2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)

Large Minibatch Training on Supercomputers with Improved Accuracy and Reduced Time to Train

Valeriu Codreanu
Compute Services
SURFsara B.V.
Amsterdam, The Netherlands
Email: valeriu.codreanu@surfsara.nl

Damian Podareanu
Compute Services
SURFsara B.V.
Amsterdam, The Netherlands
Email: damian.podareanu@surfsara.nl

Vikram Saletore
AI Products Group
Intel Corp.
Hillsboro, United States of America
Email: vikram.a.saletore@intel.com

***Abstract*—For the past 6 years, the ILSVRC competition and the ImageNet dataset have attracted a lot of interest from the Computer Vision community, allowing for state-of-the-art accuracy to grow tremendously. This should be credited to the use of deep artificial neural network designs. As these became more complex, the storage, bandwidth, and compute requirements increased. This means that with a non-distributed approach, even when using the most high-density server available, the training process may take weeks, making it prohibitive. Furthermore, as datasets grow, the representation learning potential of deep networks grows as well by using more complex models. This synchronicity triggers a sharp increase in the computational requirements and motivates us to explore the scaling behaviour on petaflop scale supercomputers. In this paper we describe the challenges and novel solutions needed in order to train ResNet-50 in a large scale environment. We demonstrate above 90 percent scaling efficiency and a training time of 28 minutes using up to 104K x86 cores. This is supported by software tools from Intel's ecosystem. Moreover, we show that with regular 90 - 120 epoch train runs we can achieve a top-1 accuracy as high as 77 percent for the unmodified ResNet-50 topology. We also introduce the novel Collapsed Ensemble technique that allows us to obtain a 77.5 percent top-1 accuracy, similar to that of a ResNet-152, while training a unmodified ResNet-50 topology for the same fixed training budget.**

***Keywords*-deep learning; scaling; convergence; large mini-batch; ensembles;**

I. Introduction

The popularity of deep neural network (DNN) approaches has grown steadily since the, by now famous, AlexNet architecture demonstrated unprecedented levels of performance on the computer vision ImageNet [1] classification challenge in 2012 [2]. This seminal paper paved the way to more and more neural network research, particularly applied to Computer Vision. All subsequent ImageNet challenges have been won by deep neural networks. Due to many neural network training innovations (e.g. residual connections, dropout, batch normalization, etc.) the accuracy on this complex, 1000-category dataset was significantly reduced over the years to a mere 2.25% top-5 error on the test set by an ensemble of Squeeze and Excitation Networks [3] in 2017. Besides the major impact in this field, end-to-end DNNs have been applied in other fields like voice recognition and natural language processing [4] obtaining state-of-the-art accuracies. Recently, Silver *et al.* [5] obtained a remarkable achievement for the game of Go and improved the previous research performed for the AlphaGo agent.

These innovations could not have happened if the research community wouldn't have had a computing platform that efficiently supported linear algebra computation, heavily present in deep neural network training. By training in hours instead of days, researchers could benefit from a shorter research cycle in order to validate their ideas quicker. The single machine approaches have been dominated by GPUs that filled this role for the last 5 years [6]. However, as DNN architectures evolve, training "high-accuracy" networks using a single card, or even a single server becomes prohibitive. Presently, some of the most popular DNN designs are based on residual blocks, inspired by the design proposed by He *et al.* [7] in 2015. Training a 50 layer residual network, ResNet-50, on the ImageNet-1K dataset takes around 10 days using an NVIDIA P100 card. Training a larger ResNet-152 in the same setting would take roughly 3 weeks. As the size and complexity of the training datasets increases, supervised classification systems become even more accurate [8]. This increases of course the time-to-trained model. By continuing the previous example, it would take roughly one year to train a ResNet-152 model using the full ImageNet-22K dataset, which is about one order of magnitude larger than ImageNet-1K. Decreasing this prohibitive execution time by using modern supercomputers is the main focus of this work.

There has been a lot of research around scaling stochastic gradient descent based machine learning algorithms [9], [10], [11], [12], [13]. Most of these works, as well as our current work focus on data parallel scaling. This effectively means that at each training iteration a large minibatch is evenly divided among the workers. Historically, many algorithms would not scale to large minibatch sizes, as this would either hinder the convergence abilities of the underlying network, or would need many more training epochs in order to reach the desired validation accuracy. Keskar *et al.* [14]

978-1-7281-0180-4/18/$31.00 ©2018 IEEE
DOI 10.1109/MLHPC.2018.000-2

concluded that large-batch training leads to a generalization gap, meaning that the trained networks perform poorly on the validation set. This happens because in the large-batch regime training seems to converge to sharp minima of the training and testing functions. The research from [15] suggests that the generalization gap can be partially closed by just training longer. Another recent approach by Dinh *et al.* [16] promotes reparametrization as a means to improve the geometry of the minima. On the other hand, Facebook's research [10] suggests that large-batch training is more an optimization problem, that can be solved up to a global batch size of 8192 using a linear learning rate scaling and gradual warm-up. Another notable example of large-scale DNN training on non-ImageNet scientific problems using the Intel Knights Landing based Cori-2 system is the one from [17].

The experiments presented in this work push large-batch training research forward, both in terms of scaling efficiency and preserving a high validation accuracy for large mini-batches of up to 65536 images.

The main contributions of this paper are:

- We propose different learning schedules that set new accuracy standards for the unmodified ResNet-50 residual network architecture on the ImageNet-1K dataset.
- We scale out synchronous SGD deep learning training on up to 1536 Knights Landing (KNL) nodes or 512 Intel 2CPU Xeon Skylake (SKX) nodes. We report convergence in 28 minutes using 1536 KNL nodes. We also obtain SOTA accuracies on batch sizes up to 64K.
- We propose a collapsing ensemble technique as a reasonable approach for increasing the accuracy of any deep learning model while keeping the training budget constant.

In Section 2 we present the background and related works on this subject. We describe the optimized communication stack that we used in Section 3. In Section 4 we present our experimental and execution methodology. We present and discuss our results in detail in Section 5. In Section 6 we present our novel technique to accelerate distributed training using ensembles. Section 7 presents how we can use multi-NUMA domains on Intel CPUs to significantly improve hardware efficiency. Section 8 gives some conclusions and future research directions.

II. Background

In the last year, a lot of research has been conducted on scaling DNNs to a large collection of compute nodes, as the data and network sizes grossly exceeded the compute capabilities of single servers. It is definitely a trend set to continue, as it was suggested already [8] that if the data neural networks are trained on is expanded, so is the performance achieved by the networks.

Despite the abundance of newly developed methods and techniques, only a few seriously consider the impact of large-batch training on the validation accuracy. Notably, Goyal *et al.* [10] is one of the few that report competitive validation accuracy when scaling ResNet-50 training to 32 GPU-enabled servers. His work is revolutionary, showing excellent scaling properties with the Caffe2 framework, and actually achieving around 1% better top-1 validation accuracy than the one obtained by the original ResNet authors [7]. We adopted the 5 epoch gradual warm-up technique with momentum correction, as well as their proposed batch normalization initialization of the γ parameter in our training procedure ($\gamma = 0$ for the final BN layer of each residual block). However, we also introduce different techniques in order to overcome the large-batch convergence issues, and compared to their work, we perform the training process at a much larger scale, up to 1536 servers. [11] also published ResNet-50 scaling results, using the Caffe and Torch frameworks. Although their 50 minutes ResNet-50 training result on ImageNet-1K is faster than the 60 minutes obtained by [10] using comparable amounts of hardware, the achieved top-1 validation accuracy is lower, at around 75%. We think that maintaining state-of-the-art accuracy is of paramount importance when scaling up the batch size.

Another related work is the one described in [12]. This is similar to our work in terms of scaling and type of hardware used. They introduce the LARS (Layer-wise Adaptive Rate Scaling) technique to cope with the large-batch training, and similarly to our work they scale the batch size up to 64K. In a recent version of their paper, they have improved the validation accuracy by using data augmentation, similar to our work. The LARS technique seems to help with the optimization difficulties at large batch sizes, and they show they can scale up to 16K batch size without accuracy loss. We note that we can use a batch size of 32K, and still maintain state-of-the-art accuracy. In the same large-batch training context, recent research from [18] suggests that the batch size can be expanded during training to up to 64K for ImageNet-1K. However, although the authors use a better baseline than the ResNet-50, namely the much heavier Inception-ResNet v2 architecture [19], they degrade the model performance from above 80% to below 77% top-1 accuracy when using large batches. They achieve this result in only 2500 SGD updates. We performed a similar experiment, using 2100 updates and ResNet-50, and show we can achieve around 74% accuracy, less than 1.5% degradation compared to ResNet-50's baseline.

As far as we are aware, besides the research from [10] and [12], most other ResNet-50 scaling works report a validation accuracy lower than the 75.3% presented in the original paper on residual networks [7]. In our view, this also makes the timing results not fully comparable. As we will present in this current work, there is a clear trade-off between the time needed to train a model and the accuracy that is achieved by the model. When training the popular ResNet-50 architecture on ImageNet-1K, the resulting model achieves a

top-1 validation accuracy of 76.83% using a single 224x224 center crop evaluation, by efficiently engineering the training procedure, but without increasing the number of epochs. Moreover, when using the collapsed ensemble technique, we can reach 77.5% accuracy with mojority voting, exceeding ResNet-152 performance with a ResNet-50 training budget. All results are achieved using servers equipped solely with Intel CPUs.

III. Distributed Deep Learning using Intel Caffe

All the experiments performed for this research rely on Intel's optimized branch of the Caffe framework [20]. Intel Caffe is quickly following the developments of Caffe's master, but has a clear focus on achieving the highest training performance for Intel architectures. Intel's ML-SL [21] (Machine Learning Scaling Library) is a software library that efficiently deals with the communication involved in neural networks. It is tightly coupled to the training framework, allowing simultaneous compute and communication when performing the backward propagation pass. ML-SL allows for both data and model parallelism, and does that while efficiently using the bandwidth offered by high-speed interconnects such as Intel's Omni-Path Architecture (OPA). When using model parallelism the model is divided across the participating workers, and workers communicate both in the forward and backward passes. This has the potential to accommodate larger models, as each worker will hold a part of the parameter set. On the other side, in the case of data parallelism, the parameters are replicated on each worker, but communication happens only in the backward pass. We have efficiently scaled synchronous SGD (SSGD) using ML-SL to 1536 KNL nodes with ˜104K x86 cores. In this work we will only focus on the data parallel case, as we believe current hardware is more efficiently used in this manner, especially since CPU-based servers feature enough memory to hold even the largest models.

IV. Experimental methodology

A. ResNet-50 training methodology

The goal of our research is the fastest time to a given validation accuracy level. We worked on achieving this goal using the ResNet-50 architecture, and altered parts of the training methodology. These are described below. We evaluate our method on the ImageNet 2012 classification dataset [1] that consists of 1.28 million training images split across 1000 classes. By using this training procedure our models can achieve validation accuracy much higher than the ones presented by [7], peaking at close to 77% top-1 accuracy after 120 epochs. As previously mentioned, we use all the 50,000 images from the ILSVRC2012 validation set for evaluating the performance of our models. As for data augmentation, we only employ basic scale and aspect ratio augmentation. All model evaluations were done on a single 224x224 center crop extracted from the 256x256 resized input image, unless otherwise noted. Also, our models were trained from scratch. We used standard values for the hyperparameters. The momentum value was set to 0.9, and the weight decay, λ, to 0.0001, as these values seem to work quite well up to a relatively large batch size. To push the performance of the model further, we change the weight decay value dynamically during training as will be further explained in Section V-B.

Table I: 3-step decay versus linear decay of the learning rate

Deacay	Batch size	# nodes	# epochs	top-1/top-5 accuracy
Linear	7680	240	90	75.68/92.95
3-Step	7680	240	90	75.44/92.69

B. Batch normalization considerations

Before starting the experiments on ImageNet-1K using the ResNet-50 architecture, we performed an extensive set of experiments using the Inception-v1 architecture, that does not feature batch normalization layers [22]. It was noticeable that this was limiting the model performance when using a large-batch for training. During training we used normalization over the current minibatch and global statistics were accumulated by a moving average. During testing, the accumulated mean and variance values were used for normalization. Smaller values make the moving average decay faster, giving more weight to the recent values. With each iteration the moving average is updated with the current mean becoming $S_t = (1 - \beta)Y_t + \beta \cdot S_{t-1}$, where β is the moving average fraction parameter. For the moving average fraction, we have empirically set the value of the parameter from Caffe's BatchNorm layer to 0.95 for all BN layers.

C. Learning rate schedule

The most widely used method to decay the learning rate is the classical 3-step 10-fold decrease. Empirically, we noticed that it is more effective to use polynomial decay with the power of 1 for the learning rate decay schedule (basically linear decay, same as the linear increase from the warm up phase), instead of the classic 3-step decrease. To ensure that this is the case we have performed some ablation experiments. An example comparative experiment for a 90 epoch training run using 240 KNL nodes is presented in Table I. For this experiment we kept all settings fixed besides changing the learning rate from a 3-step decay, at epochs 30, 60, and 80, to a linear decay. We note that both runs include a warm-up phase of 5 epochs out of the 90 epoch training budget, as described in [10], and the value from which the learning rate decreases follows the linear scaling rule proposed in the same work. All further experiments in the paper use a polynomial decrease of the LR.

When using a polynomial decay, the learning rate decreases from its original value to 0 over the number of training iterations. This allows us to easily control the duration of the training, the only detail changing between

a quick run and a full run being the decay slope of the learning rate. Up to a global batch size of 16K we noticed that we can achieve good model performance even quicker than 90 epochs. To achieve the last bit of performance we perform the final 3-5 epochs in a "collapsed" fashion with augmentation disabled, as will be explained in Section VI. We therefore set four performance levels for ResNet-50, namely:

- 75.5% top-1 accuracy achieved using 48 training epochs. The results from the original ResNet implementation [7] falls in this category.
- 76% top-1 accuracy achieved using 64 training epochs. The result from [10] falls in this category.
- 76.5% top-1 accuracy achieved using 78 training epochs. This is a state-of-the-art result for large-batch training.
- a new SOTA (state-of-the-art) 77% top-1 accuracy when training for 120 training epochs.

Sections V and VI describe both scaling and convergence results in more detail.

D. Execution methodology

All experiments were executed on either Intel KNL based servers, or dual-socket Intel SKX servers. We use three separate HPC infrastructures for this study:

- TACC's Stampede2 [1] system composed of Intel Knights Landing 7250 nodes. All experiments are performed with the nodes set in Cache-Quadrant mode. We use this system to perform the KNL scaling experiments. This system was also recently extended with dual-socket SKX 8160 nodes, and we have also used this for the experiments in Section VII.
- BSC's MareNostrum4 [2] system composed of dual-socket SKX 8160 nodes. We use this system to perform Skylake scaling experiments.
- An internal Intel KNL [3] cluster with 192GB of RAM per node, and large local storage. The KNLs in this system are configured in Flat-Quadrant mode. We use this system for ablation studies and for training exploration.

When using distributed training with Intel Caffe and Intel ML-SL, explicitly choosing the number and IDs of cores that are performing communication and computation is essential. In the case of the KNL experiments, the affinity is set explicitly so that the ML-SL processes use the last 4 cores to perform communication. The first 64 cores from the core list take part in the OpenMP team for computation. In the SKX case, since it is a dual socket system, we have noticed that we achieve the best performance by pinning 2 parameter servers on each of the sockets, and 22 OpenMP threads on each socket, for a total of 44 OpenMP threads participating in the computation, as each SKX 8160 part features 24 cores.

[1]https://www.tacc.utexas.edu/systems/stampede2

[2]https://www.bsc.es/marenostrum/marenostrum

[3]256-Node Xeon Phi 7250 Intel Research cluster

V. Experimental results

A. Scaling ResNet-50 training up to 12K batch size

In order to keep our results measurable with similar ones in terms of hardware capabilities, we scale the training procedure of ResNet-50 up to 256 Intel KNL and Intel SKX nodes. Analogously, we also employ a minibatch size of 32 per worker. However, it is clear that scaling to a number of separate nodes greater by four to eight times puts greater pressure on the communication interconnect. In Section V-C we do not limit to 256 nodes, and scale ResNet-50 training all the way up to 1536 KNL nodes as well as on up to 512 SKX nodes. All our scaling experiments are performed on production systems such as TACC's Stampede Intel KNL-based supercomputer as well as BSC's MareNostrum 4 Intel SKX-based supercomputer.

The duration of a training epoch is a function of the local batch size and number and types of workers participating in the run. In our fastest scenario we can go through the 1.28 million training images from ILSVRC2012 in around 16.8 seconds while using 1536 KNL nodes, each working with a local batch size of 32 examples. This is the largest batch size, 49152, we have tested so far on a production system. However, for this very large-batch scenario needs around 100 epochs to reach a reasonable accuracy of 74.6%. The full training run is performed in 28 minutes, as will be described in Section V-C. This is significantly faster than the 33.3 seconds from [11], and 40 seconds from [10] for processing one epoch on ImageNet-1K using the ResNet-50 architecture. You *et al.* [12] can process one epoch in 20.66 seconds in the latest version of their paper, while using 1600 Intel SKX nodes.

1) Stampede2 KNL partition results: Stampede2 Xeon Phi nodes features 96 GB RAM and 16GB HBM configured in cache-mode. Since the compressed ImageNet-1K LMDB file size is 42GB, it can be copied to RAM at the beginning of the run, clearly improving execution efficiency. This also reduces any additional issues with the filesystem and networking.

We discovered empirically that after 5 warm-up epochs, and around 24 training epochs following the linear learning rate decay policy as described in Section IV-C, a ResNet-50 model can achieve around 73% accuracy. After around 50 epochs of training, the model achieves 74%. At the other extreme, after 90 epochs of training, the models achieve 75-75.8% top-1 validation accuracy.

Table II presents the validation accuracy results using single-crop evaluation of ResNet-50 models, as well as the total time-to-train required to achieve it. In order to keep the global batch size around the 12000 examples mark, we use a local batch 16 size for 720 nodes and 24 for 612, while the 256 node case is our 32 batch size baseline. Performing a full 90 epoch schedule leads to a SOTA top-1 validation error on the full ILSVRC2012 validation set.

Table II: ResNet-50 SOTA on Stampede2 KNL partition

Batch size	# nodes	# epochs	top-1/top-5 accuracy	TTT[min]
8192	256	90	75.81/92.93	140
12288	512	90	75.25/92.90	80
11520	720	90	75.03/92.69	62

Table III: Stampede2 KNL partition fastest-time-to-74%

Batch size	# nodes	# epochs	top-1/top-5 accuracy	TTT[min]
4096	256	37	74.05/92.12	63
8192	256	50	74.12/92.16	70
8192	512	50	74.12/92.16	49
12288	512	54	74.05/92.07	46
12288	768	61	74.20/92.20	39

Table IV: ResNet-50 on Stampede2 KNL partition

Batch size	# nodes	# epochs	top-1/top-5 accuracy	TTT[min]
2048	256	29	72.43/91.45	64
4096	256	37	74.05/92.12	63
8192	256	29	73.21/91.58	41.5
8192	256	37	73.92/91.83	52
8192	512	37	73.78/91.78	37

One of our goals is to decrease the time to train for a good quality model. In this case we set a target accuracy 74% top-1, and reduced the number of epochs in order to reach it. In Table III we present results that were obtained using a various number of training epochs, batch sizes, and number of nodes, all reaching above 74% validation accuracy. They show convergence for the 512-node case within 46 minutes, and for the 768-node case in only 39 minutes. The local minibatch size was set to 16. Larger global batch size techniques are discussed in Section V-C. It is interesting to observe that in the case of the 256-node run using a local minibatch size of 16 (first row from Table III), we can achieve the target validation accuracy by training for only 37 epochs, in 63 minutes. This result is comparable to the other research results in terms of time-to-train and compute performance.

Going even further, Table IV indicates that validation accuracy can be controlled with the number of training epochs and batch size. For example, sacrificing 0.13% final top-1 accuracy shortens the training run 11 minutes, which is 17% less than the time needed to obtain 74% accuracy.

When taking scaling efficiency into consideration, our approach based on Intel Caffe and Intel ML-SL can achieve around 97% when going from 1 to 256 nodes, higher compared to similar research [10] and [11]. This is supported by Figure 1. A detail to consider here is that after each forward pass, each worker node needs to send and receive a complete model, which is around 98MB for ResNet-50. Thus, in our case, after each iteration 256 worker nodes send and receive 98MB, leading to 49GB of traffic. In the case of [11] this pressure is only 12GB, while for [10] case it is only 6GB, due to the optimised intra-node allreduce achieved through NVIDIA's NCCL library.

Local batch size	# nodes	time/epoch
16	256	102 sec
32	256	85 sec
16	512	60 sec
20	512	55 sec
24	512	51.3 sec
32	512	48.8 sec
16	768	38 sec
16	1024	31.5 sec
32	1024	24.5 sec
32	1536	16.8 sec

Table V: Time per epoch when using KNL nodes in Stampede2

Local batch size	# nodes	time/epoch
32	256	84.3 sec
24	400	62.5 sec
16	400	67.2 sec
16	512	52.5 sec

Table VI: Time per epoch when using MareNostrum 4

In order to see how well this efficient scaling holds, we decided to verify some of the strong scaling properties and push the training of ResNet-50 by using 512, 768, and 1024 Xeon Phi node runs with a batch size of 16 images per node. This leads, of course, to a global batch size of 16384 for the 1024-node case, making the usual overlap of communication and computation harder to achieve than in the case of local batch size 32, as computation takes less time and the gradient transfers are performed more frequently. The weak scaling efficiency for a local batch size of 16 is 81% when going from 256 to 1024 nodes. When using a local batch size of 32, the weak scaling efficiency is 88% when going from 256 to 1024 workers. Table V outlines the number of seconds required for training one full epoch of ImageNet-1K under the various batch sizes and number of nodes. Each value is calculated as an average over a full training run.

The techniques developed subsequently and described in Section V-B and Section VI are not used in any of the weak and strong scaling results presented in this subsection.

However, for completion, we mention here also the 1536 node result, further described in Section V-B. It has a scaling efficiency of above 81%. Interestingly, in this scenario, after just 2600 updates - 100 epochs of SSGD, the top-1 accuracy is already at 74.6%.

2) BSC MareNostrum 4 results: Besides the above experiments on Stampede2, we have also performed similar ones on the Intel Xeon SKX nodes from Barcelona Supercomputing Center's (BSC) MareNostrum 4 supercomputer.

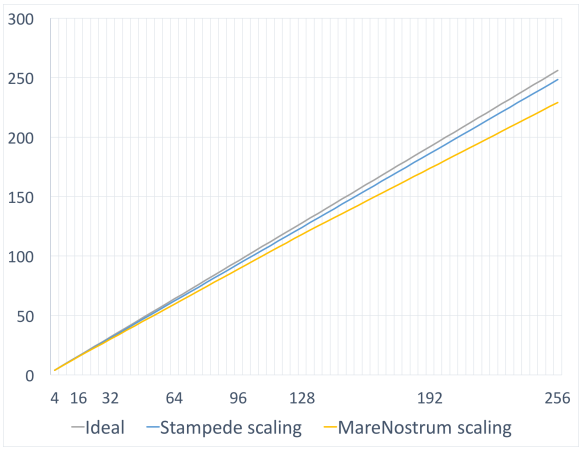


Figure 1: Scaling efficiency on Stampede2 (speedup vs number of workers) and MareNostrum 4

Table VII: MareNostrum 4 fastest-time-to-74%

Batch size	# nodes	# epochs	top-1/top-5 accuracy	TTT[min]
8192	256	50	74.11/92.14	70
9600	400	60	74.29/92.29	62.5
6400	400	50	73.94/92.03	56
8192	512	64	74.15/92.25	56
8192	512	50	74.01/92.03	44

MareNostrum 4 features around 3200 2S Intel Xeon Processor 8160 (SKX) nodes, each with 96GB RAM and around 200GB of local storage.

The scaling efficiency observed from 1 to 256 SKX nodes is 90%, as can be seen in Figure 1. Larger runs, of 256+ nodes, together with their validation accuracies and achieved time-to-train are presented in Table VII. As in the Stampede2 case, we set the target accuracy for all our trained models to be greater than 74% on the ILSVRC2012 validation set. Interestingly, for strong scaling purposes the Skylake architecture appears to be more appropriate. This is suggested by the fact that the 512-node, 16 batch size per node experiment performed quicker than on the KNL-based Stampede2 system using a similar number of nodes. Due to our limited compute budget we could not perform larger scale experiments, like 512+ SKX node runs, on MareNostrum 4. We also did not perform full 90 epoch experiments on MareNostrum 4, but the conclusions from Stampede2 large-batch experiments are architecture agnostic in terms of final accuracy. Table VI gives the average time per epoch to perform ResNet-50 training using the SKX architecture under various configurations. Using 74% as our top-1 performance target, some of the highlights of our results are achieving convergence in 70, 56, and 44 minutes, when training on 256, 400, or 512 nodes respectively.

B. Improving the baseline accuracy

When performing various scaling experiments, we observed that the validation accuracy degrades after exceeding a batch size of 8K. We discuss here how to overcome this behaviour and obtain SOTA accuracy with batch sizes of up to 64K when using a ResNet-50 architecture.

Table VIII: Impact of techniques to improve baseline accuracy at large batch sizes

Batch size	weight decay (λ)	collapse	top-1 accuracy
4096	0.0001	no	75.8
4096	0.00005	no	76.22
4096	0.00005+0.0001	yes	76.65

Table IX: Very large batch training results

Batch size	# nodes	# epochs	top-1/top-5 accuracy	TTT[min]
10240	512	91	76.40/93.16	82
16384	512	92	76.26/93.19	74
32768	1024	100	75.31/92.70	42
49152	1536	100	74.6/92.1	**28**

1) Smaller weight decay: The weight decay hyperparameter has a big impact on the optimization, particularly in the first phase of training, when the learning rates tend to be large. Adjusting this to a smaller value of half of the default value, 0.00005, improves top-1 accuracy from 75.8% to 76.22%.

2) Final collapse: Another empiric observation is that the weight decay needs to actually be dynamically adjusted. We train the first and largest chunk of the training with the lower weight decay and still with a linear learning rate decay. Afterwards, the last 5-7% of the training is performed in a collapsed fashion. The learning rate is decayed with a power of 2 polynomial in this phase (starting from the value where it left off), the weight decay doubled to 0.0001, and scale/aspect ratio augmentation is disabled. This regime is typically performed for 4-7 epochs, allowing us to increase the top-1 accuracy to 76.65% for various batch sizes (up to 16K). Some experiments using these improvements are presented in Table VIII. They are performed on an internal cluster using 128 KNL nodes and a batch size of 4096. These two techniques hold to much larger batch sizes, as is presented in the following section.

C. Scaling to extremely large batch sizes

Increasing the batch size to 64k requires an additional adjustment in order to keep the accuracy close to SOTA. Instead of scaling the learning rate linearly, we keep it at a value of 6.4 for all runs with batch size 16K and above and then decrease it in the same linear way. Also, the warm-up period is typically longer, at around 7-10 epochs. Even in this cae accuracy degrades as the batch size increases over a certain limit.

We achieve 74.6% accuracy using a batch size of 49152 distributed over 1536 KNL nodes in just 28 minutes. We also achieve state-of-the-art 75.3% accuracy at a batch size of 32K. The large batch results are present in Table IX.

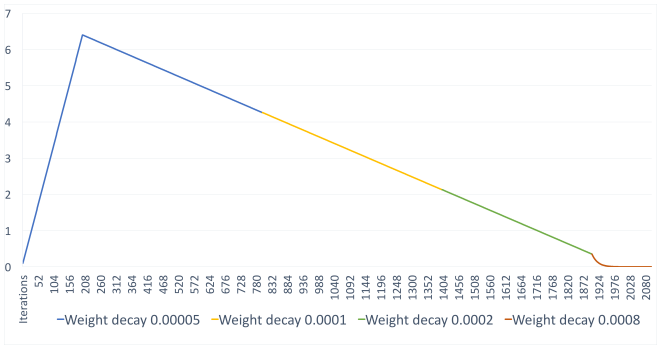


Figure 2: Plot of learning rate behaviour when performing a final collapse and coloured for different weight decays for 64k batch size

Table X: Comparative accuracy of large batch training. Original ResNet-50 accuracy is 75.3% top-1

Batch size	8K	16K	32K	48K	64K
Cho *et al.* [11]	75%	-	-	-	-
Goyal *et al.* [10]	76.2%	75.2%	72.4%	-	66.04%
You *et al.* [12]	75.3%	75.3%	74.7%	-	72%
This work	**76.6%**	**76.26%**	**75.31%**	74.6%	**73.94%**

We have also performed an experiment with a batch size of 64K, using 128 KNL nodes from an internal infrastructure, each with a local batch size of 512. The model achieved a top-1 accuracy of 73.94% in only 2100 SGD updates, sacrificing just 1.3% top-1 accuracy compared to the original implementation [7]. To reach this level of accuracy with relatively few iterations, we adjusted the learning rate scheduling even further. This new schedule is presented in Figure 2 and explores multi-step weight decay increase during training.

Table X compares our results to the most relevant approaches from the literature that consider very large batch training, and show that we exceed the state-of-the-art accuracy for every batch size, while also maintaining very good hardware and scaling efficiency.

VI. Collapsed ensembles

In an attempt to improve the final accuracy, we decided to combine multiple collapsed snapshots obtained during the training run by using an ensemble average over them, while using a constant training budget. By using the LR schedule from Figure 3 we can achieve good any-time performance from a total 120-epoch training run. We chose a semi-cyclic learning rate schedule in a fashion similar to [23]. The training is composed of multiple cycles. The duration and number of cycles controls the accuracy of the resulting model. We have chosen 5 cycles starting at epoch 45. Thus, the first part of the training, until around epoch 30 we linearly decay the learning rate.

$$\alpha_{t_x} = \frac{(\alpha_{t_n} - \alpha_{t_{x-1}})}{(t_n - t_{x-1})} + \alpha_{t_{x-1}}$$

We then increase the decay slope by employing a polynomial decay.

$$\alpha_{t_x} = \frac{(\alpha_{t_n} - \alpha_{t_{x-1}})}{\sqrt{2 * (t_n - t_{x-1})}} + \alpha_{t_{x-1}}$$

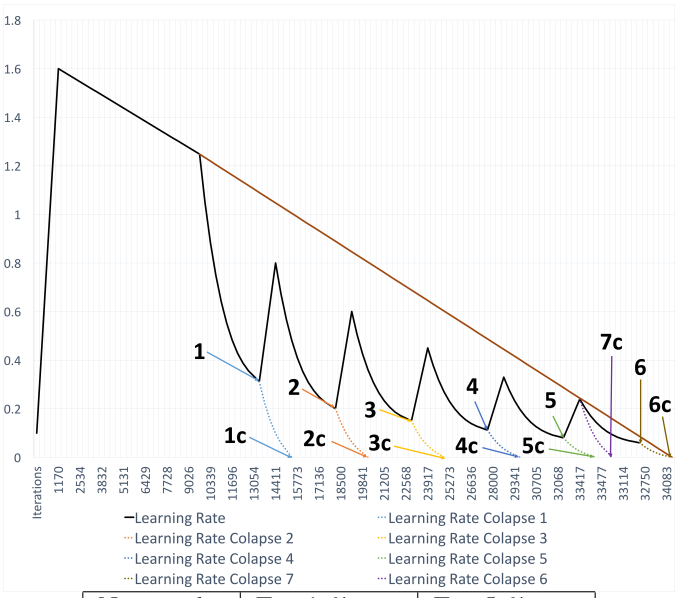
This is done for another 15 epochs, and at this point, after 45 epochs the learning rate decays to around 22% of its original value. In order to prevent model overfitting, starting from this point we use a cyclic learning rate schedule, where we increase the learning rate by a factor of 3 in a linear fashion over around 3 epochs (similar to a re-warmup for 3 epochs), and decay it again with a power of two for the following 12 epochs (The decay is computed such that after 12 epochs, the learning rate is divided by a factor of 4). This results in the first cycle finishing at epoch 60. The same follows for cycles at epoch 75, 90, 105, and 120. An important observation is that this is a hand tuned, empirically derived schedule, in future work we will analyse this behaviour further and encapsulate it in an analytic form. Based on the fact that the final 3 collapses show relatively similar performance, it seems that the amplitude of the cycles needs to be adjusted in order to produce snapshots that converge to different minima.

Figure 3 shows that already after around 60 epochs, the performance level is above 76% top-1 accuracy. This grows to a 76.5% after 75 epochs.

We call this "Collapsed Ensembles"(CE). Similar to the snapshot ensemble technique introduced by Huang *et al.* [24], we reuse the same training budget to create an ensemble of snapshots. For the ImageNet dataset we create a majority voting ensemble from 5 models, achieving a ResNet-50 record 77.5% accuracy, totalling a single standard 120 epoch training run. This schedule is applied for a batch size of 4096, but the same holds for batch sizes of up to 16K.

Regular large-batch (as well as small-batch) ResNet training uses a weight decay of 0.0001, momentum of 0.9 and linear scaling of the learning rate to accommodate the potentially larger batches. We have empirically noticed that particularly for the large-batch regime it is important to start the training run with a smaller weight decay, and then increase it towards the end of the run, in the "collapsing phase". All "collapse" experiments start with a weight decay of 0.00005, thus half of the standard one, and use scale and aspect ratio augmentation. In the "collapse" phase, we disable data augmentation (while still keeping the random crop), and we double the weight decay to its standard value of 0.0001. In this collapse phase we typically perform between 3-10 epochs, with the larger collapses offering greater accuracy benefits. As described in Section V-C, gradually increasing the weight decay multiple times during training is advised, particularly for very large batches.

By using the time-per-epoch computed in Table V and



No. on plot	Top-1 % acc.	Top-5 % acc.
1	68.33	88.71
1c	75.50	92.83
2	71.54	90.78
2c	76.15	93.17
3	73.28	91.58
3c	76.50	93.24
4	73.31	91.53
4c	76.57	93.24
5	73.89	91.97
5c	76.83	93.32
6	74.49	92.13
6c	76.81	93.32
7c	76.70	93.32

Figure 3: Plot of learning rate behaviour when obtaining the ensemble snapshots

VI, coupled with the accuracy results obtained by the cyclic collapses, we can project a time to a given top-1 accuracy on Stampede2. Note that for these projections we estimate the need of 48 epochs to reach 75.5%, 64 epochs to reach 76%, and 78 epochs to reach 76.5% as demonstrated by the learning rate schedule from Figure 3 . Table XI outlines this projection.

VII. Making efficient usage of NUMA systems

While performing the experiments in the previous sections, we noticed limited intra-node scaling behaviour when increasing the number of compute threads. This effect is more pronounced on the SKX nodes, where to overcome it, we start one process on each SKX CPU, each process using 22 cores through OpenMP, with 4 cores reserved for communication as described in the methodology [25]. The 44 threads are physically executed on two separate NUMA

Table XI: Projected time required to reach a given top-1 accuracy using CE

# nodes	type	batch size	TT-75.5%	TT-76%	TT-76.5%
256	KNL	4096	82 min	109 min	133 min
256	KNL	8192	68 min	90 min	110 min
512	KNL	8192	48 min	64 min	78 min
512	KNL	10240	44 min	58 min	72 min
512	KNL	12288	41 min	55 min	67 min
768	KNL	12288	31 min	41 min	49 min
256	SKX	8192	68 min	90 min	110 min
400	SKX	6400	54 min	72 min	87 min
400	SKX	9600	50 min	67 min	81 min
512	SKX	8192	42 min	56 min	68 min

Table XII: Efficient training using SKX nodes in Stampede2

# nodes	90-epoch runtime	Average throughput	Time/epoch
256	101 min	19009 img/s	67.3 sec
400	66 min	20480 img/s	44 sec
448	58 min	33160 img/s	38.6 sec

domains. This proved very beneficial also when scaling out, obtaining a speed-up of around 25% for the 256-node case compared to the results we presented earlier in Table VI . By making more efficient use of hardware, we performed a final set of experiments using 256, 400, and 448 nodes respectively. All these runs have a batch size of 32, divided evenly over the two sockets, use a 90 epochs, and reach above 76% top-1 accuracy. The timing details are provided in Table XII.

Finally, in order to assess both the scaling performance as well as single-device hardware efficiency, we take this 448 node training run and compare it to the similarly sized training runs presented in [10] and [12], that achieve an accuracy in excess of Resnet-50's SOTA in 90 epochs, in below one hour. Table XIII aims to compare these runs using a metric that captures both hardware efficiency and scaling efficiency (HW eff. in the table), representing the average throughput divided by the system peak performance, measured in Images/s/TFLOPS. The "Peak" column in the table refers to the peak performance of either the SKX CPUs or the P100 GPUs, multiplied by the number of devices used for the training run (in FP32 TFLOPS). In the case of [10], we have only counted the performance of the GPUs, although the CPU hosts also contribute to the training.

It is interesting to note that using comparable hardware (896 SKX CPUs vs. 256 P100 GPUs), the time-to-train is similar (58 minutes vs. 60 minutes), and the hardware efficiency is slightly higher for the CPU case.

VIII. Conclusion

Using a combination of techniques (very large global batch sizes, modified batch normalization, aggressive learning rate schedules, warm-up strategies, weight decay improvements, collapsed ensembles), we have achieved a scalable solution based on Intel's distribution of Caffe that

Table XIII: Comparing the efficiency of CPU and GPU systems for Resnet-50 training. The GPU system peak performance does not include the CPU hosts.

HW type	nodes/devices	Peak FP32	TTT	HW eff.
[1] SKX 8160	448/896	2682 TF	58 min	12.36
[2] NVIDIA P100	32/256	2658 TF	60 min	12.03
[3] SKX 8160	1024/2048	6144 TF	48 min	6.51

[1] This work [2] Goyal *et al.* [3] You *et al.*

provides state-of-the-art neural network training both in terms of time to trained model and in terms of the achieved accuracy. We have performed several trade-offs between the training time and the accuracy of the resulting model. We have scaled up ResNet-50 training to up to 1536 KNL nodes working with a global batch size of 49152, achieving state-of-the-art accuracy in only 28 minutes, while demonstrating a scaling efficiency of over 80%. We have also experimented with the Intel Skylake CPUs, and acknowledge it as a viable architecture for training deep neural networks, especially when strong scaling is involved, as the Skylake CPUs sustain small local batches more efficiently. Moreover, by making efficient use of the separate NUMA domains, we can achieve slightly better hardware efficiency compared to a similar scale NVIDIA P100 system.

By using the collapsed ensemble learning rate schedule, we achieved a single-model accuracy of 76.5% after 78 epochs and an ensemble result of 77.5%. Although similar to Huang *et al.*, our technique performs better on the ImageNet dataset when using a 5 model majority voting with the same training budget.

In the future, we are planning to investigate other learning rate strategies, such the cosine learning rates also presented in [26]. Based on our own empirical results and inspired by research such as the Cyclic learning rate [23] and SGDR [26], we believe that as far as SGD based methods are concerned, there is a correlation between a measure of regularity of a dataset, learning rate policy, and maximum global batch size. We plan to investigate a way to automatically learn this relation. More than this, we want to analyze the relationship between width and depth of convolutional networks, total number of inputs, total number of classes, and number of examples per class. More specifically, we will aim at further improving accuracy when using very large batches, but also we plan to use larger-scale datasets, such as the full ImageNet.

ACKNOWLEDGMENT

We would like to thank PRACE for awarding access to the MareNostrum 4 system based in Spain at BSC. We would also want to thank all the consultants from TACC, BSC, and from the VLAB cluster for promptly helping us whith various issues on the mentioned systems. We want to acknowledge our collaboration with Intel under the IPCC framework, and thank Intel for providing access to Stampede2 and to the VLAB cluster.

REFERENCES

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

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

[3] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *arXiv preprint arXiv:1709.01507*, 2017.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, 2015.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[6] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," *arXiv preprint arXiv:1707.02968*, vol. 1, 2017.

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

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

[11] M. Cho, U. Finkler, S. Kumar, D. Kung, V. Saxena, and D. Sreedhar, "Powerai ddl," *arXiv preprint arXiv:1708.02188*, 2017.

[12] Y. You, Z. Zhang, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Imagenet training in 24 minutes," *arXiv preprint arXiv:1709.05011*, 2017.

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

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

[15] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," *arXiv preprint arXiv:1705.08741*, 2017.

[16] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets," *arXiv preprint arXiv:1703.04933*, 2017.

[17] T. Kurth, J. Zhang, N. Satish, E. Racah, I. Mitliagkas, M. M. A. Patwary, T. Malas, N. Sundaram, W. Bhimji, M. Smorkalov *et al.*, "Deep learning at 15pf: supervised and semi-supervised classification for scientific data," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 7.

[18] S. L. Smith, P.-J. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[19] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, 2017, pp. 4278–4284.

[20] Intel, "Intel distribution of Caffe," https://github.com/intel/caffe, 2017.

[21] ——, "Intel Machine Learning Scaling Library," https://github.com/01org/MLSL, 2017.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[23] L. N. Smith and N. Topin, "Exploring loss function topology with cyclical learning rates," *arXiv preprint arXiv:1702.04283*, 2017.

[24] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get m for free," *arXiv preprint arXiv:1704.00109*, 2017.

[25] V. Saletore, D. Karkada, V. Sripathi, K. Datta, and A. Sankaranarayanan, "Boosting Deep Learning Training & Inference Performance on Intel® Xeon® and Xeon Phi™ Processor Based Platforms: Performance and Methodology," *Technical Report*, 2017.

[26] I. Loshchilov and F. Hutter, "Sgdr: stochastic gradient descent with restarts," *arXiv preprint arXiv:1608.03983*, 2016.