, A.; Diamos, G.; et al. 2015. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*.

Catanzaro, B. 2013. Deep learning with cots hpc systems.

Chen, J.; Monga, R.; Bengio, S.; and Jozefowicz, R. 2016. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*.

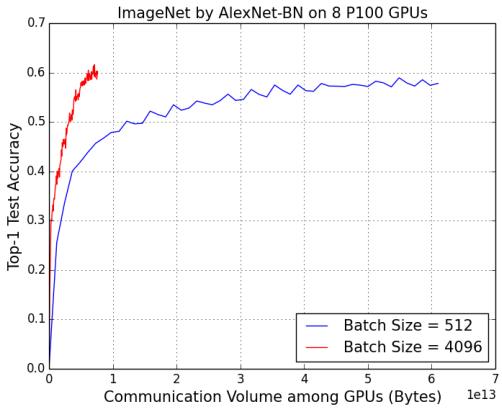



Figure 11: Let us use $E$, $n$, and $B$ to denote the number of epochs, the total number of pictures in the training dataset, and the batch size, respectively. Then we can get the number of iterations is $E \times n/B$. When we fix the number of epochs and increase the batch size, we need much less iterations. 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, large batch needs to move much less data than small batch when they finish the number of floating point operations.


Das, D.; Avancha, S.; Mudigere, D.; Vaidynathan, K.; Sridharan, S.; Kalamkar, D.; Kaul, B.; and Dubey, P. 2016. Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709*.

Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Senior, A.; Tucker, P.; Yang, K.; Le, Q. V.; et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*, 1223–1231.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 248–255. IEEE.

Dongarra, J.; Meuer, M.; Simon, H.; and Strohmaier, E. 2017. Top500 supercomputer ranking.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Gropp, W.; Lusk, E.; Doss, N.; and Skjellum, A. 1996. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing* 22(6):789–828.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Horowitz, M. Energy table for 45nm process.

Iandola, F. N.; Ashraf, K.; Moskewicz, M. W.; and Keutzer, K. 2015. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. *CoRR* abs/1511.00175.

Iandola, F. N.; Moskewicz, M. W.; Ashraf, K.; and Keutzer, K. 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.

Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; and Darrell, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, 675–678. ACM.

Jin, P. H.; Yuan, Q.; Iandola, F.; and Keutzer, K. 2016. How to scale distributed deep learning? *arXiv preprint arXiv:1611.04581*.

Keskar, N. S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; and Tang, P. T. P. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

Krizhevsky, A. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

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

Li, M. 2017. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. Ph.D. Dissertation, Intel.

Mitliagkas, I.; Zhang, C.; Hadjis, S.; and Ré, C. 2016. Asynchrony begets momentum, with an application to deep learning. In *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*, 997–1004. IEEE.

Recht, B.; Re, C.; Wright, S.; and Niu, F. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 693–701.

Seide, F.; Fu, H.; Droppo, J.; Li, G.; and Yu, D. 2014a. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Interspeech*, 1058–1062.

Seide, F.; Fu, H.; Droppo, J.; Li, G.; and Yu, D. 2014b. On parallelizability of stochastic gradient descent for speech dnns. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, 235–239. IEEE.

You, Y.; Gitman, I.; and Ginsburg, B. 2017. Scaling sgd batch size to 32k for imagenet training.

Zhang, S.; Choromanska, A. E.; and LeCun, Y. 2015a. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, 685–693.

Zhang, S.; Choromanska, A. E.; and LeCun, Y. 2015b. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, 685–693.