Whitepaper

GPU-Based Deep Learning Inference:
A Performance and Power Analysis

November 2015

Contents

Abstract

Deep learning methods are revolutionizing various areas of machine perception. On a high level, working with deep neural networks is a two-stage process: First, a neural network is *trained,* i.e. its parameters are determined using labeled examples of inputs and desired output. Then, the network is deployed to run *inference,* using its previously trained parameters to classify, recognize, and generally process unknown inputs.

It is widely recognized within academia and industry that GPUs are the state of the art in *training* deep neural networks, due to both speed and energy efficiency advantages compared to more traditional CPU-based platforms.

In this whitepaper, we take the next step and investigate GPU performance and energy efficiency for deep learning *inference*. We compare two standard deep learning frameworks, Caffe and Intel's Deep Learning Framework (IDLF), running on four publicly available hardware platforms, an NVIDIA Jetson™ TX1 developer kit, an NVIDIA GeForce GTX™ Titan X, an Intel Core™ i7 6700K, and an Intel Xeon™ E5-2698 v3.

Our results show that GPUs provide state-of-the-art inference performance and energy efficiency, making them the platform of choice for anyone wanting to deploy a trained neural network in the field. In particular, the Titan X delivers between 5.3 and 6.7 times higher performance than the 16-core Xeon E5 CPU while achieving 3.6 to 4.4 times higher energy efficiency. The Tegra X1 also achieves impressive results, achieving similar performance (258 vs. 242 images/second) with much higher energy efficiency (45.0 vs. 3.9 images/second/Watt) than the state-of-the-art Intel Core i7 6700K.

Introduction

Deep learning is revolutionizing the world. Many say the beginning of this revolution was the 2012 ImageNet competition [1] entry by Krizhevsky, Sutskever, and Hinton [2] using a convolutional neural network now referred to as "AlexNet" that outperformed the entire conventional computer vision competition by a large margin. Since then, one by one, deep neural networks (DNN) have conquered various algorithm domains related to computer vision in particular and machine perception in general. The potential use cases are endless: From self-driving cars [3] to faster drug development [4], from automatic image captioning [5] in online image databases to smart real-time language translation [6] in video chat applications, deep learning is providing exciting opportunities wherever machines interact with the human world.

These days, working with deep neural networks goes hand in hand with the use of GPUs. Deep learning users everywhere achieve dramatically reduced training times by transitioning from CPUs to one or more massively parallel GPU accelerators. With the release of the GPU-optimized CUDA-based cuDNN 3 library [7], GPU deep neural network training took another leap forward, taking full advantage of NVIDIA's most recent Maxwell GPU architecture. Besides improved Maxwell support, cuDNN 3 also introduced FP16 storage data types, making it possible to train larger models and increase performance on bandwidth-limited neural network layers, along with other important new features. NVIDIA DIGITS [8], built on top of cuDNN 3 running on one or more Maxwell GPUs, is a powerful application that makes it simpler than ever before to train neural networks.

Once a deep neural network has been trained for hours, days, or weeks, the next step is to put it to use by deploying it in the field, where it will repeatedly run *inference* computations. Examples of inference (giving an input to a DNN which then extracts information based on that input) include classifying images, localizing faces, or translating human speech in real-time.

Inference versus Training

Both DNN training and Inference start out with the same *forward propagation* calculation, but training goes further. After forward propagation, the results from the forward propagation are compared against the (known) correct answer to compute an error value. A *backward propagation* phase propagates the error back through the network's layers and updates their weights using gradient descent in order to improve the network's performance at the task it is trying to learn. It is common to batch hundreds of training inputs (for example, images in an image classification network or spectrograms for speech recognition) and operate on them simultaneously during DNN training in order to prevent overfitting and, more importantly, amortize loading weights from GPU memory across many inputs, increasing computational efficiency.

For inference, the performance goals are different. To minimize the network's end-to-end response time, inference typically batches a smaller number of inputs than training, as services relying on inference to work (for example, a cloud-based image-processing pipeline) are required to be as responsive as possible so users do not have to wait several seconds while the system is accumulating images for a large batch. In general, we might say that the per-image workload for training is higher than for inference, and while high *throughput* is the only thing that counts during training, *latency* becomes important for inference as well.

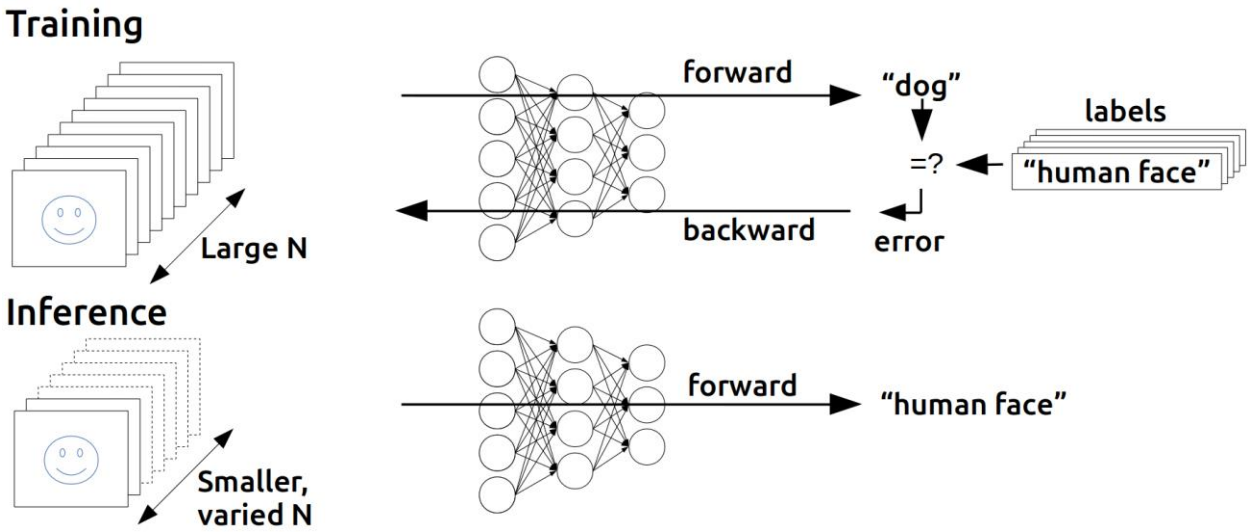

Figure 1: Deep learning training compared to inference. In training, many inputs, often in large batches, are used to train a deep neural network. In inference, the trained network is used to discover information within new inputs that are fed through the network in smaller batches.

Achieving the right balance between latency (due to wait time while assembling a batch, and because larger batches take longer to process) and throughput (because larger batch sizes utilize computing resources more efficiently) is application-dependent; but generally speaking, applications strive to

maximize a usable batch size while keeping latency under a given application-specific maximum. This enables inference computations to happen at various scales, all the way from cloud applications running in a large-scale datacenter to real-time inference, such as pedestrian detection, on an embedded processor inside an autonomous vehicle.

One particularly useful aspect of deep neural networks is their adaptability. Figure 2 shows the architecture of the AlexNet neural network. At a high level, AlexNet consists of five convolutional layers (starting from the left) and three fully connected layers (on the right). While the convolutional layers extract features from the input images given to the network, the fully connected layers learn to classify these features in order to fulfill the classification task at hand.

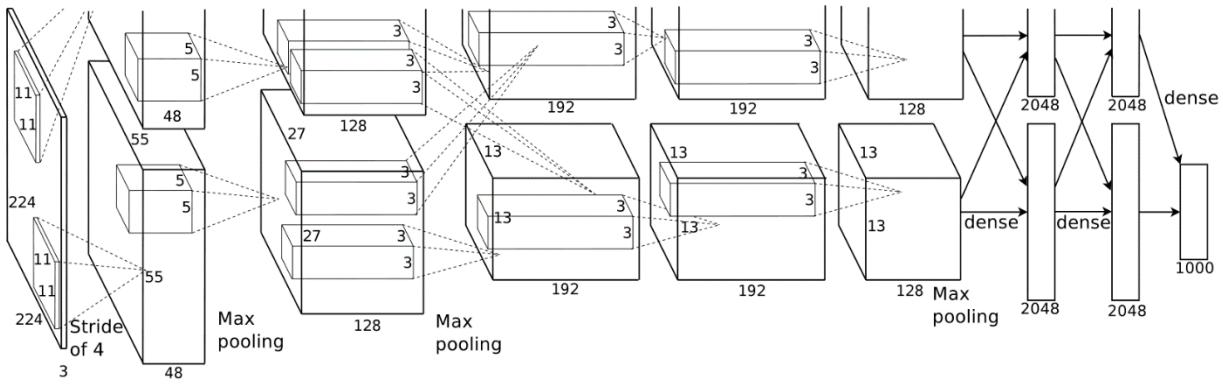

Figure 2 AlexNet neural network architecture. AlexNet consists of five convolutional layers of varying size (starting from the left) followed by three fully connected layers at the end. The input to the network is a 224x224 RGB image.

Even a relatively simple architecture such as AlexNet can adapt to many of the deep learning use cases described earlier. For example, in a datacenter application, AlexNet may run inference on down-sampled versions of user-submitted images to generate keyword suggestions. In a self-driving car, an AlexNet-like network can be used to identify objects and their bounding boxes (for example to detect and avoid obstacles).

GPUs Excel at Neural Network Inference

Neural network processing implementations like eBay's maxDNN [9] have already shown that tuned fully custom deep learning codes can achieve stellar performance with very high GPU utilization (more than 96% for maxDNN). However, for developing new networks, it is also valuable to be able to work within a standard framework that makes the process of testing and modifying networks fast and lightweight. Therefore, in this whitepaper, we consider how much performance one can achieve while continuing to work within the popular Caffe deep learning framework [10]. To achieve high performance, we combine Caffe with the upcoming cuDNN 4 library that is being released as a preview version with the Jetson™ TX1 product, with general availability following within a few months.

Inference Optimizations in Caffe and cuDNN 4

Where previous versions of cuDNN have mostly focused on improving training performance and providing support for the various neural network layers used by researchers, cuDNN 4 provides a series of optimizations for inference both on small and big GPUs.

With previous versions of cuDNN, one challenging area of the deep learning design space has been inference on big GPUs like the GeForce GTX Titan X that has 24 multiprocessors (SMs) for a total of 3,072 CUDA cores. In small batch scenarios, existing convolution algorithms struggled to fill the entire GPU because the parallelization strategy, albeit efficient, simply did not create enough parallel threads to fill the machine. cuDNN 4 improves this scenario by using a more efficient convolution algorithm. cuDNN 3 computed convolutions using an algorithm called a precomputed implicit GEMM (generalized matrix-matrix product) that is optimized for large output matrices. Unfortunately, batch size is a multiplicative factor in one of the output matrix dimensions. For example, on AlexNet convolution layer 5, the output matrix has dimensionality (Nx13x13) x (128), where N is equal to the batch size. For this particular layer and single-image inference, the default parallelization strategy of precomputed implicit GEMM launches only *two* thread blocks, or eight warps in total. As each thread block is assigned to exactly one of the GPU's streaming multiprocessors, 22 out of 24 SMs remain unused in this case. In cuDNN 4, this algorithm has been improved to split the work in an additional dimension. This reduces the amount of work per thread block, but enables launching significantly more blocks, which increases occupancy and, in turn, performance.

Another major improvement in cuDNN 4 is related to reduced precision floating point performance. In cuDNN 3, we introduced support for 16-bit floating-point (FP16) storage, i.e. neural net layer activations could be stored in reduced precision but all computations were still performed in FP32. Using FP16 has been proven to achieve the same classification accuracy as FP32 [11] [12]. FP16 storage also improves performance in bandwidth-limited scenarios and reduces the overall memory footprint required to run a network. However, convolutional layer performance is typically not limited by memory bandwidth, and so convolutional layers did not see much benefit from FP16 storage. In cuDNN 4, we provide support for FP16 arithmetic in convolution algorithms. On supported chips, such as Tegra X1 or the upcoming Pascal architecture, FP16 arithmetic delivers up to 2x the performance of equivalent FP32 arithmetic. Just like FP16 storage, using FP16 arithmetic incurs no accuracy loss compared to running neural network inference in FP32.

In addition to the new features in cuDNN 4, inference scenarios provide opportunity for other optimizations inside the Caffe framework that are not relevant to or possible in training use cases.

A key optimization for inference without batching (i.e. with a batch size equal to 1) is to use matrix-vector multiplication on fully connected layers instead of matrix-matrix multiplication. In the batched inference case, fully connected layers multiply a weight matrix A by a matrix of activations B, where each matrix column corresponds to the activation vector of one image in the batch. Caffe implements this operation using the GPU-optimized cuBLAS library for dense linear algebra. In the non-batched case, the activation matrix is just a vector, as it contains only a single column. Implementing this operation with a GPU matrix-matrix multiplication achieves very low efficiency, because cuBLAS matrix multiplication is optimized for large matrices. Thankfully, cuBLAS also provides a much more efficient GEMV (generalized matrix-vector product) operation that is a perfect fit here. This particular optimization has already been pushed to the official Caffe github repository [13], and is available today.

cuDNN 4 contains many more optimizations besides the ones described here, and the cuDNN engineers at NVIDIA are continuously working on improving performance further by integrating state-of-the-art research results.

Experimental Setup and Testing Methodology

For our performance analysis, we focus on two classical neural network architectures: AlexNet (2012 ImageNet winner), and the more recent GoogLeNet [14] (2014 ImageNet winner), a much deeper and more complicated neural network compared to AlexNet.

To cover a range of possible inference scenarios, we will consider two cases. The first case allows batching many input images together, to model use cases like inference in the cloud where thousands of users submit images every second. Here, large batches are acceptable, as waiting for a batch to assemble does not add significant latency. The second case covers applications that are extremely latency-focused; in this case, some batching is usually still feasible but for our testing, we consider the worst case of no batching at all.

We compare four different devices: The NVIDIA Tegra X1 and the Intel Core i7 6700K as client-side processors, and the NVIDIA GeForce GTX Titan X and a 16-core Intel Xeon E5-2698 v3 as high-end processors. To run the neural networks on the GPU, we use Caffe compiled for GPU usage using cuDNN. We use the default Caffe version provided by BVLC for FP32 benchmarking on Tegra X1, and NVIDIA's Caffe branch providing FP16 support [15] and enabling more efficient convolutions [16] when benchmarking FP16 Tegra X1 and Titan X, respectively. The Intel CPUs run the most optimized CPU inference code available, the recently release Intel Deep Learning Framework (IDLF) [17]. IDLF only supports a neural network architecture called CaffeNet that is similar to AlexNet with batch sizes of 1, 8, and 48. We do not provide GoogLeNet numbers for the CPUs as that would require running the CPU version of Caffe, which is widely known to not provide competitive performance.

For reproducibility reasons, we attempt to stay as close as possible to the publicly downloadable packages and standard benchmarks. For IDLF, we made a minor fix to fully utilize all the threads on a single socket system, and compiled the package with Intel's ICC compiler. The only supported neural network architecture in IDLF, CaffeNet[1], has fewer total operations than AlexNet because the local response normalization layers are smaller, but we accept that minor difference as CaffeNet is still a close approximation to AlexNet. For Caffe runs on the GPU, we start with the Caffe-provided bvlc_alexnet[2] protocol buffer files, and use Caffe's standard performance benchmarking mode, "caffe time." We remove per-layer timing code to avoid any associated overhead and only measure the full end-to-end inference execution time. The default benchmark protocol buffer files fill in all the image and weight data with zeros, which would not result in accurate data toggling rates for power measurements. To address that, we made minor modifications to the neural network descriptor files to use a DummyDataLayer that generates nonzero constant data as input images, and we fill in the weight data with a Gaussian distribution. In our internal testing, this closely matches the power when running real images with pretrained weights.

To compare power between different systems, it is important to measure power at a consistent point in the power distribution network. Power is distributed at a high voltage (pre-regulation) and then voltage regulators convert the high voltage to the correct level for the system-on-chip (SOC) and DRAM (post-regulation), typically with roughly 20% of power lost in the regulator and 80% remaining to provide to the SOC and DRAM. For our analysis, we are comparing pre-regulation power of the entire application

[1] https://github.com/01org/idlf/blob/master/bin/cpu_caffenet.config
[2] https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

processor (AP) / SOC and DRAM combined. On Xeon E5, IDLF is running on only one socket with a single Xeon E5 processor. CPU socket and DRAM power are as reported by the pcm-power utility, which we believe are measured on the input side of the associated regulators. To measure CPU and DRAM power on the Core i7, we instrumented major supply rails after the corresponding inductors to give us post-regulation power. We assume 80% regulator efficiency to report pre-regulation AP+DRAM power for the Core i7. To measure pre-regulation power for Tegra X1, we use a production Jetson™ TX1 module powered by a 9V supply with major supply rails (CPU, GPU, SOC, DDR_IO and DDR) instrumented at the input side of the regulators. On the Titan X, we report the total board power (both PCIE and 12V connectors combined) consumed by a production Titan X card. We do not include the system CPU's power in our Titan X measurements as the entire computation is happening on the GPU; the CPU only submits the work to the GPU.

Inference on Small and Large GPUs

We now present results for the case without batching (batch size 1) and with high batching (we pick a standard value of 128 for the GPUs, and the IDLF-supported maximum of 48 for the CPUs). This allows us to both see how fast and efficient the different processors can be in the optimal (high batch) case, while simultaneously being able to investigate how much utilization and efficiency the different devices are able to retain in non-batched scenarios.

On Tegra X1, we show results for inference in both 16-bit and 32-bit floating point precision, as Tegra X1's higher FP16 arithmetic throughput and the FP16 arithmetic support in cuDNN 4 allow for significantly higher performance without incurring any losses in classification accuracy.

Table 1 shows the first set of results comparing Tegra X1 to the Core i7 6700K. The table shows performance (images/second), power (Watts) and energy efficiency (images/second/Watt). Tegra X1 supports a range of clock frequencies up to 998 MHz. For our results, we configured TX1 to run with 690 MHz GPU core clock in order to achieve performance similar to the Core i7.

Network: AlexNet	Batch Size	Tegra X1 (FP32)	Tegra X1 (FP16)	Core i7 6700K (FP32)
Inference Performance	1	47 img/sec	67 img/sec	62 img/sec
Power		5.5 W	5.1 W	49.7 W
Performance/Watt		8.6 img/sec/W	13.1 img/sec/W	1.3 img/sec/W
Inference Performance	128 (Tegra X1) 48 (Core i7)	155 img/sec	258 img/sec	242 img/sec
Power		6.0 W	5.7 W	62.5 W
Performance/Watt		25.8 img/sec/W	45.0 img/sec/W	3.9 img/sec/W

Table 1 Inference performance, power, and energy efficiency on Tegra X1 and Core i7 6700K. We present cases without batching (batch size 1) and with large batching (128 on the GPU, 48 on the CPU). FP16 results are comparable to FP32 results as FP16 incurs no classification accuracy loss over FP32.

The results show that inference on Tegra X1 with FP16 is an order of magnitude more energy-efficient than CPU-based inference, with 45 img/sec/W on Tegra X1 in FP16 compared to 3.9 img/sec/W on Core i7 6700K, while achieving similar absolute performance levels (258 img/sec on Tegra X1 in FP16 compared to 242 img/sec on Core i7).

Table 2 shows the results for the Titan X GPU and the Xeon E5-2698 v3 server-class processor.

Network: AlexNet	Batch Size	Titan X (FP32)	Xeon E5-2698 v3 (FP32)
Inference Performance	1	405 img/sec	76 img/sec
Power		164.0 W	111.7 W
Performance/Watt		2.5 img/sec/W	0.7 img/sec/W
Inference Performance	128 (Titan X) 48 (Xeon E5)	3216 img/sec	476 img/sec
Power		227.0 W	149.0 W
Performance/Watt		14.2 img/sec/W	3.2 img/sec/W

Table 2 Inference performance, power, and energy efficiency on Titan X and Xeon E5-2698 v3.

The comparison between Titan X and Xeon E5 reinforces the same conclusion as the comparison between Tegra X1 and Core i7: GPUs appear to be capable of significantly higher energy efficiency for deep learning inference on AlexNet. In the case of Titan X, the GPU not only provides much better energy efficiency than the CPU, but it also achieves substantially higher performance at over 3000 images/second in the large-batch case compared to less than 500 images/second on the CPU. While larger batch sizes are more efficient to process, the comparison between Titan X and Xeon E5 with no batching proves that the GPU's efficiency advantage is present even for smaller batch sizes. In comparison with Tegra X1, the Titan X manages to achieve competitive Performance/Watt despite its much bigger GPU, as the large 12 GB framebuffer allows it to run more efficient but memory-capacity-intensive FFT-based convolutions.

Finally, Table 3 presents inference results on GoogLeNet. As mentioned before, IDLF provides no support for GoogLeNet, and alternative deep learning frameworks have never been optimized for CPU performance. Therefore, we omit CPU results here and focus entirely on the GPUs. As GoogLeNet is a much more demanding network than AlexNet, Tegra X1 cannot run batch size 128 inference due to insufficient total memory capacity (4GB on a Jetson™ TX1 board). The massive framebuffer on the Titan X is sufficient to allow inference with batch size 128.

Network: GoogLeNet	Batch Size	Titan X (FP32)	Tegra X1 (FP32)	Tegra X1 (FP16)
Inference Performance	1	138 img/sec	33 img/sec	33 img/sec
Power		119.0 W	5.0 W	4.0 W
Performance/Watt		1.2 img/sec/W	6.5 img/sec/W	8.3 img/sec/W
Inference Performance	128 (Titan X) 64 (Tegra X1)	863 img/sec	52 img/sec	75 img/sec
Power		225.0 W	5.9 W	5.8 W
Performance/Watt		3.8 img/sec/W	8.8 img/sec/W	12.8 img/sec/W

Table 3 GoogLeNet inference results on Tegra X1 and Titan X. Tegra X1's total memory capacity is not sufficient to run batch size 128 inference.

Compared to AlexNet, the results show significantly lower absolute performance values, indicating how much more computationally demanding GoogLeNet is. However, even on GoogLeNet, all GPUs are capable of achieving real-time performance on a 30 fps camera feed.

Conclusion

The results in this whitepaper show that GPUs are just as fast and efficient for deep learning inference as they are already known to be for training. Our results suggest that the GPU advantage holds true both in the high-performance and consumer computing spaces, and extends all the way from NVIDIA's most powerful GPU (the Titan X) to one of NVIDIA's smallest processors, the Tegra X1.

References

[1] ImageNet, "Large Scale Visual Recognition Challenge (ILSVRC)," [Online]. Available: http://image-net.org/challenges/LSVRC/.

[2] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012.

[3] C. Chen, A. Seff, A. Kornhauser and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," arXiv, 2015.

[4] T. Unterthiner, A. Mayr, G. Klambauer, M. Steijaert, J. K. Wegner, H. Ceulemans and S. Hochreiter, "Deep Learning as an Opportunity in Virtual Screening," in *Proceedings of the Deep Learning Workshop at NIPS*, 2014.

[5] A. Karpathy and L. Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions," in *Proceedings of the Annual Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, 2015.

[6] D. Bahdanau, K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," arXiv, 2014.

[7] NVIDIA, "NVIDIA cuDNN - GPU Accelerated Deep Learning," [Online]. Available: https://developer.nvidia.com/cudnn.

[8] NVIDIA, "NVIDIA DIGITS - Interactive Deep Learning GPU Training System," [Online]. Available: https://developer.nvidia.com/digits.

[9] A. Lavin, "maxDNN: An Efficient Convolution Kernel for Deep Learning with Maxwell GPUs," 2015.

[10] Berkeley Vision and Learning Center, "Caffe," [Online]. Available: http://caffe.berkeleyvision.org/.

[11] J. Dean, "Techniques and Systems for Training Large Neural Networks Quickly," Google.

[12] S. Gupta, A. Agrawal, K. Gopalakrishnan and P. Narayanan, "Deep Learning with Limited Numerical Precision," arXiv, 2015.

[13] Berkeley Vision and Learning Center, "GitHub Caffe repository," [Online]. Available: https://github.com/BVLC/caffe.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," 2014.

[15] NVIDIA, "GitHub FP16 Caffe repository," [Online]. Available: https://github.com/NVIDIA/caffe/tree/experimental/fp16.

[16] NVIDIA, "GitHub Caffe repository," [Online]. Available: https://github.com/NVIDIA/caffe/tree/caffe-0.14.

[17] Intel, "Intel Deep Learning Framework," [Online]. Available: https://01.org/intel-deep-learning-framework.

Notice

Copyright