Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs

Da Li*[1], Xinbo Chen*[2], Michela Becchi[1], Ziliang Zong[2]
[1]Dept. of Electrical and Computer Engineering, University of Missouri
[2]Dept. of Computer Science, Texas State University
dlx7f@mail.missouri.edu, x_c7@txstate.edu, becchim@missouri.edu, ziliang@txstate.edu

***Abstract* — In recent years convolutional neural networks (CNNs) have been successfully applied to various applications that are appropriate for deep learning, from image and video processing to speech recognition. The advancements in both hardware (e.g. more powerful GPUs) and software (e.g. deep learning models, open-source frameworks and supporting libraries) have significantly improved the accuracy and training time of CNNs. However, the high speed and accuracy are at the cost of energy consumption, which has been largely ignored in previous CNN design. With the size of data sets grows exponentially, the energy demand for training such data sets increases rapidly. It is highly desirable to design deep learning frameworks and algorithms that are both accurate and energy efficient. In this paper, we conduct a comprehensive study on the power behavior and energy efficiency of numerous well-known CNNs and training frameworks on CPUs and GPUs, and we provide a detailed workload characterization to facilitate the design of energy efficient deep learning solutions.**

***Keywords*—**energy-efficiency; neural networks; deep learning; GPUs

I. Introduction

The history of neural network research can be traced back to the second half of the last century. In 1958, Frank Rosenblatt, a psychologist, proposed the concept of *Perceptron* and a theory on the operation of neurons in the human brain [2]. This theory has led to the emergence of a new field of artificial intelligence, called *neural networks*. About thirty years later, Yann LeCun et al. [3] successfully applied neural networks to recognize handwritten checks and ZIP codes in mail. However, the training of their neural network required approximately three days. Later, neural networks became a core computation in various applications, from wake-sleep algorithms [4] to the vanishing gradient problem [5]. However, the high computational requirements of the training phase continue to be a key factor hindering the advancement of algorithms and applications based on neural networks. Recent advancements in software and hardware, including the use of high throughput GPUs to accelerate neural network training, have alleviated this problem. It is now possible to train large and complex neural networks in reasonable time on relatively inexpensive hardware. This has led to the rapid growth of neural network-based deep learning algorithms.

Many deep learning algorithms used in computer vision rely on convolutional neural networks (a.k.a. CNN). The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [6], which runs annually since 2010, allows researchers to evaluate their object detection and image classification algorithms on over 15 million labeled high-resolution images from roughly 22 thousands categories. In 2012, a team from the University of Toronto improved the error rate of image classification from 25.8% to 16.14% using a CNN model (*AlexNet* [7]). This neural-network-based approach has been subsequently increasingly adopted leading to continuous improvements in the classification accuracy [8, 9], and CNNs are nowadays widely used in image recognition, segmentation and object detection [10]. With continuous improvement in neural network models and algorithms, the accuracy of ILSVRC has already surpassed human recognition ability (~5%) in 2015 [11]. The need for large CNN models providing good classification accuracy has led to efforts aimed to accelerate neural networks' training.

However, the race for speed and accuracy comes at the cost of energy consumption, an aspect that has been overlooked in previous work. While the classification accuracy has traditionally been considered as the primary metric of success for image and video recognition applications, the community has recently recognized the need for deep neural network implementations that are both accurate and energy efficient. As a result, in 2015 IEEE Rebooting Computing has launched the "Low-Power Image Recognition Challenge" (*LPIRC*), an initiative aimed to promote the design of energy efficient image classification methods. Currently, research in designing energy efficient CNNs is still in its infancy. The knowledge on the power consumption behaviors of different CNNs and training frameworks is very limited.

Modern GPUs comprise hundreds to thousands of compute cores and can provide high computational power and throughput. As a result, GPU computing has become the de-facto approach for CNNs training [12]. Meanwhile, Intel has recently released the Intel Deep Learning Framework (IDLF) [13], which provides high performance CNNs training on CPU platforms. While Nvidia claims that its GPU framework reports a 11x-14x speedup over the CPU version of Caffe on an Intel IvyBridge processor [14], S. Hadjis et al. [15] point out that Nvidia's comparison is based on an

* = authors contributed equally

unoptimized CPU baseline, and they introduce CPU optimizations reducing the GPU-CPU performance gap to only 1.86x. None of these analyses, however, consider the energy efficiency of training the neural networks.

In this work, we conduct a comprehensive study on the power behavior and energy efficiency of CNNs on both CPUs and GPUs. We evaluate popular deep learning frameworks using energy-related metrics and, for each of these frameworks, we provide accurate power measurements using a set of carefully selected networks and layers. We conduct our evaluation on different processor architectures (i.e., Intel Xeon CPUs, Nvidia Kepler and Maxwell GPUs) and explore the effect of a variety of hardware settings on performance and energy-efficiency.

Our contributions can be summarized as follows:

- We present a comparative study on the energy efficiency of several popular CNN training frameworks on different hardware (i.e., Xeon CPU, K20 GPU, Titan X GPU). Our results provide insights on how the selection of the neural network design as well as the software and hardware configuration affect the energy efficiency of the application.
- We propose a methodology to analyze both the network- and layer-wise energy consumption of CNN architectures. By quantifying the energy consumption of each CNN layer, we identify the power-hungry layers in the neural network.
- We analyze the behavior of different implementations (i.e., OpenMP-based) and libraries (e.g. ATLAS, OpenBLAS, cuDNN) on CPUs and GPUs. Our analysis covers both performance and energy consumption, allowing either performance- or energy efficiency-oriented tuning.
- We study the impact of different hardware settings (i.e., Hyper-Threading, ECC) on the training of CNN models. We also explore the impact of dynamic voltage/frequency scaling (DVFS) on power and energy consumption.

II. Background

Deep learning is a branch of machine learning that uses a layered architecture of data processing stages for pattern recognition. Due to its effectiveness in many applications, deep learning has gained popularity in both academia and industry. Convolutional neural networks (CNNs) are the most successful models for deep learning, and they have been used in various domains, including computer vision [16] and speech recognition [17].

A. Convolutional Neural Networks

At a high level, convolutional neural networks simulate the way in which human brains process and recognize images. They belong to the family of multi-layer perceptrons (MLP) [2]. A MLP is a multi-layer neural network consisting of an input layer, an output layer and multiple hidden layers between the input and output layers. Each hidden layer represents a function between its inputs and outputs that is defined by the layer's parameters.

Convolutional neural networks mainly consist of three types of layers: *convolutional layers* (Conv), *pooling layers* (Pooling) and *fully connected layer*s (FC). Each layer may contain thousands to millions of neurons. A single neuron takes some inputs, computes their weighted sum, and sends the output to the neurons in the next layer. In this way, distinct layers apply different operations to their inputs and produce outputs for the layers that follow. Figure 1 shows an example of convolutional neural network (LetNet [1]), which consists of alternated convolutional and pooling layers followed by a few fully-connected layers.

Convolutional layers: These layers apply convolutions to the input with several filters and add a bias term to the results. Very often, a nonlinear function (called *activation function*) is also applied to the results. Convolutional layers exploit spatial connectivity and shared weights. The parameters of a convolutional layer are reduced dramatically compared to a typical hidden layer of a MLP. Convolutional layers are the most computational intensive layers in CNNs.

Pooling layers: These layers perform a nonlinear down-sampling operation on the input. They partition the input into a set of sub-regions and output sampled results from these sub-regions. Based on their sampling method, pooling layers can be categorized into: *maximum pooling*, *average pooling* and *stochastic pooling*. Pooling layers progressively reduce the amount of parameters as well as control model over-fitting. Pooling layers are usually placed between two convolutional layers.

Fully connected layers: Unlike in convolutional layers, neurons in FC layers have full connections to all output from the preceding layers. As a consequence, a FC layer has many more parameters than a convolutional layer. Nonetheless, since convolution operations are replaced by multiplications, fully connected layers require less computational power.

B. Learning and inference

Using CNNs for machine learning tasks involves three steps: (1) designing the CNN architecture, (2) learning the parameters of the CNN (also called "training"), and (3) using the defined CNN for inference. Since CNNs are back-propagation learning algorithms, their learning phases can be divided into: *forward propagation*, *backward propagation* and *weight update*. In the forward propagation phase, input data are sent to the neural network to generate the outputs. In the backward propagation phase, the errors between the standard outputs and the produced outputs are propagated in a backward fashion to compute the errors in each layer. These errors (also called *gradients*) in each layer will be used in every weight update. However, for inference, the parameters of the networks are given and there is only

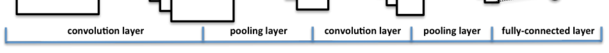

Fig. 1. Example of Deep Convolutional Neural Network (LeNet). Image source: [1].

forward propagation to produce the prediction.

III. Methodology

A. Introduction to CNN Frameworks

In order to be practically applicable, CNNs require software frameworks that allow high performance training of large-scale networks including millions of parameters. Popular open-source CNN frameworks include: *Caffe* [18], *Torch* [19], *TensorFlow* [20], *MXNet* [21], *Nervana* [22] and *CaffeConTroll* [15]. All these frameworks except CaffeConTroll offer both CPU- and GPU-support. In all cases, GPU support is based on the Nvidia cuDNN library [23]. In addition to the cuDNN-based implementation, Caffe and Torch include custom GPU implementations of convolutional layers, pooling layers and activation functions.

B. Experimental Setup

Benchmark Suite: In our experimental evaluation we use Convnet [24], an open-source benchmark that includes most publicly accessible implementations of CNNs. This benchmark is designed to measure the execution time of the forward and backward propagation of different layers/networks. To obtain accurate power measurements, we apply minor modifications (as described in [25]) to the Convnet benchmark (e.g., we remove unnecessary timing and logging code). We test the training phase (forward and backward propagation) and configure each run to have multiple (>100) iterations, which simulates the real use of these frameworks. Also, by default, the batch size is set to 128, which is a commonly used value.

Neural networks: In our evaluation we use four ImageNet winner neural network models: *AlexNet* v2 [26], *OverFeat* [27], *VGG_A* [8] and *GoogleNet* [9]. These networks have been proven successful models and are included in the Convnet benchmark.

Hardware: We perform all experiments on a single machine including a 16-core Intel Xeon E5 - 2650 v2 @ 2.6GHz with Hyper-Threading enabled, an Nvidia Tesla K20m GPU with 5GB memory and an Nvidia Titan X GPU with about 12GB memory. The machine has 32GB DDR3 main memory and a 128 GB SSD hard drive, and runs CentOS v7.

Software: The drivers, libraries and frameworks used in our experiments are: CUDA 7.0, cuDNN v3, OpenBLAS 0.2.16, Caffe (commit ID be163be), Torch 7 (commit ID eb8d7f2), and TensorFlow (commit ID fd464ca), MXNet (commit ID d25053) and CaffeConTroll (commit ID 8191f6c). The CPU and DRAM power data are collected via Intel's Running Average Power Limit (RAPL) interface [28] and the GPU power is obtained via the Nvidia System Management Interface [29].

C. Organization of Experiments

Our evaluation starts with an analysis of performance, power and energy consumption of the aforementioned CNN frameworks on GPU and CPU (Section IV). This analysis also covers the use of different libraries to perform NN operations (cuDNN on GPU; Atlas, OpenBLAS and MKL on CPU). We then study how the network topology and the setting of the batch size and of several hardware parameters (hyper-threading, ECC, and DVFS) affect performance and energy efficiency (Section V and VI). Due to the popularity of the framework and in the interest of space, the evaluation presented in Section V and VI focuses on Caffe.

IV. Overall Energy Efficiency Results on CPU & GPU

A. Native GPU implementations versus cuDNN

Because GPUs can provide more computational power than general-purpose CPUs, most deep learning frameworks rely on GPUs to provide fast training and inference. To facilitate GPU-accelerated deep learning, Nvidia released the cuDNN library, which includes highly optimized implementations of common operations found in neural networks (e.g. convolution, pooling). Although some early developed frameworks like Caffe and Torch have their own GPU implementations, newer frameworks (e.g. TensorFlow) rely

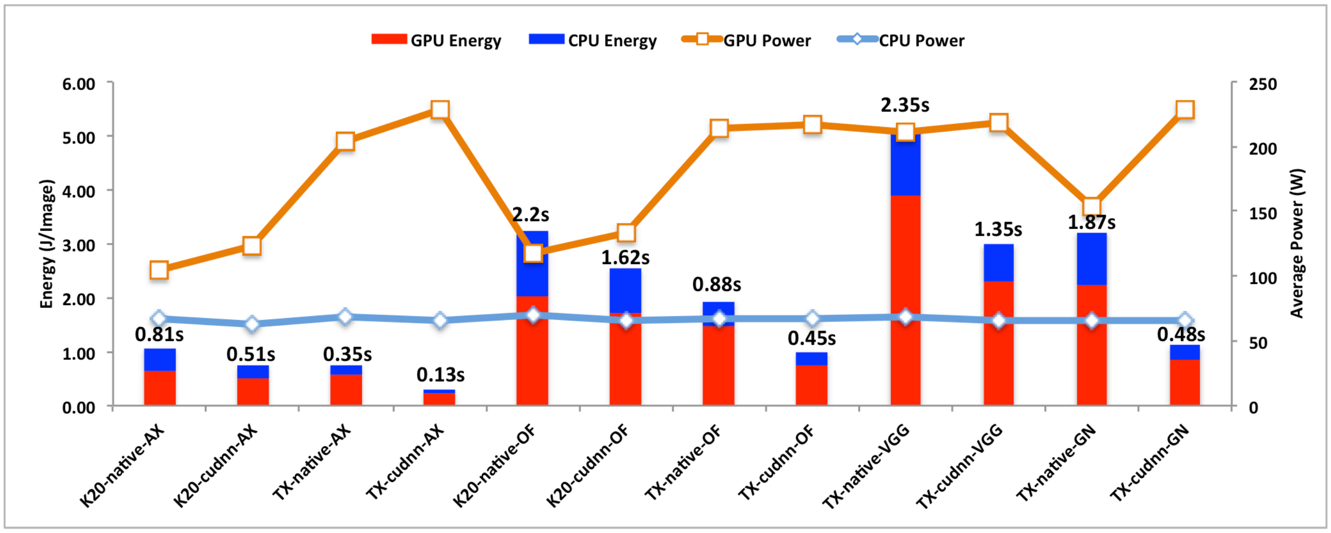

Fig. 2. Comparison between native GPU implementation and cuDNN v3 library in Caffe. The numbers on the bars represent the execution time of a single forward and backward propagation iteration.

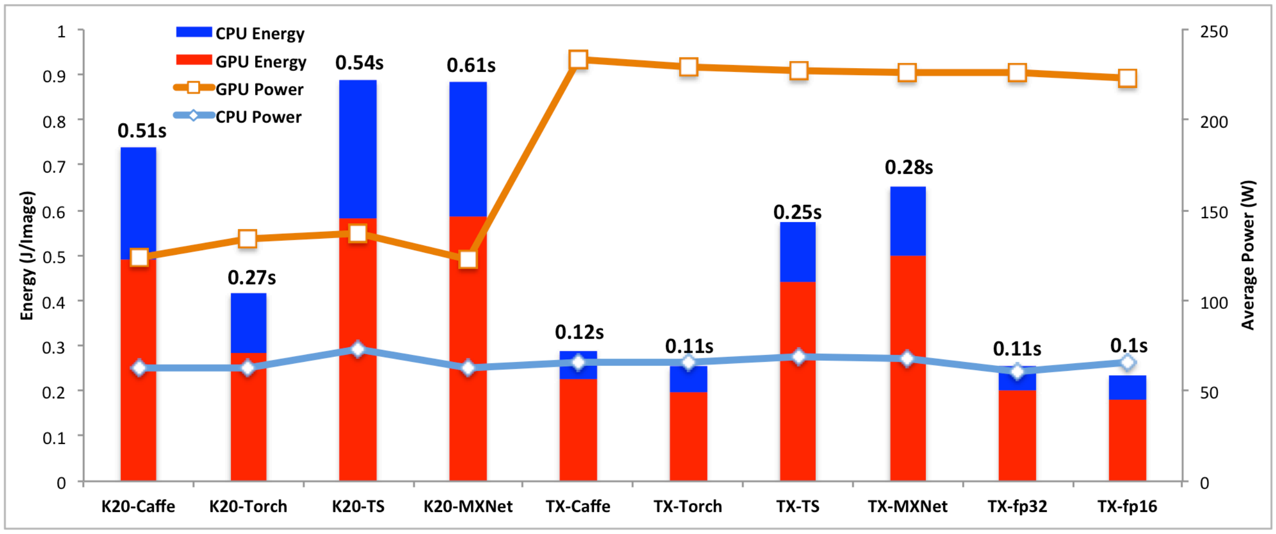

Fig. 3. Comparison among different frameworks on K20m and TitanX GPUs. The numbers on the bars represent the execution time of a single forward and backward propagation iteration.

on cuDNN for neural network related operations. In this section we compare native GPU implementations of neural network operations with those found in the cuDNN library.

Figure 2 presents the results of this analysis on the Caffe framework, which can be configured to either use a native GPU implementation or rely on the cuDNN library. We tested 500 iterations of forward and backward propagation with a batch size of 128. In the figure, the bars represent the energy consumption per image processed (left y-axis). Specifically, the bottom (red) and top (blue) part of each bar indicate the energy consumption of GPU and CPU, respectively. The two lines show the average power consumption (in Watts) of GPU and CPU (idle); the power consumption scale is on the right y-axis. On top of each bar we report the execution time of a single iteration. In the experiments we use two Nvidia GPUs: a K20m and a Titan X (shown as K20 and TX along the x-axis, respectively) and four networks (AlexNet, OverFeat, VGG_A and GoogleNet). Due to its limited memory capacity, we could not run the VGG_A and GoogleNet networks on the K20m GPU.

As can be observed, cuDNN yields higher GPU power and lower energy consumption than native GPU code on both K20m and Titan X. The increase in power consumption from native GPU kernel to cuDNN varies from 2% (OverFeat on Titan X) to 48% (GoogleNet on Titan X). On average, cuDNN increases power consumption by 16% and reduces energy consumption by 42%. This can be explained as follows. Due to its higher GPU utilization, cuDNN leads to higher power consumption than native GPU code. However, by significantly reducing the training time, cuDNN diminishes the total energy consumption.

If we compare the power and energy consumption of the CNN code on different GPU platforms, we can conclude that, although Titan X consumes more power than K20m, it is more energy-efficient. Taking AlexNet as an example, when moving from K20m to Titan X, the energy consumption is reduced by 15% (for native GPU code) and by 54% (for cuDNN). Compared to K20m, Titan X can deliver more computational power, leading to higher power consumption but also to lower execution time. The reduction in execution time is significant enough to yield better energy efficiency despite the higher power consumption.

Figure 2 shows another interesting fact: the CPU consumes a significant amount of energy although it mostly is in the idle state, which should not be ignored. As can be seen, the CPU idle power is about 67w in all cases, while the GPU power varies from 104w to 134w on K20m and from 204w to 228w on Titan X. In general, the CPU accounts for 22% to 40% of the total energy consumption. This indicates that, to be energy-efficient, a CNN framework should utilize both CPU and GPU during the training phase.

Since cuDNN leads to better performance and is more energy-efficient than native GPU implementations of neural network related kernels, we use cuDNN as the default GPU library in all remaining experiments.

B. *GPU frameworks*

There are many frameworks that use GPUs to accelerate CNN training. We selected five popular frameworks: *Caffe* [18], *Torch* [19], *TensorFlow* [20], *MXNet* [21] and *Nervana* [22]. The first four support both K20m and Titan X, while Nervana supports only Titan X. Since Nervana supports 16- and 32-bit floating-point arithmetic, for this framework we have two settings (TX-fp16 and TX-fp32, respectively). Figure 3 reports the results obtained by running forward and backward propagation on AlexNet using a batch size of 128. As in Figure 2, the bars represent the energy consumption per image (y-axis on the left), the lines show the CPU/GPU power in Watts (y-axis on the right), and the numbers on the bars represent the execution time of each iteration.

As can be seen, in these experiments Titan X consumes an average power of 227w, 74% more than K20 (130w). Torch outperforms other frameworks in terms of energy efficiency on K20m, and performs similarly to Caffe and Nervana on Titan X. Using 16-bit floating-point arithmetic

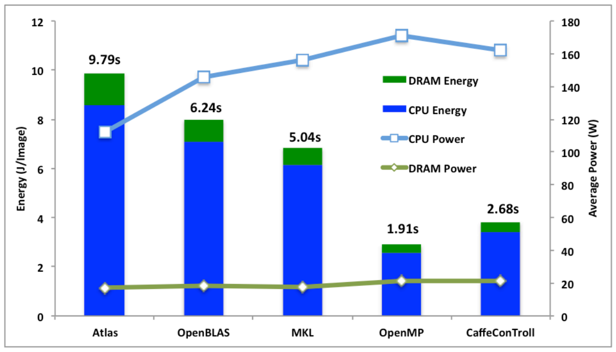

Fig. 4. Comparison among CNN frameworks on CPU.

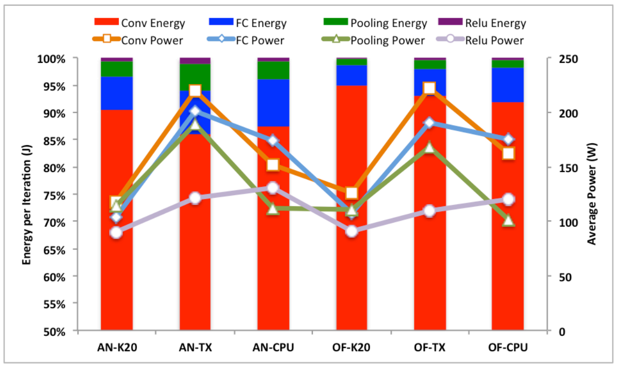

Fig. 5. Breakdown of energy consumption of AlexNet and OverFeat on K20, Titan X and CPU using Caffe.

allows a 10% energy reduction over single precision arithmetic. We can again observe that CPU idle power cannot be ignored, and is especially significant in the case of K20 (since this GPU leads to longer execution times).

C. *CPU frameworks*

In this section we study the energy and power behavior of CNN frameworks when using only CPUs. Because of its wide use and flexibility, in this set of experiments we focus on Caffe and its derivatives. Caffe supports three CPU libraries (Atlas, OpenBLAS and MKL) that can be statically configured. In our evaluation, we also consider Caffe-OpenMP (an optimized CPU version of Caffe) [30] and CaffeConTroll (a Caffe's derivative that uses an optimization called "*lowering*" [15]). The results of this comparison are shown in Figure 4.

The performance of these frameworks and libraries on CPU is significantly affected by the degree of multithreading. Our machine has 16 physical cores and all the CPU versions are configured to spawn at most 16 threads. To accurately measure power and energy consumption of CPU-based frameworks, we also measure and report the DRAM power data.

As can be seen from Figure 4, the average power consumption of different CPU-based frameworks varies from 103w to 188w. Compared to CPU, DRAM consumes a relatively small portion of energy (11%). Among these CPU versions, Caffe-OpenMP is the most energy-efficient and consumes 2.9 Joules per image processed. It is worth noting that while Caffe-OpenMP is more energy efficient than other CPU implementations, it consumes over twice the amount of energy than all considered GPU frameworks. We can conclude that CPUs are generally less energy-efficient than the GPUs for training CNNs.

V. Effect of NN and Batch Size Configuration

A. *NN Structure*

In this section, we focus on the impact of the network's structure on energy efficiency. To this end, we disassemble AlexNet and OverFeat into four types of layers (*convolutional*, *pooling*, *fully connected*, and *ReLU*) and measure the distribution of the energy consumption across these layers. In order to ensure that our approach is accurate, we verified that the cumulative energy consumption results from all layers are coherent with the measurements on the integrated network. In Figure 5 we show the percentage breakdown of the energy consumption across layers on GPU and CPU using Caffe. As can be seen, convolutional layers are predominant and consume 87% of the total energy consumption. The second most power-hungry layers are fully connected layers, which account for 10% of the total energy consumption. Pooling layers and ReLU layers (which apply activation functions) account for less than 5% of the energy consumption. This trend is even more noticeable for OverFeat. In OverFeat, the convolutional layers have more filters (leading to a larger number of neurons) than in AlexNet. Although OverFeat has the same number of layers as AlexNet, its convolutional layers consume a larger portion of the energy, as high as 95%, 93% and 92% on K20m, Titan X and CPU, respectively. From this analysis we can conclude that, in order to optimize energy consumption of deep CNNs, the priority should be on improving the energy efficiency of the convolutional layers.

B. *Batch Size*

The batch size is an important setting when training a neural network. Larger batch sizes lead to more images being packed into a batch and sent to the network for training in a single iteration. Intuitively, larger batches allow more data-level parallelism. However, loading a larger batch requires more memory to store the data, possibly exhausting the limited GPU memory.

Figure 6 reports the power and energy consumption of AlexNet on K20, Titan X and CPU when the batch size varies from 8 to 256. On all three hardware platforms, increasing the batch size leads to linear growth in power. Small batch sizes (e.g., 8 or 16) cause CPU and GPU under-utilization. An increase in the batch size will yield higher hardware utilization, and, consequently, an increase in power consumption. However, as can be seen, increasing the batch size can reduce the energy consumption per image. For instance, when the batch size is increased from 8 to 16, the energy consumption per image is reduced by 13%, 18% and 31% on K20, Titan X and CPU, respectively. When the hardware utilization saturates, however, a further increase in the batch size does not further improve energy consumption. For example, when increasing the batch size from 128 to 256, the energy consumption per image reduces only by 4%

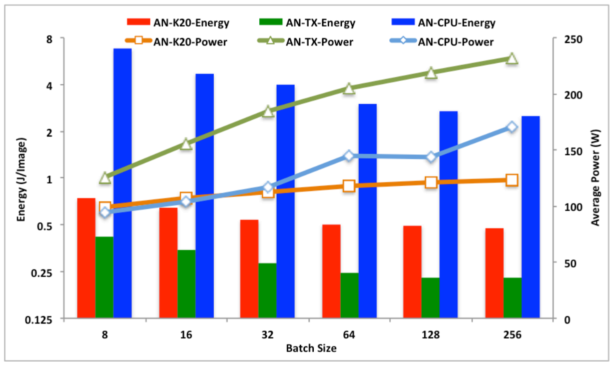

Fig. 6. Power (W) and energy consumption (Joule/Image) on K20m, Titan X and CPU with different batch sizes using Caffe.

on K20 and by 7% on CPU, while Titan X does not benefit from large batch sizes.

VI. Effect of Hardware Settings

In this section, we analyze the impact of different hardware settings on power and energy consumption. Specifically, we investigate how the use of Hyper-Threading on CPU and the use of ECC and DVFS on GPU affect the performance, power and energy consumption of CNNs.

A. Hyper-Threading

Hyper-Threading (HT) is a technology introduced by Intel in order to improve the performance obtainable through parallelization. With HT enabled, the operating system treats each physical processor as two logical cores. HT can improve the performance of memory/IO intensive applications by hiding their latencies, but it often degrades the performance of compute intensive workloads. To evaluate the impact of HT on deep learning frameworks, we conducted experiments with HT enabled/disabled, in each case configuring the number of threads to equal the number of logical cores. Because our machine has sixteen physical cores, when HT is enabled we set the application to spawn thirty-two threads.

Figure 7 shows the results reported on Caffe and its derivatives. CPU and DRAM energies are stacked into one bar named *HT-E-XXX* or *HT-D-XXX*. In the former case, HT is enabled; in the latter case, HT is disabled. The y-axis on the left side shows the energy consumption per image. The lines represent the power curves of CPU and DRAM and the power scale is on the y-axis on the right side. The values on the bars indicate the total execution time of each setting. Although different libraries and implementations experience different power consumptions, the power consumption is not significantly affected by HT. However, HT affects the execution time and, consequently, the energy consumption. As can be seen, while MKL reports a slight benefit from enabling HT (5% speedup and 4.8% energy saving), OpenBLAS, OpenMP, CaffeConTroll all suffer various degrees of performance degradation and consume more energy when HT is enabled. In the worst case (Caffe with OpenBLAS), enabling HT leads to an increase in execution time and energy consumption by 45% and 42%, respectively. This experiment shows that performance and energy

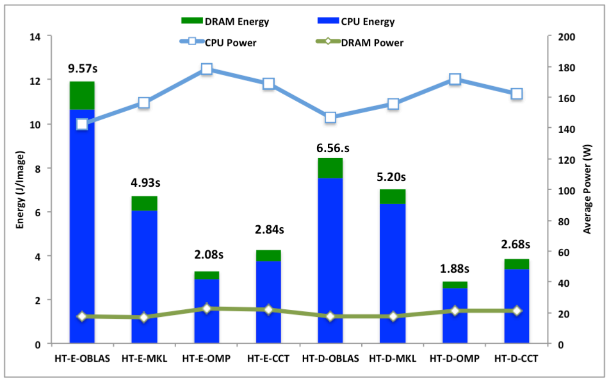

Fig. 7. Experimental results with different Hyper-Threading settings. HT-E-XXX indicates that HT is enabled, while HT-D-XXX indicates that HT is disabled. The experiments are conducted on Caffe.

efficiency of neural network training are generally negatively affected by HT, since this application saturates the hardware with computation and does not experience benefits from memory latency hiding.

B. ECC

Nowadays Error Correction Code RAM is the standard configuration for high performance computing clusters. With ECC enabled, RAM can detect and correct single bit errors. This feature is also available on high-end GPUs, like the ones used in this study.

Although enabling ECC can reduce errors, this feature does not come for free. When ECC is enabled, some bits are reserved, thus reducing the available memory footprint and bandwidth. In addition, enabling ECC causes applications to suffer from more expensive synchronization and uncoalesced memory accesses [31].

We have tested the use of ECC on K20m using Caffe along with the cuDNN library. Our experiments show no significant changes in performance and energy efficiency when ECC is enabled. This is because CNN training does not require synchronization operations and present relatively regular and coalesced memory access patterns. However, we have observed that enabling ECC leads to a 6.2% increase in memory utilization. Since deep CNN applications typically do not require bit-level accuracy and can tolerate random errors, for these applications disabling ECC can be beneficial in that it allows a better utilization of the limited memory capacity of the GPU. We note that modern GPUs have from 1 to 12 GB of device memory, while CPUs are typically equipped with 16GB up to 1 TB RAM.

C. DVFS

Dynamic Voltage and Frequency Scaling (*DVFS*) is an advanced power-saving technology that aims to lower a component's power state while still meeting the performance requirement of the workload [32]. Both Titan X and K20m GPUs support DVFS with various clock frequencies. On these devices, DVFS can control two clock frequencies: memory frequency and core frequency. Nvidia provides the *nvidia-smi* utility and the *Nvidia Management Library (NVML)* to control these frequencies.

TABLE I. MEMORY AND CORE FREQUENCIES SUPPORTED ON K20M GPU

Memory Frequency (MHz)	GPU Core Frequency (MHz)
2600	758
	705
	666
	640
	614
324	324

Table 1 shows the clock frequencies supported on K20m for GPU cores and memory. When the memory frequency is set to 324MHz, the only core frequency is 324Hz. Titan X support wider range of core and memory frequencies. Due to space limits, in this paper we only report results K20m.

Figure 8 shows the power, energy and performance results obtained when applying different memory and core frequencies to CNN training. In the experiments, the batch size is varied from 16 to 128. As expected, power consumption increases with the operation frequency. In the experiments, the power consumption varies from 44w at 324/324 MHz frequencies to 118w at 758/2600 MHz frequencies. This wide range of power consumption degrees shows that frequency scaling is an effective method that can be leveraged to meet power cap requirements.

For energy-consumption, our experiments show a "valley" trend: the energy-consumption is relatively high at both the lowest and highest frequencies, and is relatively low at intermediate frequencies. This is because low frequency leads to low power consumption at the cost of longer execution time, negatively affecting the total energy consumption. Conversely, increasing frequency beyond a certain level increases the power consumption while providing only a limited return on performance, also leading to energy inefficiency. This "energy-valley" trend indicates that training deep neural networks in an energy-aware fashion requires operating at frequencies that allow a good trade-off between execution time and power consumption.

VII. RELATED WORK

With the emergence of powerful GPUs and the availability of large data sets for training, we have witnessed a significant improvement of deep CNNs in terms of training time and accuracy. The Visual Geometry Group (VGG) at University of Oxford has designed a 16- and a 19-layer model with a 7.4% and a 7.3% top-5 error rate, respectively [8]. Microsoft has recently proposed a NN model with 152 layers – 8x deeper than the VGG nets – reporting a 3.57 % error rate [11]. This fast development has led to the proliferation of applications based on NNs. Examples of emerging applications based on NNs include: auto tagging [33], the estimation of a person's pose [34], and the generation of a descriptive sentence from an image [35]. While previous work has focused on performance, our paper focuses on evaluating the energy efficiency of deep neural networks on CPUs and GPUs.

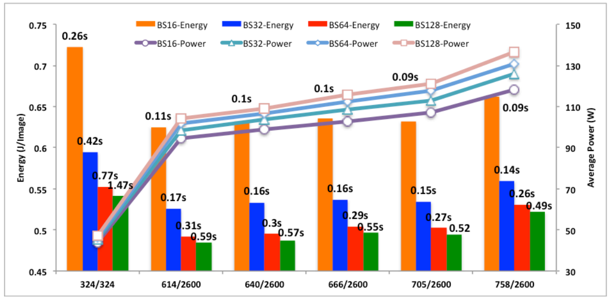

Fig. 8. Power and energy consumption using different memory and core frequencies. The batch size is varied from 16 to 128. The numbers on the bars are values of execution time per iteration.

Although the mainstream approach for training deep convolutional neural networks is using CPUs and GPUs, researchers have recently started to explore the use of other architectures and devices, including FPGA [36], RRAM [37], neuromorphic processors [38] and Tetra-Parallel architecture [39]. These hardware implementations are specifically tailored to convolutional neural networks and yield impressive results in terms of performance and energy efficiency. However, these hardware innovations are still at an early stage and it is urgent to understand the power and energy behavior of commonly used neural network frameworks on CPU and GPU – the main goal of our paper.

In spite of the advancement in the development of deeper and more complicated neural network structures, research on investigating the energy behavior of different neural networks and software frameworks is still in its infancy. In its white paper [25], Nvidia provides limited power and energy consumption results for CNN inference on two frameworks. Our paper distinguishes itself by providing a comprehensive study on the energy-efficiency of deep neural network training that covers different frameworks, different platforms and different hardware settings.

VIII. CONCOLUSION

In this paper, we have conducted a comprehensive evaluation of the energy efficiency of deep CNN training frameworks. We have performed experiments using several popular DNN frameworks and libraries. Our evaluation considers both CPU and GPU. Besides showing network-wide results, we have studied how the network topology and the batch size affect power and energy consumption. In this process, we have identified the power-hungry layers of two ImageNet-winner neural networks. Finally, we have explored the impact of several CPU and GPU hardware settings (i.e. HT, ECC and DVFS) on performance and energy efficiency of CNNs. Our results can be used for designing energy-efficient deep CNN frameworks and neural network architectures.

ACKNOWLEDGMENT

This work has been supported by NSF awards CNS-1216756, CCF-1452454, CNS-1305359, and by equipment donations from Nvidia Corporation.

References

[1] TheanoTeam. "Convolutional Neural Networks (LeNet)," http://deeplearning.net/tutorial/lenet.html.

[2] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychological review,* vol. 386, 1958.

[3] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard, “Handwritten digit recognition: Applications of neural net chips and automatic learning,” *IEEE Communication,* vol. 27, no. 11, pp. 41-46, 1989.

[4] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, “The" wake-sleep" algorithm for unsupervised neural networks,” *Science,* vol. 268, no. 5214, pp. 1158-1161, 1995.

[5] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* vol. 6, no. 02, pp. 107-116, 1998.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Santheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision,*vol.115, pp. 211-252, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proc. of NIPS, 2012.

[8] K. Simonyan, and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in Proc. of ICLR, 2015.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” in Proc. of CVPR, 2015.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” arXiv, no. 1312.5602, 2013.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification."

[12] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and A. Ng, “Deep learning with COTS HPC systems,” in Proc. of ICML, 2013.

[13] Intel. "Intel® Deep Learning Framework," 2016; https://01.org/intel-deep-learning-framework.

[14] Nvidia. "Accelerate Machine Learning with the cuDNN Deep Neural Network Library," https://devblogs.nvidia.com/parallelforall/accelerate-machine-learning-cudnn-deep-neural-network-library/.

[15] S. Hadjis, F. Abuzaid, C. Zhang, and C. Re, “Caffe con Troll: Shallow Ideas to Speed Up Deep Learning,” in Proc. of DanaC, 2015.

[16] Y. LeCun, K. Kavukcuoglu, and C. Farabet, “Convolutional networks and applications in vision,” in Proc. of ISCS, 2010.

[17] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, “Convolutional Neural Networks for Speech Recognition,” *IEEE/ACM Trans. on Audio, Speech, and Language Processing,* vol. 22, pp. 10, 2014.

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional Architecture for Fast Feature Embedding,” in Proc. of ACMMM, 2014.

[19] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A MATLAB-like environment for machine learning,” in BigLearn, NIPS Workshop, 2011.

[20] Google. "TensorFlow: an Open Source Software Library for Machine Intelligence," https://www.tensorflow.org.

[21] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems,” in Proc. of LearningSys, 2016.

[22] N. Systems. "Preview release of convolutional kernels," https://github.com/NervanaSystems/nervana-lib-gpu-performance-preview.

[23] Nvidia. "NVIDIA CUDNN GPU Accelerated Deep Learning," https://developer.nvidia.com/cudnn.

[24] S. Chintala. "convnet-benchmarks," https://github.com/soumith/convnet-benchmarks.

[25] Nvidia, *GPU-Based Deep Learning Inference: A Performance and Power Analysis*, 2015.

[26] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” *arXiv*, no. 1404.5997, 2014.

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,” in Proc. of ICLR, 2014.

[28] Intel. https://01.org/rapl-power-meter.

[29] Nvidia. "NVIDIA System Management Interface," https://developer.nvidia.com/nvidia-system-management-interface.

[30] B. Ginsburg. "Caffe-OpenMP," https://github.com/borisgin/caffe/tree/openmp.

[31] N. Wilt, *The cuda handbook: A comprehensive guide to gpu programming*: Pearson Education, 2013.

[32] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong, “Effects of dynamic voltage and frequency scaling on a k20 gpu,” in Proc. of ICPP, 2013.

[33] Y. Taigman, M. Yang, M. A. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in Proc. of CVPR, 2014.

[34] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, “Joint training of a convolutional network and a graphical model for human pose estimation,” in Proc. of NIPS, 2014.

[35] R. Lebret, P. O. Pinheiro, and R. Collobert, “Phrase-based image captioning,” *arXiv*, no. 1502.03671, 2015.

[36] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” in Proc. of ISSCC, 2016.

[37] T. Tang, R. Luo, B. Li, H. Li, Y. Wang, and H. Yang, “Energy efficient spiking neural network design with RRAM devices,” in Proc. of ISIC, 2014.

[38] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, “Backpropagation for energy-efficient neuromorphic computing,” in Proc. of NIPS, 2015.

[39] S. Park, J. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H. Yoo, “An Energy-Efficient and Scalable Deep Learning/Inference Processor With Tetra-Parallel MIMD Architecture for Big Data Applications,” *IEEE Trans. on Biomedical Circuits and Systems*, 2016.