Coordinated DVFS and Precision Control for Deep Neural Networks

Seyed Morteza Nabavinejad, Hassan Hafez-Kolahi, and Sherief Reda, *Senior Member, IEEE*

***Abstract*—Traditionally, DVFS has been the main mechanism to trade-off performance and power. We observe that Deep Neural Network (DNN) applications offer the possibility to trade-off performance, power, and accuracy using both DVFS and numerical precision levels. Our proposed approach, Power-Inference accuracy Trading (PIT), monitors the server's load, and accordingly adjusts the precision of the DNN model and the DVFS setting of GPU to trade-off the accuracy and power consumption with response time. At high loads and tight request arrivals, PIT leverages INT8-precision instructions of GPU to dynamically change the precision of deployed DNN models and boosts GPU frequency to execute the requests faster at the expense of accuracy reduction and high power consumption. However, when the requests' arrival rate is relaxed and there is slack time for requests, PIT deploys high precision version of models to improve the accuracy and reduces GPU frequency to decrease power consumption. We implement and deploy PIT on a state-of-the-art server equipped with a Tesla P40 GPU. Experimental results demonstrate that depending on the load, PIT can improve response time up to 11% compared to a job scheduler that uses only FP32 precision. It also improves the energy consumption by up to 28%, while achieving around 99.5% accuracy of sole FP32-precision.**

***Index Terms*—Deep Neural Network, Hardware Accelerator, Power, Accuracy, Response Time**

I. Introduction

DEEP Neural Networks (DNNs) provide state-of-the-art results in various areas such as computer vision [1] and speech recognition [2]. Coupled with efficient GPGPU hardware and large datasets, these modern neural networks use very large and deep architectures to achieve state-of-the-art results. These modern DNNs require enormous computational resources for both training and inference.

To address the resource complexity, hardware manufactures such as Nvidia and AMD offer reduced-precision arithmetic, which requires less resources (memory, computation, and power), together with software libraries. They can convert the models trained using higher precision arithmetic to reduced-precision ones, while trying to preserve the accuracy of the model [3]. This approach has been also used in Google's Tensor Processing Unit (TPU) [4]. It is suitable for ML services that are interested in faster and more resource-efficient approaches to improve their resource utilization and customer satisfaction. Previous works try to improve the resource utilization, performance, and energy efficiency of ML inference applications via resource autoscaling or designing new hardware configurable units [5], [6].

In this work, we aim to leverage the GPU computing units that support reduced-precision and the various frequency levels of GPU that lead to different amount of power consumption to improve the balance between the latency of DNN inference, power consumption, and classification accuracy. We observe that in addition to GPU frequency, the GPU precision itself also affects the power consumption. For example, the power consumption of INT8-precision is less than the FP32-precision. Our proposed approach, *Power-Inference accuracy Trading (PIT)*, monitors the system's load, and accordingly adjusts the precision and DVFS depending on the expected response time and accuracy. Our contributions are as follows:

- We devise a runtime controller that monitors inference time targets and accordingly adjusts DVFS and the DNN model precision, leveraging the ability of modern GPUs that support several precision arithmetic, in order to improve inference time and energy consumption, while keeping the accuracy as high as possible.
- It is commonplace to use the slack time to decrease DVFS, and consequently, power consumption. However, our PIT controller trades the slack time of requests with inference accuracy, in addition to power consumption, considering the status of requests in the queue and server's load.
- Using a high-end server, we implement and evaluate *PIT* on a state-of-the-art DNN network architecture, ResNet, and demanding test set, ImageNet. We compare our solution against other standard scheduling techniques and quantify the improvement in response time and power consumption of our methodology.

The organization of the rest of the paper is as follows. In Section I-A, we motivate the problem. Section II describes our proposed methodology. Next, in Section III, we provide the main results obtained from evaluating our methodology. Finally, in Section V we provide a summary of our main conclusions.

A. Motivation

To motivate our methodology, we use ResNet [7], to classify 50000 images from ImageNet dataset [8]. We use TensorRT to generate two versions with different precision, float 32 (FP32) and integer 8 (INT8), of the native model, where the *native precision* is the original precision used to train the model, which is double precision, i.e., FP64. Then, we deploy the models on GPU for inference of the aforementioned images. The inference time of images and power and energy consumption under different precision for various GPU frequencies is presented in Fig. 1. For inference accuracy, both top-1 and top-5 labels are considered. Since the accuracy is not affected by GPU frequency, only one value per GPU precision is reported. As can be seen from Fig. 1, changing the precision affects the runtime, power, and energy significantly. Moreover, the various GPU frequency levels also affect the aforementioned parameters dramatically. However, the precision has negligible effect on the final accuracy of classification. Considering the results illustrated in Fig. 1, *we conclude that different precision arithmetic, as well as GPU frequencies, can be used as control knobs to improve the power/energy consumption, while maintaining the response time, by sacrificing the accuracy as little as possible.*

II. Methodology

A. System Architecture

We present the overall system architecture in Fig. 2, where users submit their requests to the DNN Inference service. This scenario corresponds to system architecture of ML as a service platform

S.M. Nabavinejad is with School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Iran.E-mail: nabavinejad@ipm.ir

H. Hafez-Kolahi is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.E-mail: hafez@ce.sharif.edu

S. Reda is with the School of Engineering, Brown University, Providence, RI.E-mail: sherief_reda@brown.edu

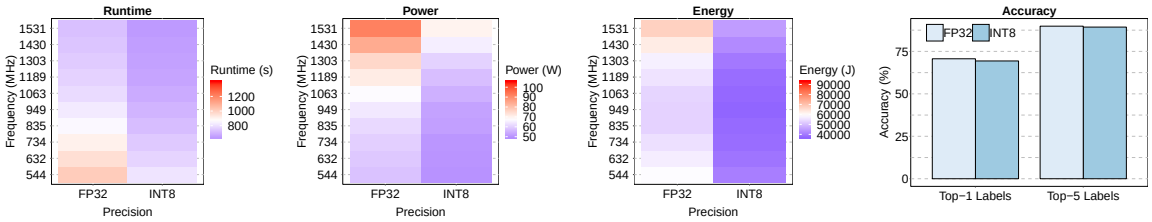

Fig. 1. Inference time, power, energy, and accuracy using different precision and GPU frequency.

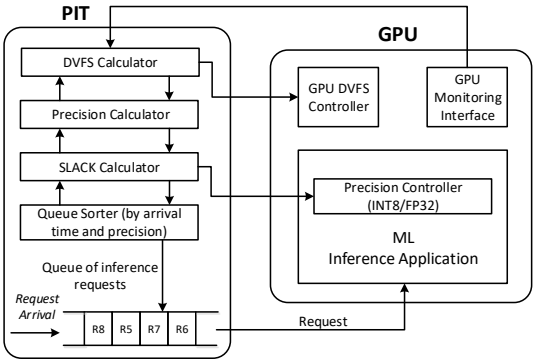

Fig. 2. Proposed overall flow of system architecture

(MLaaS) [5]. Each request includes a batch of images that should be processed and an expected response time as required by the service-level agreement. In our work, we consider the expected response time of request as the estimated runtime of the request under FP32-precision at highest DVFS plus a slack time that is equal to estimated runtime. But it can be considered differently for other scenarios and our proposed approach can handle them as well.

Upon submission of a request, the system assigns a time stamp to each request to save its submission time and stores it in the queue. The response time window of a request starts from its submission time. Based on previous profiling, or online profiling on a subset of images for new models, the system administrator can estimate the average execution time of a request for different precision (FP32, INT8) under highest GPU frequency. Having this estimation and status of the queue, our *PIT* controller decides on the precision of each request and its scheduling. This initial estimation helps *PIT* decides the precision that should be used for the request, since the precision cannot be changed while the request is being executed by GPU. During runtime of a request, *PIT* constantly monitors the power consumption of GPU and determines its DVFS based on the elapsed time of the request and its expected response time. The objective is to maintain the expected response time and decrease the power consumption, while keeping the accuracy of results as high as possible.

B. Power-Inference accuracy Trading (PIT)

The pseudo-code of *PIT* is presented in Algorithm 1 with summary of acronyms in Table I. At the first step, *PIT* updates the time each request has been spending in queue (Line 4). Since *PIT* aims to keep the accuracy as high as possible, first it estimates the runtime of requests assuming they are going to be executed using FP32-precision. Then, it calculates the SLACK parameter, which is the time left for the request before reaching the expected response time divided by the estimated runtime (ETR), i.e., $(ExRT - TQ)/ETR$, and sets their precision as FP32 (lines 5 to 7). After that, it checks to make sure that all the requests can meet their *ExRT* ($SLACK \geq 1$) with FP32-precision in current state of queue. If so, it sorts the requests according to their *SLACK* and selects the one with lowest value (i.e., the request with lowest slack time), schedules it for execution on the GPU, calls the coordinatedDVFS routine that dynamically sets the GPU frequency, and waits for it to finish before choosing the next request (lines 8 to 11).

If there are requests with $SLACK < 1$, *PIT* tries to increase the value of their *SLACK* by employing more aggressive approximation, i.e., INT8-precision. Accordingly, it updates the requests with *SLACK* less than one (lines 15 to 18). Further approximations after INT8-precision are not possible. Hence, *PIT* only sorts the requests based on their current *SLACK* and sends the one at the head of the queue to the GPU for execution.

After starting the execution of a request on GPU, *PIT* coordinates GPU frequency by *coordinatedDVFS* routine. In this stage, *PIT* considers the expected execution time of the request, which is the difference between its expected response time (*ExRT*) and time spent in queue (*TQ*). *PIT* starts from lowest possible level of frequency (for the sake of power consumption) and processes a predefined number of batch of images and save their execution time before changing the frequency. Then, it calculates the average execution time of *K* previous batches (*K=10* in our experiments) and estimates the time needed for processing the remaining batches with current pace. *PIT* considers the average of *K* previous batches to avoid excessive change of GPU frequency. Based on the sum of elapsed time from request submission and estimated runtime of remaining batches, it decides to either increase the frequency to the next level or decrease it, provided that higher or lower frequency is available.

TABLE I
LIST OF THE PARAMETERS USED IN THE PAPER.

Parameter	Description
NI	Number of Images
ExRT	Expected response time for a request
TQ	Time in Queue for a request
ETR	Estimated Runtime of a request
SLACK	(ExRT-TQ)/ETR
ER_{FP32}	Estimated runtime for one image using FP32-precision
ER_{INT8}	Estimated runtime for one image using INT8-precision

Algorithm 1 Power-Inference accuracy Trading

Input: N, ExRT, NI, ER_{FP32}, ER_{INT8}, TQ
Output: Sorted N with determined precision for each request

1: N: Set of requests in queue
2: **while** N is not empty **do**
3: **for** $i \in N$ **do**
4: Update(TQ_i)
5: $ETR_i = NI_i \times ER_{FP32}$
6: $SLACK_i = (ExRT_i - TQ_i)/ETR_i$
7: Set the precision of request as FP32
8: **if** $\forall i \in N, SLACK_i \geq 1$ **then**
9: Sort N in ascending order with respect to $SLACK$
10: Send N[1] to GPU with FP32-precision
11: coordinatedDVFS($ExRT_i - TQ_i$)
12: Go to 2
13: **else** //$SLACK < 1$ for some requests
14: **for** $i \in N$ **do**
15: **if** $SLACK_i < 1$ **then**
16: $ETR_i = NI_i \times ER_{INT8}$
17: $SLACK_i = (ExRT_i - TQ_i)/ETR_i$
18: Set the precision of request as INT8
19: Sort N in ascending order with respect to $SLACK$
20: Send N[1] to GPU with its determined precision (FP32 or INT8)
21: coordinatedDVFS($ExRT_i - TQ_i$)
22: Go to 2

TABLE II
RESULTS STATISTICS FOR 10 WORKLOADS

	Response Time (s)			Accuracy (%) (Top-1, Top-5)			Power (W)			Energy (J)		
	SLURM	SLURM_DVFS	PIT	SLURM	SLURM_DVFS	PIT	SLURM	SLURM_DVFS	PIT	SLURM	SLURM_DVFS	PIT
Average	1356.7	1762.6	1751.0	(70.65, 89.86)	(70.65, 89.86)	(70.17, 89.66)	92.12	71.09	62.69	68230.8	61639.4	54093.1
Min	901.0	826.2	1358.4	(70.65, 89.86)	(70.65, 89.86)	(69.75, 89.48)	91.65	61.21	57.51	67902.8	58270.0	48751.3
Max	1872.8	2414.5	2194.2	(70.65, 89.86)	(70.65, 89.86)	(70.52, 89.81)	92.65	81.80	66.24	68768.6	64837.7	58299.8
STD	361.5	491.2	287.5	0.00	0.00	(0.23, 0.10)	0.30	6.63	2.58	250.0	2248.0	2894.6

III. EXPERIMENTAL RESULTS

A. Experimental Setup

We run our experiments on a dual-socket Xeon server where each of the E5-2680 v4 Xeon chips has 28 cores running at 2.4 GHz. The server has 128 GB of DDR4 memory. Ubuntu 16.04 with kernel 4.4 is installed on the server with the python 2.7, CUDA 9.0, TensorFlow 1.10, and TensorRT 4.0. The server is equipped with a Nvidia Tesla P40 GPU Accelerator. The Tesla P40 leverages Nvidia Pascal architecture and has 3840 CUDA cores with 24 GB GDDR5 memory. It supports INT8 arithmetic which is optimized for deep learning inference [9]. The GPU has 79 DVFS levels, starting from 544 MHz to 1531 MHz. Levels are very close to each other and moving from one level to another has negligible effect on GPU performance and power consumption. So, we chose 10 levels that have more distance from each other, and consequently, their effect is more significant: 544 MHz, 632 MHz, 734 MHz, 835 MHz, 949 MHz, 1063 MHz, 1189 MHz, 1303 MHz, 1430 MHz, and 1531 MHz.

Comparisons. Since there is no direct prior work similar to *PIT*, we contrast *PIT*'s performance to SLURM [10] and a modified versions of it that is DVFS aware. We compare *PIT* against SLURM as a baseline scheduler to show the advantage of our techniques compared to traditional schedulers. SLURM is a well-recognized cluster management and job scheduling system that uses a best fit algorithm that schedules the requests according to a defined priority to minimize their response time. However, SLURM does not leverage approximations by default. In SLURM, the priority of each request is defined as the weighted sum of its age (time in queue, TQ), size (number of images, NI) and expected response time (*ExRT*). We implement two versions of it: 1) *SLURM* that executes all the requests with FP32-precision, and does not leverage DVFS and 2) *SLURM_DVFS* is similar to SLURM, but leverages DVFS to adjusts the GPU frequency. It uses the same routine as *PIT* (coordinatedDVFS) for coordinating GPU frequency.

B. Results

We provide our results on workloads that calculate inference for batch of images from ImageNet dataset [8], using the ResNet-v2-152 [7] network. For the first set of experiments, we have 10 workloads, each of which has 10 inference requests with various arrival times. Since the arrival times of requests are generated randomly, we have considered 10 workloads to provide statistically conclusive results. The arrival times of requests have exponential distribution with a mean parameter (μ) proportional to estimated response time of the requests in the workload. Our model is similar to arrival time model considered for jobs in the 3Sigma model by Park *et al.* [11]. We estimate the runtime of each request by multiplying the number of images of a request by the inference time of FP32 model for one image. The *ExRT* is an input to *PIT* which system administrator can configure as desired. In our experiments, the *ExRT* of each request is the sum of estimated runtime (by FP32-precision) plus a slack time that is equal to estimated runtime (i.e., $ExRT = 2\times$ estimated runtime). However, *PIT* works with any other definition for expected response time as well.

To *calculate the response time* of a request, we add its time spending in queue (*TQ*) to its runtime. To *calculate the accuracy*, we compare the labels assigned to each image by each precision against original ones provided by dataset. Finally, we use NVIDIA System Management Interface (nvidia-smi) to coordinate the GPU frequency and measure its power consumption.

The results are presented in Table II. Since SLURM does not leverage DVFS, it uses the highest possible GPU frequency (i.e., 1531 MHz) for most of the time. Hence, we see that SLURM has relatively better response time than *PIT*. However, this high GPU frequency of SLURM leads to excessive power and energy consumption. Even in workloads where response time of SLURM is higher than *PIT*, its power/energy consumption is higher as well. In other words, SLURM always sacrifices power consumption to obtain better response time. The cost of reaching lower response time for SLURM is too much. It increases power by around 46% (on average) compared to *PIT*, to decrease the response time by 22% (on average). *PIT*, unlike SLURM, tries to trade-off between response time and power consumption, and hence, always obtains the lowest possible power/energy consumption, while maintaining the response time.

To provide more details, we present the distribution of response time of requests in Fig. 3. The results belong to all the 10 workloads. However, we have normalized the response time of requests in each workload to the maximum value of it over all the approaches in the same workload. As expected, *PIT*'s performance stands between SLURM and SLURM_DVFS, so its response time is also less than SLURM_DVFS and relatively more than SLURM. The precision selection and DVFS coordination under *PIT* for one of the workloads is shown in Fig. 4. As can bee seen, *PIT* determines different precision for each request and dynamically changes the GPU frequency to achieve its objectives, i.e., maintaining the response time and decreasing power consumption.

C. PIT Sensitivity Analysis

For sensitivity analysis, we consider eight sensitivity traces (ST), each of which with 10 inference requests, and change the arrival rate of requests from tight (ST-1) to relaxed (ST-8). We change the value of μ of exponential distribution in the random number generator from 2000 (ST-1) to 6000 (ST-8) to adjust the tightness of arrival times.

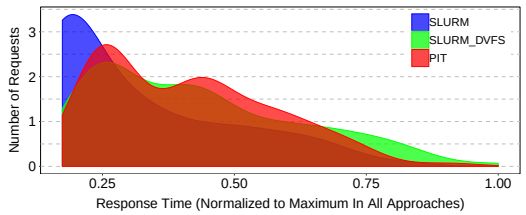

Fig. 3. Distribution of response time of requests under different approaches.

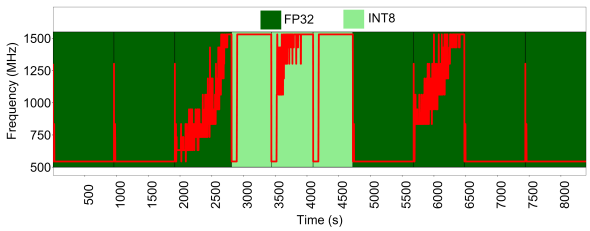

Fig. 4. Dynamic behaviour of PIT regarding model precision and GPU frequency for one of the workloads. Background color indicates the model precision and the red line shows the GPU frequency

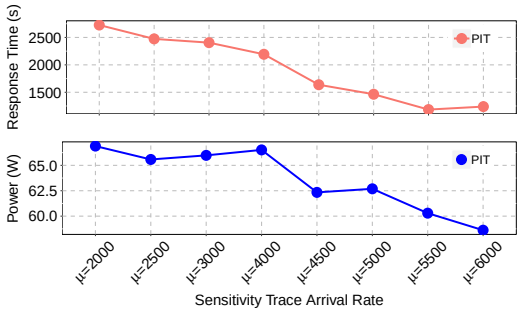

Fig. 5. Response time and power consumption of PIT for different sensitivity traces (average of all the requests in each trace)

TABLE III
NUMBER OF EACH MODEL PRECISION SELECTED BY *PIT* AND OBTAINED ACCURACY PER SENSITIVITY TRACE.

	Arrival Rate μ	FP32	INT8	Accuracy (Top-1, Top-5)
Tight arrival rate	2000	2	8	(69.626%, 89.429%)
	2500	3	7	(69.754%, 89.483%)
	3000	2	8	(69.626%, 89.429%)
	4000	3	7	(69.754%, 89.483%)
	4500	7	3	(70.266%, 89.698%)
	5000	8	2	(70.394%, 89.752%)
	5500	10	0	(70.65%, 89.86%)
Relaxed arrival rate	6000	10	0	(70.65%, 89.86%)

Fig. 5 shows the average response time and power consumption of *PIT* for each trace. As can be seen, the response time decreases when the arrival rate is relaxed. This result indicates that *PIT* is capable of leveraging the inter-arrival time of requests to reduce the response time. It also reduces the power consumption by using lower GPU frequencies, when the inter-arrival time of requests is relaxed.

The accuracy of *PIT* and number of each model precision in each trace is presented in Table III. When the arrival rate is relaxed, *PIT* exploits this opportunity to improve the accuracy, while trying to maintain the response time. Thus, we see that for relaxed traces *PIT* tends to use more precise models, and hence, obtains more accurate results.

D. Discussion

To show the relationship between precision and DVFS, and precision and accuracy, we employed eight additional image classification networks to conduct more experiments. The results are presented in Table IV. The computational complexity column shows the amount of instructions (in Mega FLOPS) required for inference of one image. Results indicate that as the computational complexity increases, the effect of reduced precision on accuracy also increases. The maximum difference between FP32 accuracy and INT8 accuracy is around 1% and 0.5% for Top-1 and Top-5 labels, respectively. We can conclude that effect of INT8-precision depends on the size and computational complexity of network, and it can provide results as accurate as 99% of FP32, or even closer. The results in Table IV also shows the difference between power consumption of networks for highest DVFS (1531 MHz) and lowest one (544 MHz) under two different precision. We see that for less computationally complex networks, e.g., MobileNet-V1, precision has negligible effect on the power consumption of different DVFS levels. However, for large and complex networks, e.g., ResNet-V2-152, changing from FP32 to INT8 significantly reduces the gap between lowest and highest DVFS level, and hence, we can conclude that impact of DVFS on power consumption can be mitigated using lower precision.

IV. Related Work

To show our contributions in this work, we discuss the difference between our approach and previous ones concerning trading off DNN computation requirement with quantization. A large body of previous approaches that leverage quantization, focus on training phase and apply quantization while training the network [12], [13]. For having different precision, and consequently, different accuracy, power consumption, and runtime, the networks need to be retrained using new parameters. It imposes significant overhead, and in some cases such as ML as a service, it is almost impossible to ask users to retrain their models with new parameters and resubmit them. Unlike these approaches, *PIT* can use quantization dynamically during the inference phase. So, the model is only needed to be trained once, with highest possible precision. Other approaches such as [6], [14] consider the lower precision for inference phase. However, they do not consider the various DVFS levels provided by hardware accelerators such as GPUs. Moreover, They consider one job at a time and tune the precision and accuracy for that single job. But, *PIT* considers a queue of jobs and decides on their precision according to the state of other jobs in the queue.

TABLE IV
EFFECT OF PRECISION ON ACCURACY AND POWER CONSUMPTION OF DIFFERENT NETWORKS WITH RESPECT TO DVFS AND COMPUTATIONAL COMPLEXITY OF EACH NETWORK.

	Top-1 Accuracy (%)		Top-5 Accuracy (%)		Difference Between Power of Min and Max DVFS (%)		Computational Complexity (M-FLOPS)
	FP32	INT8	FP32	INT8	FP32	INT8	
MobNet-V1-025	38.56	38.58	63.14	63.15	19.28	19.52	0.93
MobNet-V1-05	56.51	56.51	79.62	79.63	20.24	20.09	2.64
MobNet-V1-1	69.38	69.37	88.92	88.93	23.44	23.10	8.42
Inception-V1	68.20	67.72	88.49	88.24	23.92	21.45	13.22
Inception-V2	72.64	72.25	90.95	90.79	26.05	22.85	22.34
Inception-V3	77.22	77.22	93.44	93.54	32.40	24.98	54.25
ResNet-V2-50	67.78	66.16	87.97	87.41	29.70	23.46	51.01
ResNet-V2-101	70.08	69.39	89.33	89.06	33.78	26.24	88.89
ResNet-V2-152	70.65	69.37	89.86	89.40	38.21	27.48	120.08

V. Conclusion and Future Work

Our proposed approach, PIT, aims to trade-off response time with accuracy and power consumption by employing approximation, scheduling, and DVFS. PIT leverages INT8-precision instructions, that is supported by cutting-edge GPUs, to implement inference approximation. Experiments show that PIT can significantly decrease power consumption compared to a common cluster management and scheduling system, while reducing the accuracy slightly and considering response time.

In future works, we aim to employ GPUs equipped with INT16, so we can evaluate the efficacy of this precision as well. *PIT* needs the final frozen graph of a network, obtained from training, to generate the reduced-precision versions for inference. So, currently, we cannot use it for training. Modifying PIT to support training phase can be an interesting direction for future works. In this work, our focus was on the image classification networks, so we have studied several networks proposed for this application (see Table IV). However, studying other applications such as speech recognition and handwriting recognition, which have different properties such as different layers, is another direction for future works.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR'15*, 2015, pp. 1–9.
[2] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *ICASSP'13*. IEEE, 2013, pp. 8614–8618.
[3] S. Migacz, "8-bit inference with tensorrt," in *GPU Tech. Conf.*, 2017.
[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA'17*, pp. 1–12.

[5] A. Gujarati, S. Elnikety, Y. He, K. S. McKinley, and B. B. Brandenburg, "Swayam: Distributed autoscaling to meet slas of machine learning inference services with resource efficiency," in *Middleware '17*, 2017, pp. 109–120.
[6] M. Imani, M. Masich, D. Peroni, P. Wang, and T. Rosing, "Canna: Neural network acceleration using configurable approximation on gpgpu," in *ASP-DAC'18*, 2018, pp. 682–689.
[7] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV'16*, 2016, pp. 630–645.
[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
[9] NVIDIA. (2016, Nov.) Tesla p40 gpu accelerator. [Online]. Available: http://images.nvidia.com/content/pdf/tesla/Tesla-P40-Product-Brief.pdf
[10] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.
[11] J. W. Park, A. Tumanov, A. Jiang, M. A. Kozuch, and G. R. Ganger, "3sigma: distribution-based cluster scheduling for runtime uncertainty," in *EuroSys'18*. ACM, 2018, p. 2.
[12] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," in *CVPR'17*, 2017, pp. 5918–5926.
[13] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.
[14] S. Koteshwara and K. K. Parhi, "Incremental-precision based feature computation and multi-level classification for low-energy internet-of-things," *IEEE J. on Emerg. and Sel. Topics in Circuits and Systems*, 2018.