Hardware-Aware Machine Learning: Modeling and Optimization

(*Invited paper*)

Diana Marculescu, Dimitrios Stamoulis, Ermao Cai
Department of ECE, Carnegie Mellon University, Pittsburgh, PA
Email: dianam@cmu.edu, dstamoul@andrew.cmu.edu, ermao@cmu.edu

ABSTRACT

Recent breakthroughs in Machine Learning (ML) applications, and especially in Deep Learning (DL), have made DL models a key component in almost every modern computing system. The increased popularity of DL applications deployed on a wide-spectrum of platforms (from mobile devices to datacenters) have resulted in a plethora of design challenges related to the constraints introduced by the hardware itself. "What is the latency or energy cost for an inference made by a Deep Neural Network (DNN)?" "Is it possible to predict this latency or energy consumption before a model is even trained?" "If yes, how can machine learners take advantage of these models to design the hardware-optimal DNN for deployment?" From lengthening battery life of mobile devices to reducing the runtime requirements of DL models executing in the cloud, the answers to these questions have drawn significant attention.

One cannot optimize what isn't properly modeled. Therefore, it is important to understand the hardware efficiency of DL models during serving for making an inference, before even training the model. This key observation has motivated the use of predictive models to capture the hardware performance or energy efficiency of ML applications. Furthermore, ML practitioners are currently challenged with the task of designing the DNN model, *i.e.*, of tuning the hyper-parameters of the DNN architecture, while optimizing for *both* accuracy of the DL model and its hardware efficiency. Therefore, state-of-the-art methodologies have proposed *hardware-aware* hyper-parameter optimization techniques. In this paper, we provide a comprehensive assessment of state-of-the-art work and selected results on the hardware-aware modeling and optimization for ML applications. We also highlight several open questions that are poised to give rise to novel hardware-aware designs in the next few years, as DL applications continue to significantly impact associated hardware systems and platforms.

1 INTRODUCTION

Recent advances in Deep Learning (DL) have enabled state-of-the-art results in numerous areas, such as text processing and computer vision. These breakthroughs in Deep Neural Networks (DNN) have been fueled by newly discovered ML model configurations [2] and advances in hardware platforms. However, the demand for better performance in real-world deployment results in increased complexity for both the hardware systems and the DL models. As modern DNNs grow deeper and more complex, hardware constraints emerge as a key limiting factor. This "hardware wall" manifests its unintended, negative effects in several ways.

First, the energy requirements of DNNs have emerged as a key impediment preventing their deployment on energy-constrained embedded and mobile devices, such as Internet-of-Things (IoT) nodes and wearables. For instance, image classification with AlexNet [26] can drain the smartphone battery within an hour [38]. This design challenge has resulted in a plethora of recent methodologies that focus on developing energy-efficient image classification solutions [31]. However, identifying the most energy-efficient implementation on a given platform or the right DNN model-hardware platform pair can be challenging. Recent work shows that, while several DNN architectures can achieve a similar accuracy level [3], the energy consumption differs drastically among these various iso-accuracy DNN configurations, many times by as much as 40×.

Second, there is an ever-increasing number of mobile applications that use on-device Machine Learning (ML) models employed directly into the smartphone, rather than relying on the cloud to provide the service. Thus, recent work on the design of DNNs shows an ever-increasing interest in developing platform- and device-aware DL models [11, 37]. For instance, state-of-the-art hardware-aware methodologies from Google Brain consider DL optimization and design techniques that optimize for both the accuracy of DNNs for image classification and the runtime of the model on Google Pixel smartphones [37].

Finally, edge-cloud communication constraints are an important design consideration for learning on the edge, *e.g.*, for commercially important indoor localization applications based on low-cost sensing data (*e.g.*, Radio-frequency identification RFID tags and WiFi signals) [28]. Hence, towards enabling hardware-aware ML applications, the aforementioned "hardware wall" gives rise to two main challenges:

Challenge 1: Characterizing hardware performance of DL models: The efficiency of DL models is determined by their hardware performance with respect to metrics such as runtime or energy consumption, not only by their accuracy for a given learning task [37]. To this end, recent work explores modeling methodologies that aim to accurately model and predict the hardware efficiency of DNNs. In this paper, we review state-of-the-art modeling tools, such as the ones employed in Eyeriss [5, 38], or the Paleo [27] and NeuralPower [3] frameworks.

Challenge 2: Designing DL models under Hardware Constraints: The **hyper-parameter optimization** of DL models, *i.e.*, the design of DL models via the tuning of hyper-parameters such as the number of layers or the number of filters per layer, has

emerged as an increasingly expensive process, dubbed by many researchers to be *more of an art than a science*. Hyper-parameter optimization is a challenging design problem due to several reasons. First, in commercially important Big Data applications (*e.g.*, speech recognition), the construction of DL models involves many tunable hyper-parameters [32] and the training of each configuration takes days, if not weeks, to fully train. More importantly, the ability of a human expert to identify the best performing DL model could be hampered significantly if we are to consider hardware constraints and design considerations imposed by the underlying hardware [34]. To this end, there is an ever-increasing interest in works that co-optimize for both the hardware efficiency and the accuracy of the DL model. In this paper, we investigate the main tools for employing hardware-aware hyper-parameter optimization, such as methodologies based on hardware-aware Bayesian optimization [34, 35], multi-level co-optimization [30] and Neural Architecture Search (NAS) [11, 37].

2 RELATED WORK

Reducing the complexity of the ML models has long been a concern for machine learning practitioners. Hence, while this paper focuses on hardware-aware modeling and optimization methodologies, there are orthogonal techniques that have been previously explored as means to reduce the complexity of DL models. In this section, we briefly review these approaches as they relate to hardware-aware modeling and optimization, and we highlight their limitations.

Pruning: Prior art has investigated pruning-based methods that aim at reducing the ML model complexity by gradually removing weights of the model and by retraining it [8, 19, 38]. The key insight behind this type of work, *e.g.*, deep compression [18], is the fact that modern DL models exhibit redundancy to their features and a small degradation of accuracy can be traded off for a significant reduction in the computational cost. This insight has resulted in a significant body of work that incorporates regularizing terms directly in the objective function used for the training of DL model training. These loss regularizers, *e.g.*, L1 norm terms, "zero out" several connections, thus reducing the overall model complexity directly at training. For instance, in [15], the authors propose MorhpNet, a resource-constrained regularization that outperforms prior methodologies in terms of accuracy given a maximum FLOP constraint. Nonetheless, these methods use the number of floating points operations or the number of model parameters as a proxy for the model complexity, without explicitly accounting for hardware-based metrics such as energy or power consumption.

Quantization: Beyond pruning-based methodologies that reduce the DL model complexity, other works reduce directly their computational complexity via quantization [7]. The key insight is that, while the number of FLOPs remains the same, the hardware execution cost is reduced by reducing the bit-width per multiply-accumulate (MAC) operation. In turn, the reduced cost per operation results in overall hardware efficiency, compounded by lower storage requirements. Several methodologies have explored this trade-off by considering different bit-width values [9, 10, 17, 24]. Nevertheless, the effectiveness of quantization methods relies heavily on the performance of the pre-quantized (seed) network [15], without providing any insight on how the sizing of different DL

hyper-parameters affects the overall hardware efficiency and accuracy of the DL model.

Discussion: While pruning- and quantization-based methods have been shown to significantly reduce the computational complexity of ML models, recent work has highlighted some key limitations. First, these methodologies focus on reducing the storage required by model weights or the number of FLOPs. However, Yang *et al.* show that FLOP reduction does not yield the optimal DNN design with respect to power or energy consumption [38] largely because the parameter-heavy layers of a DL model are not necessarily the most energy-consuming. Hence, pruning and quantization methodologies rely on formulations that are hardware-unaware and they do not necessarily result in Pareto-optimal configurations in terms of hardware efficiency, even though the targeted FLOP constraint is satisfied. In contrast, hardware-aware modeling methodologies can pave the way to directly account for energy consumption or runtime, as discussed in the next section.

Second, pruning methodologies traverse the design space only locally around the original pre-trained DL model used as a seed. Consequently, if the configuration of the DL model is viewed as a hyper-parameter optimization problem to be globally solved, pruning methodologies can only reach locally optimal solutions, whose effectiveness is bound by the quality of the seed design. Nevertheless, Tan *et al.* show that that hardware-constrained models such as MobileNets [23] are not Pareto-optimal with respect to accuracy for a given level of hardware complexity. Instead, state-of-the-art results in platform-aware hyper-parameter optimization [37] yield DL design closer to the Pareto front by globally solving the hardware-aware optimization problem. In this paper, we therefore focus on hardware-aware optimization techniques based on Bayesian optimization and Neural Architecture Search (NAS).

3 HARDWARE-AWARE RUNTIME AND ENERGY MODELING FOR DL

3.1 DL model-hardware platform interplay

As mentioned before, effective optimization requires efficient and reliable models. It is not surprising, therefore, that the first efforts in hardware models for DL have been developed in the context of hardware efficient DNNs. In [38], Yang *et al.* have shown that energy-based pruning achieves better energy-efficiency compared to its FLOPs-based counterpart. During each iteration of pruning-finetuning, Yang *et al.* use an energy consumption model to decide on which DNN layers to prune next. To this end, the authors develop a predictive model based on measurements on their hardware accelerator, namely Eyeriss [5]. Their formulation models energy consumption as a weighted function of the number of accessed to the different levels of the memory hierarchy, wherein each level is profiled based on the energy consumption per operation or per memory access pattern (*e.g.*, overhead to access the register file, the global buffer, the main memory, *etc.*).

Specifically, they calculate the energy consumption for each layer in two parts, computation energy consumption (E_{comp}), and data movement energy consumption (E_{data}). E_{comp} is calculated by multiplying the number of MACs in the layer by the energy consumption for running each MAC operations in the computation core. E_{data} is the total summation of energy consumption of all the
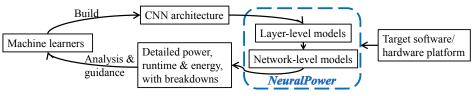
Figure 1: Overview of NeuralPower [3].

memory access for each level of energy. In addition, they account for the impact of data sparsity and bitwidth reduction on energy consumption. This is based on their assumption that a MAC and its memory access can be skipped completely when the input activation or weight is zero. The impact of bitwidth is also calculated by scaling the energy consumption of corresponding hardware units.

This work provides a good model to evaluate the energy cost for running DNNs on the ASICs. However, transferring the model or methodology to other platforms is not trivial as it may require additional model selection and training.

In a more recent development, the Paleo framework proposed by Qi *et al.* [27] proposes a methodology for modeling of hardware metrics, *i.e.*, runtime of DNNs, without resorting to FLOPs as a proxy for hardware efficiency. As opposed to Eyeriss, Paleo has been developed with general purpose platforms (GPUs) in mind. In their approach, the authors present an analytical method to determine the runtime of DNNs executing on various platforms. The framework has been shown to flexibly scale across different GPU platforms and DNN types.

Specifically, Paleo divides the runtime for each layer into two parts, computation time and communication time. For a layer u, and the operation f on a device d, the total execution time $T(u)$ can be expressed as:

$$T(u) = \mathcal{R}(Pa(u)) + C(f, d) + \mathcal{W}(f, d) \qquad (1)$$

where $\mathcal{R}(Pa(u))$ is the time to fetch the input produced by its parent layers, $\mathcal{W}(f, d)$ is the time to write the outputs to the local memory, and $C(f, d)$ is the total computation time with f. Assuming that we are interested in metrics for a device d under average speed conditions, the computation time can be determined by: $C(f, d) = FLOPs(f)/speed(d)$. Following a similar assumption on IO bandwidth, $\mathcal{R}$ and $\mathcal{W}$ can be calculated as the amount of the reading/writing data divided by the average IO bandwidth. In the case of multiple workers, the model still holds by replacing the average IO bandwidth with the communication bandwidth between two devices.

Since the peak FLOPs and peak bandwidth provided by the manufacturers are usually different than the actual speed of the devices running specific DNNs, the authors introduce the concept of *platform percent of peak* (PPP) to capture the average relative inefficiency of the platform compared to peak performance. However, for different devices, the average computation and communication speeds can vary from one to another. Therefore, one needs to evaluate those metrics with many different tests to obtain reliable values. In addition, Paleo does not consider models for predicting power or energy consumption.

3.2 The NeuralPower framework

To provide adaptive and reliable prediction of runtime, power, and energy for DNNs simultaneously, Cai *et al.* [3] have introduced NeuralPower, a layer-wise predictive framework based on sparse polynomial regression for determining the serving energy consumption of convolutional neural networks (CNNs) deployed on GPU platforms. Given the architecture of a CNN, NeuralPower provides an accurate prediction and breakdown for power and runtime across all layers in the whole network. In their framework, the authors also provide a network-level model for the energy consumption of state-of-the-art CNNs.

Specifically, NeuralPower introduces a flexible and comprehensive way to build the runtime, power, and energy models for various DNNs on a variety of platforms. The modeling can be divided into two parts, as shown in Figure 1.

The layer-level approach provides accurate models for running a specific layer on each of the considered platform. Instead of using proxies to determine how fast a layer runs as in Paleo, NeuralPower learns the models by actually running the DL models on the target platform. For example, the runtime $\hat{T}$ of a layer can be expressed as:

$$\hat{T}(\mathbf{x}_T) = \sum_j c_j \cdot \prod_{i=1}^{D_T} x_i^{q_{ij}} + \sum_s c_s' \mathcal{F}_s(\mathbf{x}_T) \qquad (2)$$

$$\text{where } \mathbf{x}_T \in \mathbb{R}^{D_T}; \ q_{ij} \in \mathbb{N}; \ \forall j, \ \sum_{i=1}^{D_T} q_{ij} \leq K_T.$$

The model in Equation 2 consists of two parts. The first part is the regular degree-K_T polynomial terms which are a function of the features in the input vector $\mathbf{x}_T \in \mathbb{R}^{D_T}$. x_i is the i-th component of $\mathbf{x}_T$. q_{ij} is the exponent for x_i in the j-th polynomial term, and c_j is the coefficient to learn. This feature vector of dimension D_T includes layer configuration hyper-parameters, such as the batch size, the input size, and the output size. For different types of layers, the dimension D_T is expected to vary. For convolutional layers, for example, the input vector includes the kernel shape, the stride size, and the padding size, whereas such features are not relevant to the formulation/configuration of a fully-connected layer. The second part is comprised of special polynomial terms $\mathcal{F}$, which represent physical operations related to each layer (*e.g.*, the total number of memory accesses and the total FLOPs). The number of the special terms differs from one layer type to another. Finally, c_s' is the coefficient of the s-th special term to learn.

Similarly, a power model can be constructed and the total energy consumption can be calculated by the product of power consumption and runtime. The models are trained on the real data obtained from real GPU platforms running state-of-the-art DNNs.

One can compare NeuralPower against Paleo with respect to runtime prediction only, as the latter does not address energy or power modeling. Table 1 shows the modeling results for each layer. As it can be seen, NeuralPower outperforms Paleo in terms of model accuracy for the most widely used types of layers in CNNs.

Table 1: Comparison of runtime models for common CNN layers between NeuralPower [3] and Paleo [27].

Layer	NeuralPower [3]			Paleo [27]	
	Model size	RMSPE	RMSE (ms)	RMSPE	RMSE (ms)
CONV	60	39.97%	1.019	58.29%	4.304
FC	17	41.92%	0.7474	73.76%	0.8265
Pool	31	11.41%	0.0686	79.91%	1.763

For the network level model, NeuralPower uses the summation of the layer level results to achieve total runtime and total energy. The average power can be calculated by dividing total energy by total runtime. Table 2 shows the runtime predictions at network level for NeuralPower and Paleo. From Table 1 and Table 2, we can see NeuralPower generally achieves better accuracy in both layer-level and network-level runtime prediction.

Table 2: Comparison of runtime models for common CNNs between NeuralPower [3] and Paleo [27].

CNN	Paleo[27]	NeuralPower[3]	Actual runtime
VGG-16	345.83	373.82	368.42
AlexNet	33.16	43.41	39.02
NIN	45.68	62.62	50.66
Overfeat	114.71	195.21	197.99
CIFAR10-6conv	28.75	51.13	50.09

Lastly and most importantly, NeuralPower is currently the state-of-the-art method in terms of prediction error when capturing the runtime, power, and energy consumption for various CNNs, with mean squared error less than 5%. Moreover, NeuralPower achieves a 70% accuracy improvement compared the previously best model, *i.e.*, Paleo. Using learning-based polynomial regression models, NeuralPower can be flexibly employed for prediction across different DL software tools (*e.g.*, Caffe [25] or TensorFlow [1]) and GPU platforms (*i.e.*, both low-power Nvidia Tegra boards and workstation Nvidia Titan X GPUs).

4 HARDWARE-AWARE OPTIMIZATION FOR DL

4.1 Model-based optimization

Background: Even without hardware-related design objectives, the hyper-parameter optimization of DL models, *i.e.*, deciding on tunable hyper-parameters such as the number of features per layer or the number of layers, is a challenging design task. Early attempts towards efficiently solving this problem use sequential model-based optimization (SMBO) when the objective function (*i.e.*, test error of each candidate DNN configuration) has no simple closed form and its evaluations are costly. SMBO methodologies use a surrogate (cheaper to evaluate) probabilistic model to approximate the objective function.
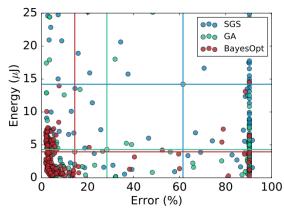

Figure 2: Model-based optimization (denoted as BayesOpt) outperforms alternative methods such as random search and genetic algorithms in terms of finding more accurate, energy-efficient designs closer to the Pareto front [29].

Different formulations have been used for SMBO probabilistic models, such as Gaussian Processes (GP) [32] or tree-structured Parzen estimators (TPE) [2]. Regardless of the choice of the SMBO formulation, intuitively the probabilistic model encapsulates the belief about the shape of functions that are more likely to fit the data observed so far, providing us with a cheap approximation for the mean and the uncertainty of the objective function.

Each SBMO algorithm has three main steps: (i) *maximization of acquisition function*: to select the point (*i.e.*, next candidate DNN configuration) at which the objective will be evaluated next, SMBO methods use the so-called acquisition function. Intuitively, the acquisition function provides the direction in the design space toward which there is an expectation of improvement of the objective. By evaluating cheaply the acquisition function at different candidate points (*i.e.*, different DNN configurations), SMBO selects the point with maximum acquisition function value. (ii) *evaluation of the objective*: the currently selected DNN design is created and trained to completion to acquire the validation error. (iii) *probabilistic model update*: based on the value of the objective at the currently considered DNN design, the probabilistic model is updated.

Hardware-aware SMBO: If Gaussian processes are used to formulate the probabilistic model, the resulting special case SMBO constitutes a powerful approach, namely Bayesian optimization [32]. Prior art has proposed general frameworks for employing Bayesian optimization with multiple objective and constraint terms [21]. To this end, a natural extension is to use these formulations to solve the hyper-parameter optimization of DNNs under hardware constraints.

Hernández-Lobato *et al.* has successfully used Bayesian optimization to design DNNs under runtime constraints [21]. More interestingly, the authors have investigated the co-design of hardware accelerators and DNNs in [22, 29], where the energy consumption and the accuracy of a DNN correspond to objective terms that depend on both DNN design choices (*e.g.*, number of features per layer or number of layers) and hardware-related architectural decisions (*e.g.*, bit width and memory size).
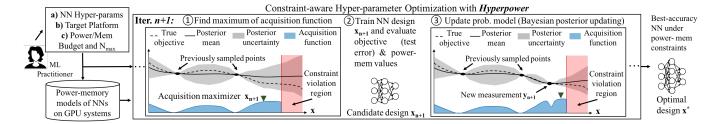
Figure 3: Overview of HyperPower flow and illustration of the Bayesian optimization procedure during each iteration [34].

In Figure 2, we can observe that the SMBO-based method (denoted as BayesOpt) outperforms alternative methods such as random search and genetic algorithms in terms of finding more accurate, energy-efficient designs closer to the Pareto front. This is to be expected, since the model-based approach provides a hardware-aware model to guide the design space exploration faster towards Pareto-optimal designs.

4.2 Multi-layer co-optimization

From a different perspective, Minerva [30] presents an automated co-optimization method including multiple layers in the system stack: algorithm, architecture and circuit levels. In that approach, five stages are used: 1. Training space exploration; 2. Microarchitecture design space; 3. Data type quantization; 4. Selective operation pruning; 5. SRAM fault mitigation. Based on these five steps, the DNNs can be trained, pruned and deployed to Minerva with low power consumption, without sacrificing their model accuracy.

In general, the savings from optimization at the software, architecture, and circuit level, can be combined together. According to tests on five different datasets, including MNIST, Forest, Reuters, WebKB, and 20NG, Minerva achieves an average power saving of 8.1× compared with a baseline accelerator. The authors conclude that fine-grain, heterogeneous datatype optimization, aggressive prediction and pruning of small activity values, and active hardware fault detection coupled with domain-aware error mitigation are the main contributors for the savings. Given its low footprint, Minerva can be applied to IoT and mobile devices, but it doesn't rely on a comprehensive optimization framework that can be extended to generic platforms. We present next one such approach.

4.3 HyperPower framework

A special case of Bayesian optimization is one where constraints can be expressed and known *a priori*; these formulations enable models that can directly capture candidate configurations as valid or invalid [14]. In HyperPower [34], Stamoulis *et al.* investigate the key insight that predictive models (as discussed in the previous section) can provide an *a priori* knowledge on whether the energy consumption of any DNN configuration violates the energy budget or not. In other words, predictive models can be used in the context of Bayesian optimization and can be formulated as *a priori* known constraints.

More specifically, HyperPower introduces a hardware-aware acquisition function that returns zero for all constraint-violating candidate designs, hence effectively guiding the design space exploration toward hardware constraint-satisfying configurations with optimal DNN accuracy. By accounting for hardware constraints directly in the SMBO formulation, HyperPower reaches the near-optimal region 3.5× faster compared to hardware-unaware Bayesian optimization. The proposed acquisition function uses predictive models for power consumption and memory utilization, thus showing the interplay of both modeling and optimization towards enabling hardware-aware DL.

The general structure of HyperPower is shown in Figure 3. Bayesian optimization is a sequential model-based approach that approximates the objective function with a surrogate (cheaper to evaluate) probabilistic model $\mathcal{M}$, based on Gaussian processes (GP). The GP model is a probability distribution over the possible functions of $f(\mathbf{x})$, and it approximates the objective at each iteration $n+1$ based on data $\mathbf{X} := x_i \in \mathcal{X}_{i=1}^n$ queried so far. HyperPower assumes that the values $\mathbf{f} := f_{1:n}$ of the objective function at points $\mathbf{X}$ are jointly Gaussian with mean $\mathbf{m}$ and covariance $\mathbf{K}$, *i.e.*, $\mathbf{f} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$. Since the observations $\mathbf{f}$ are noisy with additive noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$, the GP model can be written as $\mathbf{y} \mid \mathbf{f}, \sigma^2 \sim \mathcal{N}(\mathbf{f}, \sigma^2 \mathbf{I})$. At each point $\mathbf{x}$, GP provides a cheap approximation for the mean and the uncertainty of the objective, written as $p_{\mathcal{M}}(y|\mathbf{x})$ and illustrated in Figure 3 with the black curve and the grey shaded areas.

Each iteration $n+1$ of a Bayesian optimization algorithm consists of three key steps:

Maximization of acquisition function: one first needs to select the point $\mathbf{x}_{n+1}$ (*i.e.*, next candidate NN configuration) at which the objective (*i.e.*, the test error of the candidate NN) will be evaluated next. This task of guiding the search relies on the so-called acquisition function $\alpha(\mathbf{x})$. A popular choice for the acquisition function is the Expectation Improvement (EI) criterion, which computes the probability that the objective function f will exceed (negatively) some threshold y^+, *i.e.*, $EI(\mathbf{x}) = \int_{-\infty}^{\infty} \max\{y^+ - y, 0\} \cdot p_{\mathcal{M}}(y|\mathbf{x}) \, dy$. Intuitively, $\alpha(\mathbf{x})$ provides a measure of the direction toward which there is an expectation of improvement of the objective function.

The acquisition function is evaluated at different candidate points $\mathbf{x}$, yielding high values at points where the GP's uncertainty is high (*i.e.*, favoring exploration), and where the GP predicts a high objective (*i.e.*, favoring exploitation) [32]; this is qualitatively illustrated in Figure 3 (blue curve). HyperPower selects the maximizer of $\alpha(\mathbf{x})$ as the point $\mathbf{x}_{n+1}$ to evaluate next (green triangle in Figure 3). To enable power- and memory-aware Bayesian optimization, *HyperPower* incorporates hardware-awareness directly into the acquisition function.

Evaluation of the objective: Once the current candidate NN design $\mathbf{x}_{n+1}$ has been selected, the NN is generated and trained to completion to acquire the test error. This is the most expensive step. Hence, to enable efficient Bayesian optimization, HyperPower focuses on detecting when this step can be bypassed.

Probabilistic model update: As the new objective value y_{n+1} becomes available at the end of iteration $n + 1$, the probabilistic model $p_{\mathcal{M}}(y)$ is refined via Bayesian posterior updating (the posterior mean $\mathbf{m}_{n+1}(\mathbf{x})$ and covariance covariance $\mathbf{K}_{n+1}$ can be analytically derived). This step is quantitatively illustrated in Figure 3 with the black curve and the grey shaded areas. Attention can be paid to how the updated model has reduced uncertainty around the previous samples and newly observed point. For an overview of GP models the reader is referred to [32].

To enable *a priori* power and memory constraint evaluations that are decoupled from the expensive objective evaluation, HyperPower models power and memory consumption of a network as a function of the J discrete (structural) hyper-parameters $\mathbf{z} \in \mathbb{Z}_+^J$ (subset of $\mathbf{x} \in \mathcal{X}$); it trains on the structural hyper-parameters $\mathbf{z}$ that affect the NN's power and memory (*e.g.*, number of hidden units), since parameters such as learning rate have negligible impact.

To this end, HyperPower employs offline random sampling by generating different configurations based on the ranges of the considered hyper-parameters $\mathbf{z}$. Since the Bayesian optimization corresponds to function evaluations with respect to the test error[33], for each candidate design $\mathbf{z}_l$ HyperPower measures the hardware platform's power P_l and memory M_l values during inference and not during the NN's training. Given the L profiled data points $\{(\mathbf{z}_l, P_l, M_l)\}_{l=1}^L$, the following models that are linear with respect to both the input vector $\mathbf{z} \in \mathbb{Z}_+^J$ and model weights $\mathbf{w}, \mathbf{m} \in \mathbb{R}^J$ are trained, *i.e.*:

$$\text{Power model}: \quad \mathcal{P}(\mathbf{z}) = \sum_{j=1}^{J} w_j \cdot z_j \qquad (3)$$

$$\text{Memory model}: \quad \mathcal{M}(\mathbf{z}) = \sum_{j=1}^{J} m_j \cdot z_j \qquad (4)$$

HyperPower trains the models above by employing a 10-fold cross validation on the dataset $\{(\mathbf{z}_l, P_l, M_l)\}_{l=1}^L$. While the authors experimented with nonlinear regression formulations which can be plugged-in to the models (*e.g.*, see recent work [3]), these linear functions provide sufficient accuracy. More importantly, HyperPower selects the linear form since it allows for the efficient evaluation of the power and memory predictions within the acquisition function (next subsection), computed on each sampled grid point of the hyper-parameter space.

HW-IECI: In the context of hardware-constraint optimization, EI allows to directly incorporate the *a priori* constraint information in a representative way. Inspired by constraint-aware heuristics [14] [16], the authors propose a power and memory constraint-aware acquisition function:

$$a(\mathbf{x}) = \int_{-\infty}^{\infty} \max\{y^+ - y, 0\} \cdot p_{\mathcal{M}}(y|\mathbf{x}) \cdot$$
$$\mathbb{I}[\mathcal{P}(\mathbf{z}) \le \text{PB}] \cdot \mathbb{I}[\mathcal{M}(\mathbf{z}) \le \text{MB}] \, dy \qquad (5)$$

where $\mathbf{z}$ are the structural hyper-parameters, $p_{\mathcal{M}}(y|\mathbf{x})$ is the predictive marginal density of the objective function at $\mathbf{x}$ based on
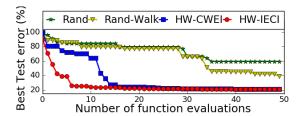

Figure 4: Assessment of HyperPower (denoted as HW-IECI) against hardware-unaware methods. Best observed test error against the number of function evaluations [34].

surrogate model M. $\mathbb{I}[\mathcal{P}(\mathbf{z}) \le \text{PB}]$ and $\mathbb{I}[\mathcal{M}(\mathbf{z}) \le \text{MB}]$ are the indicator functions, which are equal to 1 if the power budget PB and the memory budget MB are respectively satisfied. Typically, the threshold y^+ is adaptively set to the best value $y^+ = \max_{i=1:n} y_i$ over previous observations [32][14].

HyperPower captures the fact that improvement should not be possible in regions where the constraints are violated. Inspired by the integrated expected conditional improvement (IECI) [16] formulation, the authors refer to this proposed methodology as *HW-IECI*. Uncertainty can be also encapsulated by replacing the indicator functions with probabilistic Gaussian models as in [16], whose implementation is already supported by the used tool [33].

NeuralPower exploits the use of trained predictive models based on the power and memory consumption of DNNs, allowing the HyperPower framework to navigate the design space in a constraint "complying" manner. As shown in Figure 4, the authors observe that the proposed HyperPower methodology (denoted as HW-IECI) converges to the high-performing, constraint-satisfying region faster compared to the hardware unaware one.

A key advantage of model-based optimization methodologies is that, given their black-box optimization nature, they can be extended to various type of hardware constraints and design considerations. For instance, Stamoulis *et al.* [35] show that Bayesian optimization can be used to efficiently design adaptive DNNs under edge-node communication constraints. In particular, hardware-aware adaptive DNNs achieve 6.1× more energy efficient designs than state-of-the-art for same accuracy.

4.4 Platform-aware NAS

Traditional SMBO-based black-box optimization methods fit a probabilistic model over the entire design, *i.e.*, the hyper-parameters of all layers can be simultaneously changed. Hence, while Bayesian optimization methodologies are extremely efficient for design spaces with up to 20 hyper-parameters, they could suffer from high dimensionality in larger DL models, *e.g*, parameter-heavy Recurrent Neural Networks (RNNs) used in speech recognition applications.

To address this challenge, recent breakthroughs in hyper-parameter optimization focus on the following insight: DL models designed by human experts exhibit pattern repetition, such as the residual cell in ResNet-like configurations [20]. To this end, recent work focuses on the hyper-parameter optimization of a repetitive motif, namely cell, which is identified on a simpler dataset. After the optimal cell is identified, it is repeated several times to construct larger, more complex DL models that achieve state-of-the-art performance on larger datasets and more complex learning tasks.
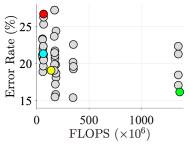
Figure 5: Pareto-optimal candidates (colored dots) identified by platform-aware NAS-based method [11].

Intuitively, this transferability property of the optimal cell allows the hyper-parameter optimization probabilistic model to focus on a smaller structured design spaces with fewer hyper-parameters. Recent work uses techniques such as Reinforcement learning (RL) or Evolutionary algorithms (EA) to guide the exploration towards the optimal cell. The body of work based on this insight is called Neural Architecture Search (NAS). Nevertheless, initial NAS-based methods focused explicitly on optimizing accuracy, without considering hardware-related metrics.

Motivated by this observation, the recently published NAS frameworks MnasNet [37] and DPP-Net [11] propose formulations that co-design the cell for both DNN accuracy and runtime. More specifically, DPP-Net proposes a progressive search for Pareto-optimal DNNs, where a probabilistic model is obtained on smaller cell designs and new, more complex cells are being proposed by the probabilistic model as the search progresses. The DPP-Net NAS-based progressive model ranks candidate cell configurations based on a weighted ranking of both the anticipated DL accuracy and the runtime of each configuration measured on a hardware platform. Similarly, MnasNet considers a multi-objective term to be optimized, then accounts for both runtime (on a Google Pixel phone platform) and accuracy of candidate cell designs. Figure 5 in [11] shows the Pareto optimal DNN candidates with respect to error and FLOPS. The NAS-based method, *i.e.*, DPP-Net, is able to identify models with a large number of parameters and slow inference time. The DPP-Nets can achieve better trade-off among multiple objectives compared with state-of-the-art mobile CNNs and models designed using architecture search methods. Similarly, the MNAS method outperforms the state-of-the-art handcrafted designs, namely MobileNet [23], by identifying designs closer to the Pareto front.

The ever-increasing interest of ML practitioners in NAS methods shows that these methodologies will provide the foundation for novel hardware-aware optimization techniques. Several open questions remain to be investigated, as we motivate further in the following section.

5 UNEXPLORED RESEARCH DIRECTIONS

5.1 Hardware-aware modeling

As motivated in the previous section, the development of predictive models to capture the hardware performance of DL application has already played a critical role in the context of hyper-parameter optimization. To this end, we postulate that predictive models are poised to have significant impact towards enabling hardware-aware

ML. However, the following directions of research are crucial for applicability of these models in a co-design environment

5.1.1 Hardware-aware models for DL with general topologies. Current work has addressed building models largely for linear or pipelined neural network structures. However, as mentioned before, such predictive models should be suitably extended or developed anew for nonlinear DNN structures [3]. This is of critical importance especially in the context of NAS-based methods, where the cell design consists of several concatenating operations that are executed in a nonlinear fashion with several non-sequential dependencies, unlike the case of traditional DNN designs. To this end, the development of hardware-aware predictive models for NAS-like frameworks is essential.

5.1.2 Cross-platform hardware-aware models. Existing runtime and energy models currently consider specific types of platforms, with Nvidia GPUs being the hardware fabric in both the Paleo [27] and the NeuralPower [3] frameworks. It is therefore important to explore other platforms and segments of computing systems spanning the entire edge to server continuum. It becomes crucial to develop predictive models for capturing the runtime or energy consumption for other types of neural accelerators and platforms, such as reconfigurable architectures presented in [12]. Towards this direction, we postulate that cross-platform models or models that account for chip variability effects [4, 6, 36] can significantly help ML practitioners to transfer knowledge from one type of hardware platform to another and to choose the best model for that new platform.

5.2 Hardware-aware optimization

There are several directions to advance state-of-the-art in the context of hardware-aware hyper-parameter optimization, especially given the recent interest in NAS-based formulations.

5.2.1 Multi-objective optimization. The design of DL models is a challenging task that requires different trade-offs beyond the currently investigated accuracy versus runtime/energy cases. For instance, while the use of a simpler DNN design can improve the overall runtime, it could significantly degrade the utilization or throughput achieved given a fixed underlying hardware platform. Hence, we believe that several novel approaches that focus on hardware-aware hyper-parameter optimization would be extending current SMBO models to multiple design objectives.

5.2.2 Hardware-DL model co-optimization. We postulate that model-based hyper-parameter optimization approaches will allow innovation beyond the design of DL models, since the predictive models can be viewed also as a function of the underlying hardware platform. That is, Bayesian optimization- and NAS-based formulations can be extended to cases where both the DL model and the hardware are co-designed. For instance, the optimal NAS-like cell can be identified while varying the hardware hyper-parameters of a reconfigurable architectures presented, such as the number of processing elements [12]. Initial efforts focusing on the design of hardware accelerators based on 3D memory designs [13] already exploit energy-based models [5] for design space exploration.

6 CONCLUSION

To conclude, tools and methodologies for hardware-aware machine learning have increasingly attracted attention of both academic and industry researchers. In this paper, we have discussed recent work on modeling and optimization for various types of hardware platforms running DL algorithms and their impact on improving hardware-aware DL design. We point out several potential new directions in this area, such as cross-platform modeling and hardware-model co-optimization.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A system for large-scale machine learning. *arXiv preprint arXiv:1605.08695* (2016).

[2] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*. 2546–2554.

[3] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. 2017. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. *arXiv preprint arXiv:1710.05420* (2017).

[4] Ermao Cai, Dimitrios Stamoulis, and Diana Marculescu. 2016. Exploring aging deceleration in FinFET-based multi-core systems. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 1–8.

[5] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[6] Zhuo Chen, Dimitrios Stamoulis, and Diana Marculescu. 2017. Profit: priority and power/performance optimization for many-core systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2017).

[7] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[8] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. 2017. NeST: A Neural Network Synthesis Tool Based on a Grow-and-Prune Paradigm. *arXiv preprint arXiv:1711.02017* (2017).

[9] Ruizhou Ding, Zeye Liu, RD Shawn Blanton, and Diana Marculescu. 2018. Quantized deep neural networks for energy efficient hardware-based inference. In *Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific*. IEEE, 1–8.

[10] Ruizhou Ding, Zeye Liu, Rongye Shi, Diana Marculescu, and RD Blanton. 2017. LightNN: Filling the Gap between Conventional Deep Neural Networks and Binarized Networks. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 35–40.

[11] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. 2018. DPP-Net: Device-aware Progressive Search for Pareto-optimal Neural Architectures. *arXiv preprint arXiv:1806.08198* (2018).

[12] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*.

[13] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. *ACM SIGOPS Operating Systems Review* 51, 2 (2017), 751–764.

[14] Michael A Gelbart, Jasper Snoek, and Ryan P Adams. 2014. Bayesian optimization with unknown constraints. *arXiv preprint arXiv:1403.5607* (2014).

[15] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. 2018. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[16] Robert B Gramacy and Herbert KH Lee. 2010. Optimization Under Unknown Constraints. *arXiv preprint arXiv:1004.4027* (2010).

[17] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*. 1737–1746.

[18] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[19] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *NIPS*. 1135–1143.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[21] José Miguel Hernández-Lobato, Michael A Gelbart, Ryan P Adams, Matthew W Hoffman, and Zoubin Ghahramani. 2016. A general framework for constrained Bayesian optimization using information-based search. *The Journal of Machine Learning Research* 17, 1 (2016), 5549–5601.

[22] José Miguel Hernández-Lobato, Michael A Gelbart, Brandon Reagen, Robert Adolf, Daniel Hernández-Lobato, Paul N Whatmough, David Brooks, Gu-Yeon Wei, and Ryan P Adams. 2016. Designing neural network hardware accelerators with decoupled objective evaluations. In *NIPS workshop on Bayesian Optimization*. l0.

[23] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[24] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *arXiv preprint arXiv:1712.05877* (2017).

[25] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[27] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. Paleo: A Performance Model for Deep Neural Networks. In *Proceedings of the International Conference on Learning Representations*.

[28] Sreeraj Rajendran, Wannes Meert, Domenico Giustiniano, Vincent Lenders, and Sofie Pollin. 2017. Distributed deep learning models for wireless signal classification with low-cost spectrum sensors. *arXiv preprint arXiv:1707.08908* (2017).

[29] Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017. A case for efficient accelerator design space exploration via Bayesian optimization. In *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*. IEEE, 1–6.

[30] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 267–278.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.

[32] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2016. Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE* 104, 1 (2016), 148–175.

[33] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[34] Dimitrios Stamoulis, Ermao Cai, Da-Cheng Juan, and Diana Marculescu. 2018. HyperPower: Power-and memory-constrained hyper-parameter optimization for neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 19–24.

[35] Dimitrios Stamoulis, Ting-Wu Chin, Anand Krishnan Prakash, Haocheng Fang, Sribhuvan Sajja, Mitchell Bognar, and Diana Marculescu. 2018. Designing Adaptive Neural Networks for Energy-Constrained Image Classification. *arXiv preprint arXiv:1808.01550* (2018).

[36] Dimitrios Stamoulis and Diana Marculescu. 2016. Can we guarantee performance requirements under workload and process variations?. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 308–313.

[37] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. 2018. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *arXiv preprint arXiv:1807.11626* (2018).

[38] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.