Optimizing neon Deep Learning Framework for Intel Architectures

Wei V. Wang, Peng A. Zhang, Jayaram Bobba, Dawn Stone, Menglin Wu, Xiaohui Zhao, Mingfei Ma, Wenting Jiang, Jason Y. Ye, Huma Abidi, Jennifer Myers, Hanlin Tang, Evren Tumer
Intel Corporation
Santa Clara, CA
firstname.[middlename.]lastname@intel.com

ABSTRACT

neon is a deep learning framework with industry leading performance on GPUs. However, its performance on CPU platforms is not as compelling. This paper presents the acceleration of neon framework on Intel architectures using Intel® Math Kernel Library (Intel MKL). The optimized implementation provides up to 98× speedup on popular benchmarks and topologies. For example, GoogLeNet v1 inference throughput is 956 images/second on the newest Intel Xeon Skylake platform, enabling high throughput inference with neon on CPUs. neon also demonstrates competitive CPU performance on topologies such as ResNet-50 (training throughput at 63 images/sec on a Skylake system). These optimizations will allow data scientists and machine learning researchers to begin leveraging readily available CPUs to develop deep learning models for both training and inference with neon.

1 INTRODUCTION

Deep Learning (DL) is gaining popularity from a wide range of application scenarios including image classification [18], object detection [22], and speech translation [25] etc. DL is becoming the core technology driving the development of various Artificial Intelligence (AI) fields like autonomous driving. Both the hardware and software should innovate accordingly to catch and support the rising of Deep Learning. From the hardware side, chips optimized for DL workloads like Google Tensor Processing Units (TPU) [14] are continuing to roll out. In the meantime, given the dominance of the Intel Xeon CPU architectures, there will be a great impact coming from software improvements on these systems. Deep Learning frameworks like `Caffe` [13] and `TensorFlow` [1] have already been well optimized to run much better on Intel architectures [15, 21]. This paper adds a third popular framework neon to the list of frameworks optimized for Intel architectures.

neon is a deep learning framework created by Nervana Systems [1] with industry leading performance on GPUs thanks to its custom assembly kernels and optimized algorithms [20]. Although neon supports Intel architectures by default with its "CPU" backend, its performance is far from being optimized. To bring superior performance to Intel CPU platforms, we relied on Intel MKLML library (the Deep Neural Network component [5] of the Intel Math Kernel Library [6], or MKL-DNN for short in this paper) to extract the performance out of the different generations of Intel architectures. Intel MKLML library provides CPU optimized implementations for widely used primitives like convolution, pooling, activation functions, and normalization etc. These MKL primitives exploit the full vectorization and parallelization capabilities of Intel architectures in contrast to existing vanilla implementations. We integrated the MKLML library into neon by replacing the original neural network layers with their MKL-DNN counterparts. Doing so significantly increased the performance of poplular Deep Learning topologies like Alexnet, Googlenet, and Resnet-50. As shown in the results section, the optimized neon implementation delivers upto 98× better training throughput on a Broadwell system. The performance of neon on a Skylake system is another 1.5× to 2× better.

The rest of the paper is organized as follows. Section 2 describes the details of the optimization of neon. Section 3 shows the machine setup we used to evaluate our optimization efforts. The performance of the optimized neon framework is presented in Section 4. Section 5 discusses the related work. Section 6 concludes the paper.

2 OPTIMIZATIONS OF NEON USING INTEL MKL

neon originally has two backends: the cpu backend (NervanaCPU) and the GPU backend (NervanaGPU). Before our optimization, the cpu backend by default features NumPy [23] support when code executes on Intel architectures. In this optimization work, e have developed a new neon backend (NervanaMKL) that utilizes MKL primitives where available. The following neon ops are currently optimized with MKL: 2D direct convolution, Pooling, ReLU, BatchNorm, MergeSum, and MergeBroadcast. To achieve peak performance, MKL primitives require N-dimensional input data to be laid out in specific SIMD-friendly formats. To reduce the burden on neon users, we have incorporated the plumbing required for automatic data layout tracking and conversion into the NervanaMKL backend. We have also rewritten element-wise operations using OpenMP to speed up execution. Together all these optimizations provide a significant performance boost for both training and inference tasks. We have validated the correctness of the implementation on a variety of models that are provided along with the neon framework. Performance has been optimized for various ImageNet-based models like AlexNet, GoogLeNet-v1, and ResNet. The optimized MKL backend is enabled by default. The latest MKL DNN release was used and it featured with optimizations for the new Intel Xeon

[1]now Intel AI Products Group

AIM'17, Sep 2017, Portland, Oregon USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: MKL-optimized neural network layer operations and their occurrences in popular models like AlexNet, GoogleNet, and ResNet. The C files are located under neon/backend/mklEngine/src directory of neon repository.

Operation	C file	Model
Convolution	conv.c	AlexNet, GoogleNet, ResNet
Deconvolution	conv.c	AlexNet, GoogleNet, ResNet
Pooling	pooling.c	AlexNet, GoogleNet, ResNet
Relu	relu.c	AlexNet, GoogleNet, ResNet
BatchNorm	batchNorm.c	ResNet
MergeSum	concat.c	ResNet
MergeBroadCast	concat.c	GoogleNet

platform (code name Skylake) and the upcoming Xeon Phi platform (code name Knights Mill). The DNN component of MKL is provided free of charge and downloaded automatically as part of the neon installation. We encourage readers checking out MKL-optimized neon [20] and try out various models on IA platforms.

2.1 Optimization Details

The modifications to neon were performed in a way that least affected the original functions while speeding up the execution on Intel CPUs. We focused on improving the performance of convnet benchmarks and ResNet-50 with real images on Intel Xeon architectures (both Broadwell and Skylake servers). Other models may also benefit from the optimizations. The following are key tasks to realize the optimization:

(1) Develop Neural Network (NN) layer operations based on MKL DNN primitives and link them into a dynamic C library
(2) Design MKL backend as Python interface for the dynamic C library
(3) Make necessary tensor/data layout modifications and remove unnecessary tensor layout conversions
(4) Enable merged execution of models with both the CPU and MKL operations (Ops)

Below describes the detailed implementation for the above tasks.

2.1.1 Neural Network Layer Operations based on MKL Primitives. All the important NN Ops for AlexNet, GoogleNet, and ResNet models are implemented based on MKL primitives, as shown in Table 1. Their MKL-based implementations were put into several C files and altogether these C files were compiled and linked to form a dynamic library called libmklEngine.so.

2.1.2 Python Interfaces for MKL Operations. The vanilla neon code is mostly python based. To be able to invoke MKL DNN operations in the C library, we implemented the MKL backend as the python interface. In this MKL backend, we used ctypes.cdll to load the libmklEngine.so C library. After the library is loaded, calls to the MKL ops are then invoked in the python file. As shown in Figure 1, when the operation is supported by MKL, the python interface (e.g. forward propagation of pooling operation in nervanamkl.py located under neon/backend/ directory) is the entry point of the execution. Inside the python interface file for the fprop_pooling op (nervanamkl.py), the python interface function first prepares the

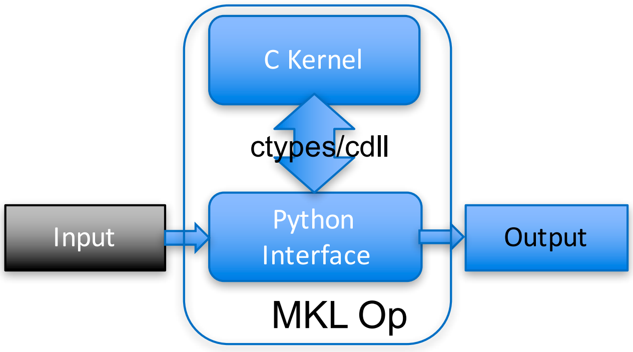


Figure 1: An example workflow for an MKL operation with python interface.

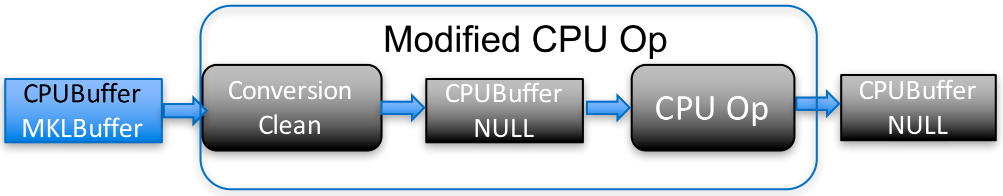
input data and then invokes the MKL API (e.g. MaxPooling_fprop) to execute the C library code. After the execution of MKL operations, data are normally output. However, not all NN Ops are implemented to support calling MKL primitives in the backend. Currently we check the type of the layer operation's backend to determine whether to go into MKL op or the original CPU Ops.

2.1.3 Tensor Layout Modifications. In order to support the flow of tensors with MKL-enabled op, a new tensor class was designed to inherit from the original CPU tensor. The new tensor class carries CPU and MKL buffer pointers as well as their layout information. During runtime, this new tensor can act like both MKL and CPU tensors. A CPU tensor only has the NUMPY buffer and it does not have the MKL buffer. A MKL tensor has both CPU and MKL buffer, although its CPU buffer is not used. MKL Ops seamlessly work on MKL buffer while CPU Ops work on NUMPY buffer. Besides, MKL Ops requires shape information of tensors, which is absent in the original tensor class (which only has two dimensions and lacks CHW dimension information, i.e. the number of channels, the heights and width information of images). Although most layers obtain shape information from the input and output tensor, some of the operations like Relu Op dose not. We added shape member in the MKLTensor to record NCHW information. Since tensor flows between ops and ops on both ends can be either MKL enabled ops or the original CPU op, explicit/implicit conversion/initialization are required, when the two operations are on different backends.

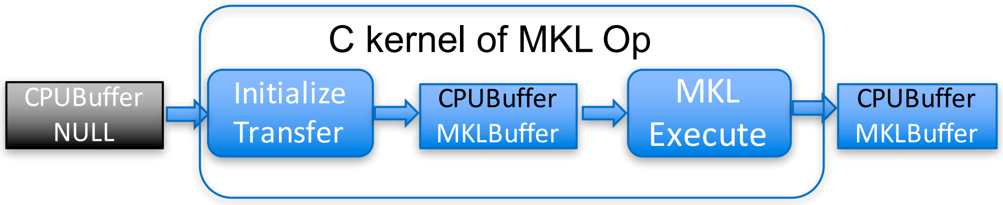
There are four cases:

(1) CPU Ops are fed with non-MKL (CPU) Tensors. Nothing is different and there is nothing to do.
(2) CPU Ops are fed with MKL Tensors. An explicit conversion/clean step is necessary. Take Bias op which is not MKL-optimized for example:

```
def bprop(self, error):
#transfer MKL buffer to CPU buffer, if has MKL buffer
self.be.convert(error)
#make MKL buffer NULL to become a non-MKL buffer
error.cleanMKL()
...
#make deltas MKL buffer NULL
self.deltas.cleanMKL()
```



(a) MKL tensors explicitly converted to CPU tensors when fed to CPU operations



(b) CPU tensors implicitly transferred to be MKL tensors when fed to MKL operations

Figure 2: Required tensor conversions between MKL-optimized and non-optimized operations

Function convert() will transfer MKL content to the NUMPY buffer if MKL buffer exists. Function cleanMKL() will make MKL buffer NULL. Figure 2a illustrates the process of converting MKL tensors to CPU tensors when they are fed to CPU operations that do not support MKL. All the Ops required by the model but unsupported should be modified. There are a number of unoptimized Ops that have to perform these conversions. As of now, in neon we have covered such operations as Bias, LRN, Dropout, Linear, SkipNode, Softmax.

(3) MKL Ops are fed with MKL Tensors. This is also a simple case where the NUMPY buffer is ignored and the MKL buffer is processed as input.
(4) MKL Ops are fed with non-MKL (CPU) Tensor. In this case, an implicit initialization and conversion step is implemented in the C code (transparent to the python code). The contents in the NUMPY buffer is converted and copied into MKL buffer before MKL ops start execution. Figure 2b shows the process of implicitly converting CPU tensor into MKL formatted tensor in the C library code.

2.1.4 Elementwise Ops and Correctness Validation. To improve performance, we also rewrote element-wise Ops in math_cpu.c and implemented the corresponding python interface in nervanamkl.py and math_cpu.py. These element wise operations (e.g. tensor broadcast and multiply) are parallelized by adding OpenMP pragmas. To help validate our optimization's correctness, we developed code that validated layer by layer outputs of the original CPU backend and the MKL-optimized backend. In addition, we also perform long-running convergence test to make sure that our optimized model converges to state-of-the-art training and inference accuracy.

2.2 Future Work

For the future releases of neon, we have more performance improvements in the pipeline. Currently, neon uses a unique "CHWN" data layout [7] for data which helped build fast GPU kernels. As noted earlier, CPU kernels prefer a different data layout which leads to data conversions between the layers of a model. We are working on both reducing the number of these conversions and on speeding up each conversion. In addition, we are also looking into improving RNN performance and also optimizing a wider variety of models. This will help us close the gap between neon and other Intel-optimized frameworks like Intel `Caffe` where it exists. In future neon will feature performance optimizations for a broader range of models including GANs and DeepSpeech2. neon will also feature Intel Nervana Graph support, enabling multinode training, new models such as ResNet-Inception, SSD, and a wide range of Reinforcement Learning models [2].

3 EXPERIMENTAL SETUP

Two systems with two generations of Intel architectures were used for evaluating the performance of the optimized neon framework: one with Broadwell CPUs and the other with Skylake CPUs. The Broadwell (BDW) system was a dual-socket 22-core (total 44 cores) Intel E5-2699 v4 CPU with a frequency of 2.2GHz. The BDW system had 500GB of DDR4 memory and ran Ubuntu 14.04. The Skylake (SKX) system had dual-socket 28-core (total 56 cores) Intel Xeon Platinum 8180 CPU, clocked at 2.50GHz. The CPU frequency scaling governor was set to "performance" via intel_pstate driver. The SKX system had 384GB of DDR4-2666 ECC memory. The operating system used was CentOS Linux release 7.3.1611 (Core) with Linux kernel 3.10.0-514.10.2.el7.x86_64. Input images were stored on an Intel DC S3700 Series SSD (800GB, 2.5in SATA 6Gb/s, 25nm, MLC). The performance was measured with KMP_AFFINITY environment variable set to "granularity=fine, compact" and OMP_NUM_THREADS to 56 for SKX and 44 for BDW. For both systems, the hyper-threading and turbo boost were disabled.

4 OPTIMIZATION RESULTS

This section reports the optimized performance of neon deep learning framework on Intel architectures with several popular deep

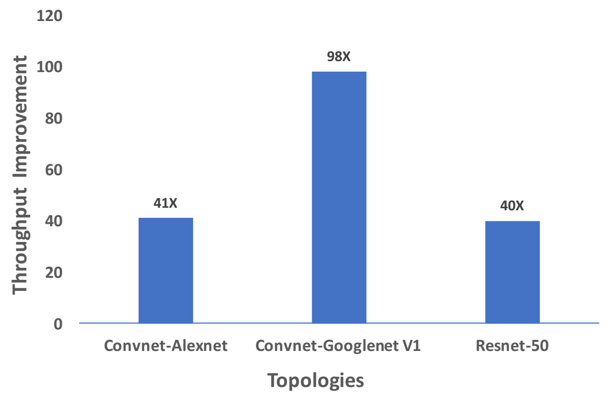


Figure 3: Training throughput improvement obtained on a dual-socket Intel Xeon E5-2699 v4 (Broadwell) system. Batch sizes of 256, 128, and 64 were used respectively for Convnet-AlexNet, Convnet-GoogleNet_V1, and ResNet-50.

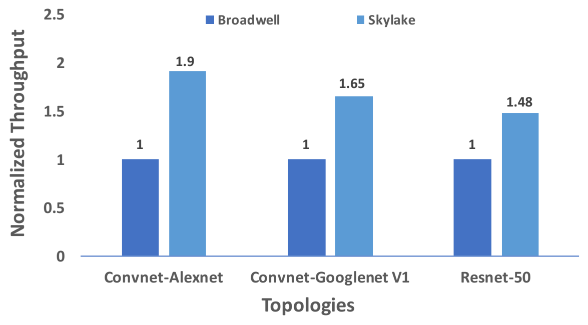


Figure 4: Training throughput of neon with the Broadwell microarchitecture system and the Skylake microarchitecture system (normalized to Broadwell performance). Batch sizes of 256, 128, and 64 were used respectively for Convnet-AlexNet, Convnet-GoogleNet_V1, and ResNet-50.

learning models. These optimization results will allow data scientists and machine learning researchers to leverage readily available CPUs to develop deep learning models for both training and inference.

4.1 Throughput Speedup from the Baseline Implementation

First we compare the throughput performance (in terms of images per second) of the MKL-enabled neon and the original NumPy based implementation. Figure 3 shows the significant training throughput improvement achieved on the dual-socket Broadwell system with popular topologies like Convnet-AlexNet, Convnet-GoogleNet_V1, and ResNet-50 (with imagenet dataset). The optimized implementation provides up to 98× speedup on these topologies. Neon demonstrates compelling CPU performance on topologies such as ResNet-50 (training throughput at 42 images/sec on the Broadwell systems). Inference throughput increase is similar to the improvement on training. For example, GoogleNet_V1 inference throughput is 539 images/sec on the Xeon platform, enabling high throughput inference with neon on CPUs.

4.2 Throughput Speedup from Broadwell to Skylake Microarchitecture

The Intel Xeon platinum processors based on the Skylake microarchitecture are some of the fastest microprocessors available in the market currently. In addition to delivering great general-purpose compute performance, they also have features like the AVX-512 instruction set that can significantly accelerate deep learning workloads. Neon and the associated MKL DNN library automatically uses AVX-512 instructions to accelerate ML kernels like convolution and inner product. In addition, they also leverage data layouts that are more suitable to the vector width (16 single-precision floats per vector) of the machine. As a result, as seen in Figure 4, we see up to about 2× improvement in workload performance compared to the previous generation Intel Xeon platform (based on Broadwell microarchitecture). For ResNet-50 training on imagenet dataset, the throughput has increased from 42 images/sec to 62 images/sec.

4.3 Comparison with `Caffe` Deep Learning Framework

`Caffe` is one of the several popular deep learning frameworks [13]. The Intel-optimized version of `Caffe` was introduced in [15] and the code is available from both the original BVLC `Caffe` website [24] (the "intel" branch) and the Intel maintained website [12]. For performance comparison, the same platinum Skylake featuring dual sockets (28 cores each) is used for running optimized neon and Intel-optimized `Caffe`. Note that GPU implementations of DL frameworks usually run with very limited batch sizes so that they do not run out of memory due to limited memory capacity even with the Pascal P100 GPU. For ResNet-50, we intentionally reduced the batch size from 64 to 50 for training ResNet-50 on Skylake so that others can compare GPU and CPU performance (i.e. GPU ResNet-50 data points will likely not be available using a large batch size like 64). Figure 5 shows the performance of neon and `Caffe` running AlexNet, GoogleNet, and ResNet-50 on Skylake using a batch size of 256, 128, and 50 respectively. For neon, the performance is competitive to that of `Caffe`. In the case of Convnet-GoogleNet_V1 training, neon achieved 98% the performance of `Caffe`. Thanks to the optimization work in neon, we have significantly closed the gap between the performance of some deep learning frameworks on Intel architectures and that of the latest generations of GPU architectures. More importantly, our optimized neon can now deliver on Skylake platforms competitive or even better performance than older GPUs like M40.

4.4 Discussion

So far our optimization has been performing very well on topologies like Alexnet, Googlenet, and Resnet. However, due to data conversion overhead mentioned in Section 2. When the MKL tensor flows from Convolution layer to Bias layer (which is typical in VGG model), data conversion needs to happen to convert MKL

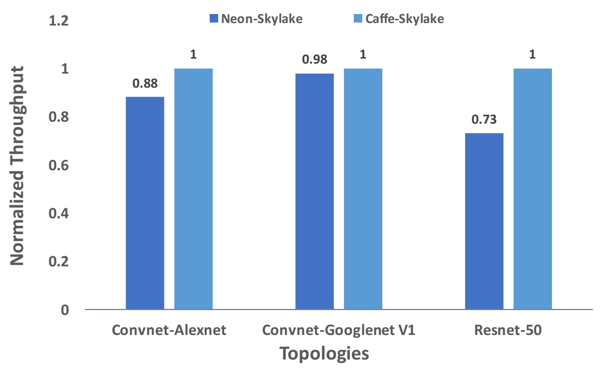


Figure 5: Training throughput of neon and Caffe on Intel Skylake Platinum 8180 platform (normalized to Caffe performance on Skylake). Batch sizes of 256, 128, and 50 were used respectively for Convnet-AlexNet, Convnet-GoogleNet_V1, and ResNet-50.

formatted data into CPU tensor format for continued execution in Bias layer, which has not been optimized by MKL library yet (i.e. the MKL library does not have a standalone Bias). However, the MKL DNN library indeed support fused convolution and bias. To avoid conversion overhead, our future work will also include optimizations that fuse the convolution layer and the bias layer to take advantage of the existing MKL APIs.

5 RELATED WORK

Deep Learning has become so critical that both the academia and the industry are dedicated to improving the performance of various deep learning frameworks. Almost every deep learning framework has a GPU implementation that runs on NVIDIA GPUs. CuDNN [10] was the main engine driving the GPU performance. On Intel architectures, Intel Math Kernel Library for Deep Neural Networks is driving the performance growth of almost all deep learning frameworks. To achieve the best IA performance, frameworks like Caffe, TensorFlow, Mxnet [16], Chainer [4], Theano [8] and Torch [9]. are all integrating the MKL DNN library. For TensorFlow, graph level optimizations were also performed to eliminate unnecessary and costly data layout conversions similar to neon [21]. In TensorFlow, graph nodes are modified to support both the original tensor and the MKL format Tensor. In neon, since there is no concept of graph, the MKLTensor comes with it a buffer for non-MKL optimized operations and a buffer for MKL-optimized operations. In TensorFlow, the fusion of convolution layer and the bias layer has already been implemented. However on the neon side, the bias layer has not been merged with convolution layer to save data conversion cost yet. NGraph (i.e. Intel Nervana Graph) [17] is a deep learning framework that supports neon and other frameworks as its front end. Since it features the graph node concept similar to that in TensorFlow, fusing of convolution and bias layer could be realized by fusing the graph nodes in NGraph. Aside from single node optimizations, the community has demonstrated advances in multi-node training with large batch sizes. Facebook researchers have demonstrated Caffe2-based ResNet-50 training on 256 GPUs in 1 hour with a batch size of 8192 [11]. Researchers from IBM presented Torch-based ResNet-50 training in 50 minutes [3]. Researchers from national labs as well as Intel labs have demonstrated deep learning at 15 PetaFlops using approximately 9600 Xeon-Phi nodes with Intel optimized Caffe [19]. The current limitation of our neon work is on multi-node. As of now, neon still lacks multi-node Intel architecture support. In future, this gap will be filled by NGraph.

6 CONCLUSION

The Intel Math Kernel Library for Deep Neural Networks has been well optimized for Intel architectures to execute deep learning workloads recently. By integrating MKL DNN library into neon, we were able to leverage the benefits offered by both software optimizations and newer hardware instruction-sets like AVX-512. This integration had enabled us to achieve improved training and inference performance by many orders of magnitude on Intel CPUs over the past year. For example, Googlenet now provides 927 images/sec inference throughput and 275 images/sec training throughput. This opens up neon deep learning model development to a wider range of users using existing general-purpose CPUs. By doing so we have also significantly closed the gap with specialized accelerators like GPUs.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[2] Jayaram Bobba. 2017. neon™ 2.0: Optimized for Intel® Architectures. (2017). https://www.intelnervana.com/neon-2-0-optimized-for-intel-architectures/

[3] Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. 2017. PowerAI: DDL. (2017). http://arxiv.org/abs/1708.02188

[4] Intel Corporation. 2017. Chainer optimized for Intel architectures. (2017). Retrieved September 1, 2017 from https://github.com/intel/chainer

[5] Intel Corporation. 2017. Intel(R) Math Kernel Library for Deep Neural Network. (2017). Retrieved September 1, 2017 from https://github.com/01org/mkl-dnn/releases

[6] Intel Corporation. 2017. Intel® Math Kernel Library. (2017). Retrieved September 1, 2017 from https://software.intel.com/en-us/mkl

[7] Intel Corporation. 2017. neon Design Choices. (2017).

[8] Intel Corporation. 2017. Theano optimized for Intel architectures. (2017). Retrieved September 1, 2017 from https://github.com/intel/Theano

[9] Intel Corporation. 2017. Torch optimized for Intel architectures. (2017). Retrieved September 1, 2017 from https://github.com/intel/torch

[10] NVIDIA Corporation. 2017. GPU Accelerated Deep Learning. (2017). Retrieved September 1, 2017 from https://developer.nvidia.com/cudnn

[11] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR* abs/1706.02677 (2017). http://arxiv.org/abs/1706.02677

[12] Intel. 2017. Intel Caffe: Dedicated to improving performance on Intel architectures. (2017). Retrieved September 1, 2017 from https://github.com/intel/caffe

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[14] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski,

Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, and Jonathan Ross. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. https://arxiv.org/pdf/1704.04760.pdf

[15] Vadim Karpusenko, Andres Rodriguez, Jacek Czaja, and Mariusz Moczala. 2016. `Caffe` Optimized for Intel® Architecture: Applying Modern Code Techniques. (2016). Retrieved September 1, 2017 from https://software.intel.com/en-us/articles/caffe-optimized-for-intel-architecture-applying-modern-code-techniques

[16] Young Jin Kim. 2017. Installing and Building MXNet with Intel(R) MKL. (2017). Retrieved September 1, 2017 from https://software.intel.com/en-us/articles/installing-and-building-mxnet-with-intel-mkl

[17] Jason Knight. 2017. Preview Release: Intel® Nervana^{TM} Graph. (2017).

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[19] Thorsten Kurth, Jian Zhang, Nadathur Satish, Ioannis Mitliagkas, Evan Racah, Mostofa Ali Patwary, Tareq Malas, Narayanan Sundaram, Wahid Bhimji, Mikhail Smorkalov, Jack Deslippe, Mikhail Shiryaev, Srinivas Sridharan, Prabhat, and Pradeep Dubey. 2017. Deep Learning at 15PF: Supervised and Semi-Supervised Classification for Scientific Data. (2017). https://arxiv.org/abs/1708.05256

[20] Intel Nervana. 2017. Intel® Nervana™ reference deep learning framework committed to best performance on all hardware. (2017). Retrieved September 1, 2017 from https://github.com/NervanaSystems/neon

[21] Elmoustapha Ould-Ahmed-Vall. 2017. `TensorFlow` Optimizations for Intel® Architecture. (2017). Retrieved September 1, 2017 from https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture

[22] S. Ren, K. He, R. Girshick, and J. Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 6 (June 2017), 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

[23] Scipy.Org. [n. d.]. NumPy. ([n. d.]). Retrieved September 1, 2017 from http://www.numpy.org/

[24] Berkeley Vision and Learning Center. 2017. Caffe: a fast open framework for deep learning. (2017). Retrieved September 1, 2017 from https://github.com/BVLC/caffe

[25] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ÅĄukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). http://arxiv.org/abs/1609.08144