MADONNA: A Framework for **M**easurements and **A**ssistance in designing Low Power **D**eep **N**eural **N**etworks.

Vinu Joseph and Chandrasekhar Nagarajan

***Abstract*— The recent success of Deep Neural Networks (DNNs) in classification tasks has sparked a trend of accelerating their execution with specialized hardware. Since tuned designs easily give an order of magnitude improvement over general-purpose hardware, many architects look beyond an MVP implementation. This project presents Madonna v1.0, a direction towares automated co-design approach across the numerical precision to optimize DNN hardware accelerators. Compared to an 64-bit floating point accelerator baseline, we show that 32-bit floating points accelerators, reduces energy by 1.5; Training time improved by 1.22x and a observable improvement in Inference as well; Across three datasets, these power and energy measurements provide a collective average of 0.5W reduction and 2x energy reduction over an accelerator baseline without almost compromising DNN model accuracy. Madonna enables accurate, low power DNN accelerators , making it feasible to deploy DNNs in power-constrained IoT and mobile devices.**

I. INTRODUCTION

Deep Neural Networks (DNNs) are Machine Learning (ML) methods that learn complex function approximations from input/output examples. These methods have gained popularity over the last 10 years, this gain is attributed in NN literature to emirical achievenments on a wide range of tasks that range from speech recognition, computer vision, and natural language processing. Apps that are based on DNNs can be found across a broad range of computing spectrum, from large high power budget datacenters down to a constrained power budget, battery-powered mobile and IoT deivices. The recent success of DNNs in these problem domains are be attributed to 3 factors

1) The availability of massive datasets
2) Access to highly parallel computational resources such as GPU boards used in this paper.
3) Improvements in the algorithms used to tune DNN's to data.

The parameters (weights) of a DNN are fitted to data in a process called *training*, which typically runs on a high performance CPU/GPU platform. The training process depends on the charecteristics of the available data and should ideally incorporate the latest advances in ML literature and practice, those that are best suited to leverage the high flexibility of software.

During the *prediction* phase, trained DNN models are used to infer unknown outputs from input data that is new. *training* is largely a one-time cost, inference or prediction computations run repeatedly once the DNN is deployed in the production environment.

Therefore speeding up prediction and minimizing its power consumption is highly desirable, especially for apps

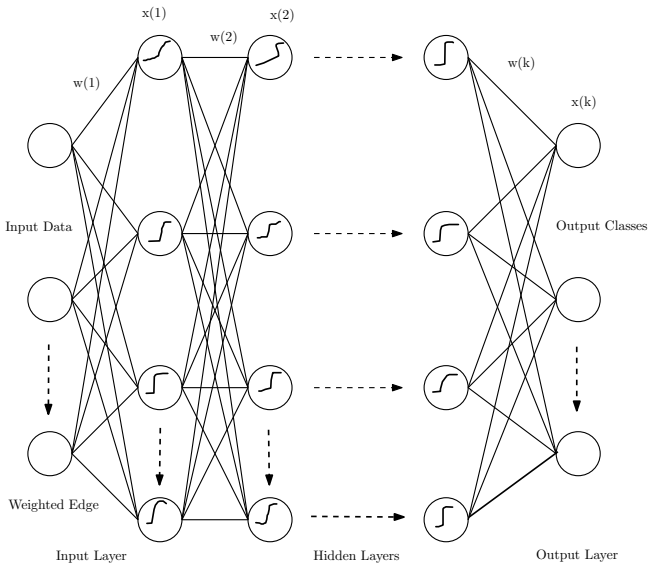

Fig. 1. The operation of a DNN during the prediction phase as a directed acyclic graph

that run on battery-powered devices with strict power budgets and computational capabilities.

One of the solutions to this problem, which we studied in this course, is to design and implement highly customized hardware accelerators for DNN predictions. This project presents a holistic methodology , with an aim to assist the architect of DNN accelerators so that they can achieve minimum power while maintaining high prediction accuracy

MNIST is a widely studied dataset used by the ML community to demostrate state-of-the-art advances in DNN techniques. Figure 6 shows the DNN used in this paper for this data set. As we studied in class there are 2 groups or community that is looking into improving DNN's.

1) ML community, focuses on minimizing prediction error and favours the computational power of GPU's over CPU's.
2) HW commuity, focuses on reducing power consumption and silicon area, moslty at the expense of non-negligible reductions in prediction accuracy.

There is a divergence in the trends observed in the research outcomes of both these teams, the Minerva tool we studied in class, revealed a notable gap for implementations that achieve competitive prediction accuracy with power budgets within the grasp of mobile and IoT platforms. In this project, we present Madonna, a framework to perform energy measurements as the architect makes design decisions, this is moti-

vated by the fact that embedded apps are running with strict power budget and to measure early will provide confidence to the designer that he/she is not making any high power design choices, (explained more in PROPOSAL) Madonna has 4 steps, Launching tuned DNN on the embedded board, Initializing energy measurement setup, Postprocessing to visualize the running time , power and energy profile of the execution and Taking appropriate correcting action to meet the power, runtime budgets for the learning problem. The 3 benchmark we simulated show that there considerable reduction in runtime, power and energy using our framework.

II. BACKGROUND

A. Neural Nets

Neural networks are a biologically inspired machine learning method for a learning complex functions which apply mulpitle nonlinear transformations to an input vector to obtain a corresponding output vector.

B. Deep Neural Nets

Deep Neural networks (DNNs) are defined as networks with one or more hidden layers. The nonlinear tranformations are performed by consecutive layers of fully connected artificial neurons. The first layer is called the input layer, and it has a neuron for each component in the input vector. The last layer is called the output layer and contains one neuron for each component in the output vector. Between input and outout layers, there are additional layers of hidden neurons. The strength of the connection between neurons is determined by a collection of weights whose values are tuned to minimize the prediction error of the network on some training data. Figure(1) shows a DNN, represented as a weighted directed acyclic graph. There are biological metaphors for all these, the are called synapses for edges and neurons for nodes.

C. Training a DNN

A neural network is trained by iteratively adjusting weights to minimize a loss function over labelled data. Training is often performed using stochastic gradient descent (SGD), and the loss function is usually a combination of the prediction error and regularization terms. This process requires hyperparameter values related to the network topology, for example the number of layers and neurons per layer and the configuraion of the SGD procedure, for example the regularization parameters. These hyperparameter values are often tuned by selecting, amoung a grid of candidates, values that minimize the prediction error of the corresponding trained DNN. This can then be used to make predictions on new inputs for which the corresponding outputs are not available.

D. Predictions using a DNN

Each neuron output is obtained by computing a linear combination of the outputs of the neurons in the previous layer and then applying a nonlinear function, where the first layer is fed by the input vector. Equation(3) summarizes this

Algorithm 1 Training a Deep Neural Network

procedure TRAIN
  Initialize all weights $w_{ij}^{(l)}$ at random
  for $t = 0, 1, 2, \cdots, do$ **do**
    Pick $n \in \{1, 2, \cdots, N\}$
    Forward: Compute all $x_j^{(l)}$
    Backward: Compute all $\delta_j^{(l)}$
    Update the weigths: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$
    Iterate to the next step until it is time to stop
  end for
  Return the final weights $w_{ij}^{(l)}$
end procedure

section, each $w_{ij}^{(l)} \in R$ represents the connection strength between the i^{th} neuron in layer $(l-1)$ and j^{th} neuron in the layer (l). The nonlinear function θ allows DNN's to become universal approximators. While many different non linear functions have been proposed, recent research favors the rectifier because of its simplicity and superior empirical performance.

$$w_{ij}^{(l)} = \begin{cases} 1 \le l \le L \text{ layers} \\ 0 \le i \le d^{(l-1)} \text{ inputs} \\ 1 \le j \le d^{(l)} \text{ outputs} \end{cases} \tag{1}$$

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)} \right) \tag{2}$$

$$\theta(s) = tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} \tag{3}$$

$$\text{Apply } \mathbf{x} \text{ to } \mathbf{x_1^{(0)}}, \cdots, x_{d^{(0)}}^{(0)} \rightarrow\rightarrow x_1^{(L)} \quad = h(x) \tag{4}$$

(5)

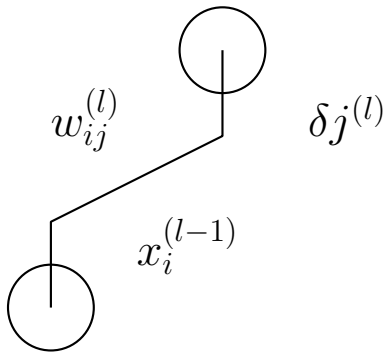

Fig. 2. Backpropagation used to train NN

E. Numeric Precision

1) **Floating point** : IEEE 754 standardizes the floating point implementation, single (32-bit) and double (64-bit) are most common and consistent implementations,

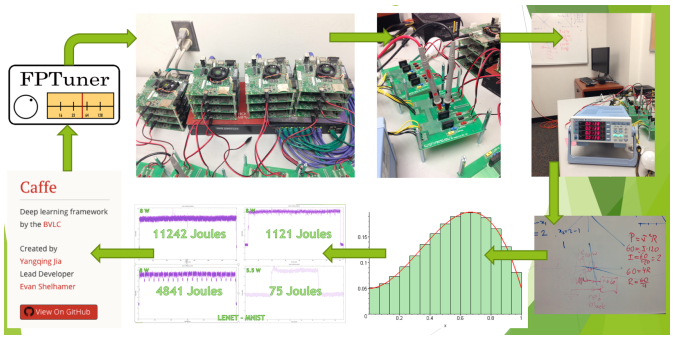

Fig. 3. Workflow of Madonna, training LeNet on MNIST dataset

there are specificaitons for extended precision (80-bit), and other numeric data-types for high precision in software, 128 bit extended precision and arbitrary-precision arithmetic

2) **Fixed point numbers** : The signed and unsigned numbers representation given are fixed point numbers, the binary point is assumed to be at a fixed location say, at the end of the number, for 8 bits, this can represent all integers between -128 and 127

III. PROPOSAL

Neural net architects who work on applications that are compute-intensive, especially on projects like Drones, Autonomous robotic systems, Mobile medical imaging, and Intelligent Video Analytics (IVA) are given strict power and running time budgets. They already have several design parameters to search over, for example which method to use to initialize weights, which activation function to use in which layer, which cost function to use, how to set the paramters like η, epochs mini-batch, λ , how to expand the training data, how to go about regularization choices L1/L2 vs Dropout. On top of these, the fact that they need to tune for power, and numerical precision is very exausting, and the presence of tool in this area for assistance in making optimal choices and performing real energy measurements will be very useful. This is motivated by one of the lectures, we saw in class where we studied about Tools to explore accelerators, like Minerva tool to explore the design space, prune/quantize, lower voltages. Madonna is a proposal in this direction, a tool to explore numerical precision parts of the design space and perform measurements as one designs the network.

IV. EXPERIMENTAL METHODOLOGY

A. Circuit Level

- The nodes are Nvidia Jetson TK-1 boards `http://elinux.org/Jetson_TK1`
- The measurement hardware is a Yokogawa wt310 `http://tmi.yokogawa.com/us/products/digital-power-analyzers/digital-power-analyzers/digital-power-meter-wt300e/`

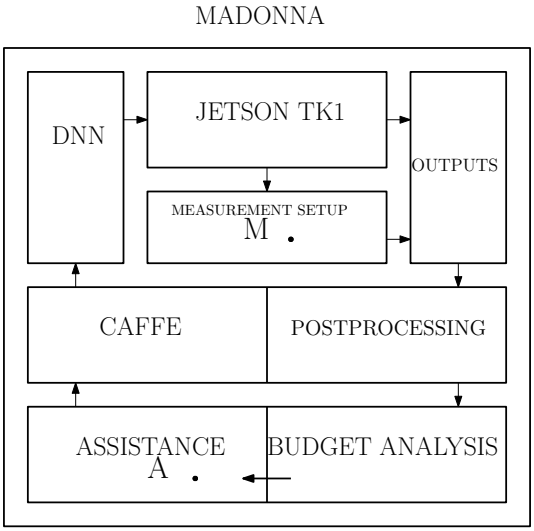

Fig. 4. Block Diagram of our validated Framework

TABLE I
SPECIFICATIONS OF THE JETSON TK1

Tegra K1	
GPU	
NVIDIA Kepler Architecture	192 NVIDIA CUDA Cores
CPU	
CPU Cores and Architecture	Quad Core ARM Cortex-A15
Max Clock Speed	2.3 GHz
Memory	
Memory Type	DDR3L and LPDDR3
Max Memory Size	8 GB with 40-bit address extension
Process	28 nm

B. Topology, Network Level

- From a network prespective:
  The head/gateway node is `mir.cs.utah.edu` (mir) and users can acess it extrnally.
- mir is then connected to the switch on the table. This switch is then connected to the nvidia jetson tk-1 boards (mir01, mir02,.... mir16). To connect to them you must first ssh into mir then ssh into the desired node.
- The power measurement computer, `mirpwr.cs.utah.edu` (mirpwr), is only connected to the University of Utah nework.

C. Electrical perspective

- Components that are directly connected to wall power with no measurement capabilities are mir, mirpwr, the switch, and the yokogawa.
- The two ATX power supplies are connected to the measurement PCB's, two PCSB's for each power supply.
- Each measurement PCB has 4 shunt resistors on them and they power 4 mir nodes.
- 8 nodes are being powered by each ATX power supply. The circuit for each measurement loop is simple, there is a 12V supply ran through the dropper resistor, then the mir node, to ground.

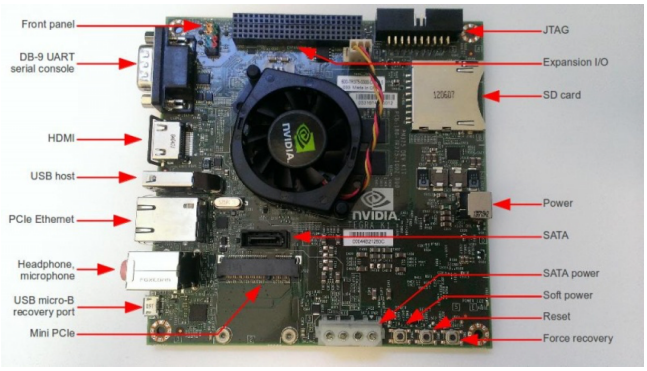

Fig. 5. Schematic of nVIDIA TK1 board , with 192 Kepler GPUs used for DNN training and Inference

D. Framework

Figure(4) shows the block diagram of the MADONNA framework, the Assistance block was studied but is not part of this version 1.0.

E. Software

- Frontend : The frontend for MADOANNA v1.0 is is python, they essentially consists of the steps where the user will setup the tuned DNN for running on the Deep Learning Hardware, and once the Energy numbers are obtained, the user can visualize the effects of his/her design choices, as shown in the Figure(17) and Figure(18)
- Backend : Energy measurerement, trigger-measure-save steps are implemented in C++, and abstracted to the user via API's
- Validation : There are simple unit tests for each block (4)
- Debugging : There is minimal debugging support now, and Hangs and Haults are attributed to fixed, known documented reasons.
- Download : MADONNA 1.0 is Available for Download at GitHub `https://github.com/vinutah/madonna`

F. Data Sets Evaluated

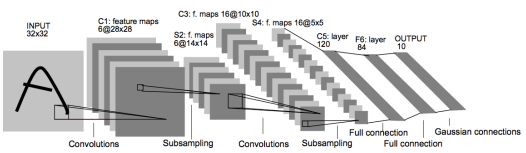

Fig. 6. LeNet on MNIST Data Set

V. RESULTS

The results are summarized in Tables(II) and (III) The visual snapshot of the output of a single madonna pass is as showin in Figure (17) and Figure (18)

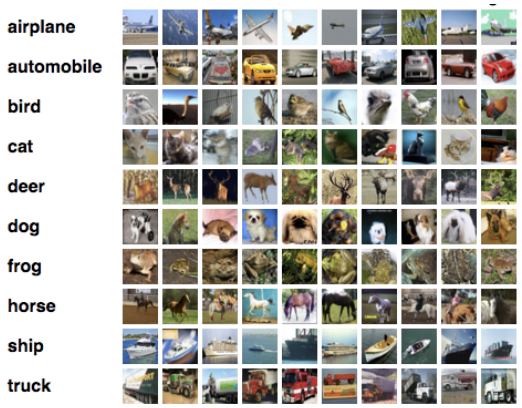

Fig. 7. CudaConvNet on CIFAR-10 Data Set

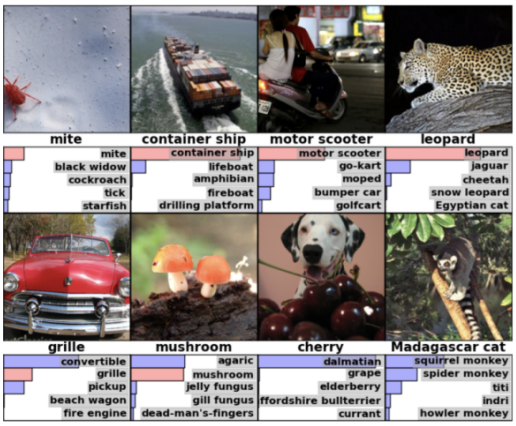

Fig. 8. CaffeReferenceNN on ImageNet Data Set

VI. DISCUSSION

We are adding a brief discussion section here as we spent, considerable time in the initial investigation phase of Madonna, while we were adding support for Assistance in our framework.

A. Ristretto

Ristretto, this tool uses floating point arithmetic to simulate fixed point behavior. All operations actually work on 32-bit floating point numbers. If we want to lower the energy consumption on a GPU, we will have to convert Ristretto weights to fixed point and implement fixed point forward propagation in CUDA, is planned to be part of our future releases of Madonna.

B. FPTuner

FPTuner is a reserach tool for rigorous floating point tuning. Adding support for 16-bit floating point, which is currently lacks, is quite easy, it involves changing the wrapper scripts to FPTuner, and not its core search + optimization logic. Reason for not taking this direction is that, we know the input space quite well so all the low-precision literature on CNN, use fixed 16-bit and not floating format for 16-bit (C/C++ does not have native support for half precision,

TABLE II
IEEE-754 32-BIT SINGLE PRECISION FLOATING POINT REPRESENETATIONS

DATA SET	SINGLE				
	TRAINING		TESTING		
	POWER	ENERGY	POWER	ENERGY	ACC
MNIST	7.5 W	4492 J	6.5W	35 J	97.5%
CIFAR-10	9.5 W	11662 J	10 W	62 J	73.3%
IMAGENET	Caffe Model Zoo		13 J	722 J	–

TABLE III
IEEE-754 64-BIT DOUBLE PRECISION FLOATING POINT REPRESENETATIONS

DATA SET	DOUBLE				
	TRAINING		TESTING		
	POWER	ENERGY	POWER	ENERGY	ACC
MNIST	8 W	10828 J	8.0 W	50 J	98.8%
CIFAR-10	10 W	31211 J	10.5 W	65 J	74.3%
IMAGENET	Caffe Model Zoo		13.5 J	750 J	–

will need to use external libraries like MPFR) . In addition to this the authors of FPTuner suggested that FPTuner (in its fully developed* version) is well suited for application that have strict restriction on precision and error bounds and deep CNNs training/classification are not such codes. The other direction was towards Improving its speed for precision allocation of medium/large expressions, the search time depends on three Externals tools These tools are Linear programming solver (Gurobi) , a Global optimizer (Gelphia) and A Tool for Rigorous Estimation of Round-off Floating-point Errors (FPTaylor) We did not want to look at these tools for this class project, as the focus will be far away from Class goals.

VII. RELATED WORK

A. *Minerva*

Minerva was published as Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators in the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture, ISCA 2016. this work presented , a highly automated co-design approach across the algorithm, architecture, and circuit levels to optimize DNN hardware accelerators. Compared to an established fixed-point accelerator baseline, the authors showed that fine-grained, heterogeneous data type optimization reduces power by 1.5x;, aggressive, in-line predication and pruning of small activity values further reduces power by 2.0x;, and active hardware fault detection coupled with domain-aware error mitigation eliminates an additional 2.7x; through lowering SRAM voltages. Across five datasets, these optimizations provide a collective average of 8.1x; power reduction over an accelerator baseline without compromising DNN model accuracy. Minerva enables highly accurate, ultra-low power DNN accelerators (in the range of tens of milliwatts), making it feasible to deploy DNNs in power-constrained IoT and mobile devices.

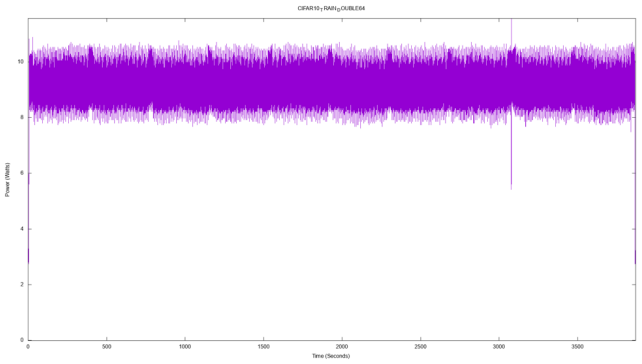

Fig. 9. Training Cuda-Convnet on CIFAR-10 dataset, in double precision

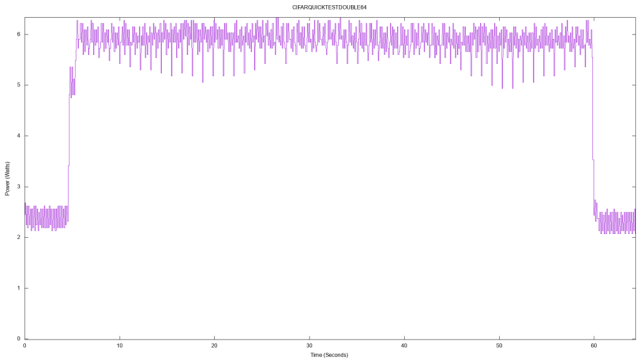

Fig. 10. Testing Cuda-Convnet on CIFAR-10 dataset, in double precision

VIII. CONCLUSIONS

In this project we have considered the problem of creating a framework that will be used for designing and building optimized hardware accelerators for deep neural networks that achieve minimal power consumption while maintaining high prediction accuracy. Madonna is inspired by Minerva which was a a holistic, highly automated co-design flow that combines insights and techniques across the algorithm, architecture, and circuit levels, enabling low-power accelerators for highly accurate DNN prediction. We definitely conclude that the high precision is not the way to go for power or performace optimization, and low precision substantially improves the power efficiency of DNN hardware accelerators. Madonna in its fully developed state will be able to assist in the overall power consumption across diverse ML datasets without impacting prediction error. Madonna makes it possible to deploy DNNs as a solution in power-constrained mobile environments.

REFERENCES

[1] Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators Brandon Reagen Paul Whatmough Robert Adolf Saketh Rama Hyunkwang Lee Sae Kyu Lee Jos Miguel Hernndez-Lobato Gu-Yeon Wei David Brooks Harvard University, 2016
[2] Ristretto: Hardware-Oriented Approximation of Convolutional Neural Networks By Philipp Matthias Gysel B.S. (Bern University of Applied Sciences, Switzerland) 2012 Thesis Submitted in partial satisfaction of the requirements for the degree of Master of Science
[3] FPTuner: `https://github.com/soarlab/FPTuner`
[4] FPTaylor: `https://github.com/soarlab/FPTaylor`

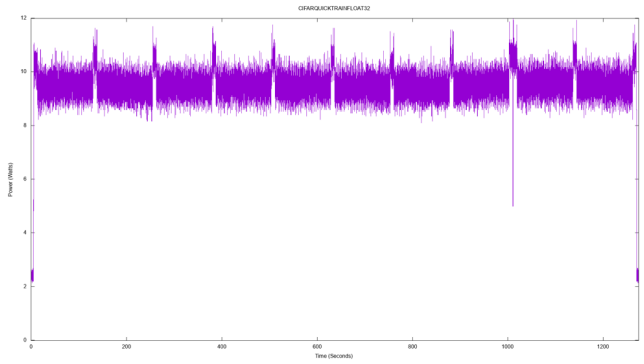

Fig. 11. Testing Cuda-Convnet on CIFAR-10 dataset, in single precision

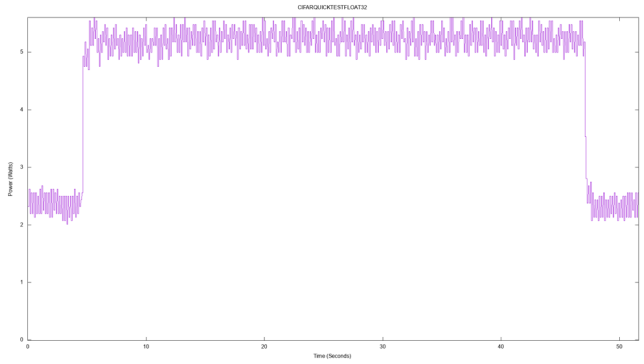

Fig. 12. Testing Cuda-Convnet on CIFAR-10 dataset, in single precision

[5] Caffe: Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor http://caffe.berkeleyvision.org/

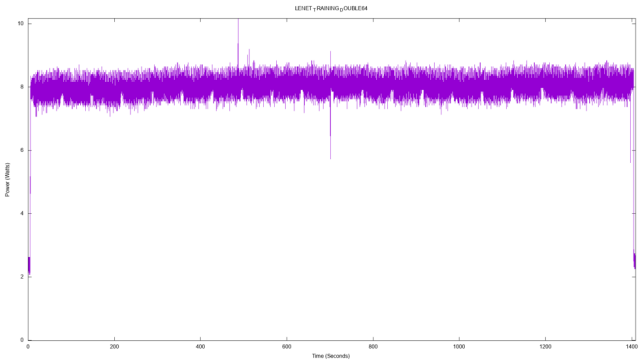

Fig. 13. Training LeNet on MNIST dataset, in double precision

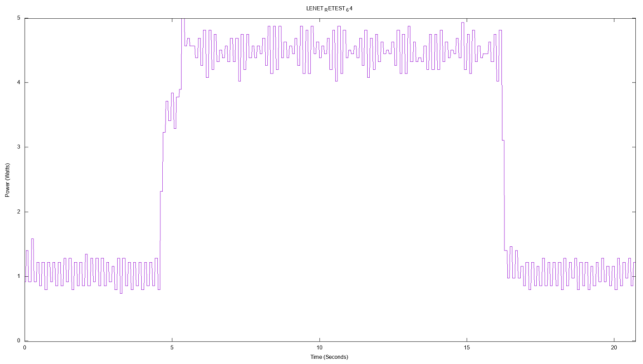

Fig. 14. Testing LeNet on MNIST dataset, in double precision

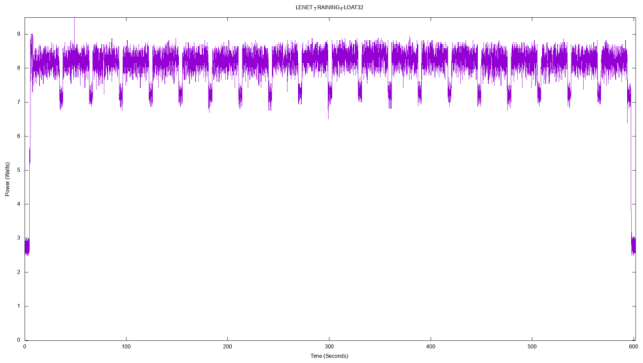

Fig. 15. Testing LeNet on MNIST dataset, in single precision

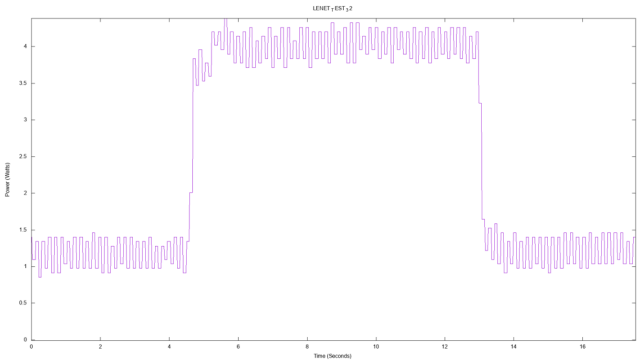

Fig. 16. Testing LeNet on MNIST dataset, in single precision

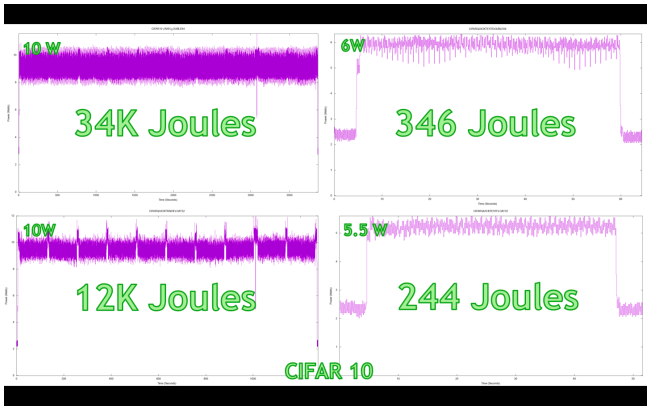

Fig. 17. A sample Madonna Pass, analyzing Cuda-Convnet on CIFAR-10 dataset

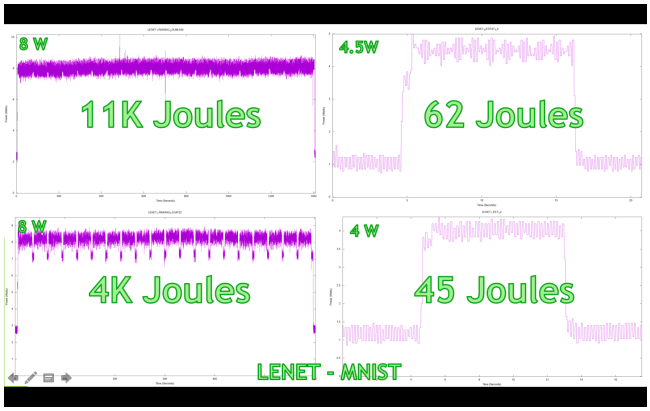

Fig. 18. A sample Madonna Pass, analyzing LeNet on MNIST dataset