TEA-DNN: the Quest for Time-Energy-Accuracy Co-optimized Deep Neural Networks

Lile Cai* I2R, Singapore caill@i2r.a-star.edu.sg
Anne-Maelle Barneche* SUPELEC, France anne-maelle.barneche@supelec.fr
Arthur Herbout* ECP, France arthur.herbout@student.ecp.fr
Chuan Sheng Foo I2R, Singapore foo_chuan_sheng@i2r.a-star.edu.sg
Jie Lin I2R, Singapore lin-j@i2r.a-star.edu.sg
Vijay Ramaseshan Chandrasekhar I2R, Singapore vijay@i2r.a-star.edu.sg
Mohamed M. Sabry NTU, Singapore msabry@ntu.edu.sg

***Abstract*—Embedded deep learning platforms have witnessed two simultaneous improvements. First, the accuracy of convolutional neural networks (CNNs) has been significantly improved through the use of automated neural-architecture search (NAS) algorithms to determine CNN structure. Second, there has been increasing interest in developing application-specific platforms for CNNs that provide improved inference performance and energy consumption as compared to GPUs. Embedded deep learning platforms differ in the amount of compute resources and memory-access bandwidth, which would affect performance and energy consumption of CNNs. It is therefore critical to consider the available hardware resources in the network architecture search. To this end, we introduce TEA-DNN, a NAS algorithm targeting multi-objective optimization of execution time, energy consumption, and classification accuracy of CNN workloads on embedded architectures. TEA-DNN leverages energy and execution time measurements on embedded hardware when exploring the Pareto-optimal curves across accuracy, execution time, and energy consumption and does not require additional effort to model the underlying hardware. We apply TEA-DNN for image classification on actual embedded platforms (NVIDIA Jetson TX2 and Intel Movidius Neural Compute Stick). We highlight the Pareto-optimal operating points that emphasize the necessity to explicitly consider hardware characteristics in the search process. To the best of our knowledge, this is the most comprehensive study of Pareto-optimal models across a range of hardware platforms using actual measurements on hardware to obtain objective values.**

***Index Terms*—Neural architecture search, hardware constraints, multi-objective optimization**

I. Introduction

Deep convolutional neural networks (CNNs) have achieved state-of-the-art performance in image classification, object detection and many other applications [1]. To achieve better accuracy, CNN models have become increasingly deeper and require more computational resources [2], [3]. This poses a challenge when these models are deployed to run on resource-limited devices, such as mobile and embedded platforms, as the memory on these devices may not be large enough to hold the models or running the model may consume more power than the device can supply.

*Equal contribution.

Much effort has been devoted to designing CNN models that can run efficiently on these devices, for instance, by manually designing more efficient convolution operations and network architectures [4]–[6]. However, this approach demands expert knowledge and obtaining an optimal model is difficult as one has to carefully balance the trade-off between accuracy and computational resources. An alternative approach is to use automated neural architecture search (NAS) algorithms to find optimal models under hardware constraints [7]–[9]. NAS algorithms usually consist of a controller, which is responsible for sampling models from the search space, a trainer, which is responsible for training the sampled models and an evaluator, which is responsible for evaluating the objective values (like model accuracy) on currently sampled models. The parameters of the controller are then updated to increase the likelihood that it subsequently samples better models.

Due to the variation in device hardware/software configurations, it can happen that models optimized for one device are suboptimal for another. Consider two hardware platforms, namely the TITAN X GPU [10] and Intel Movidius Neural Computing Stick (NCS) [11] that respectively exemplify a high-performance and embedded platform. Figure 1 displays the Pareto-optimal CNN models searched for the GPU and NCS respectively for the trade-off between classification error and inference time, which are then both executed on the TITAN X GPU (for measurement details see Section IV). It can be seen that the Pareto-optimal models searched for the NCS are far from the Pareto curve for the GPU, implying that a platform-agnostic NAS may result in highly suboptimal models – in this case resulting in up to $2\times$ increase in execution time to achieve comparable accuracy.

The problem revealed in Fig. 1 demonstrates that in order to obtain an optimal model for a hardware platform, its corresponding characteristics have to be taken into consideration during the search process. To this end, we introduce TEA-DNN (Time-Energy-Accuracy co-optimized Deep Neural Networks), a NAS framework that explicitly considers two hardware metrics – inference time and energy consumption – in addition to classification accuracy as objective metrics. We formulate this problem as a multi-objective optimization

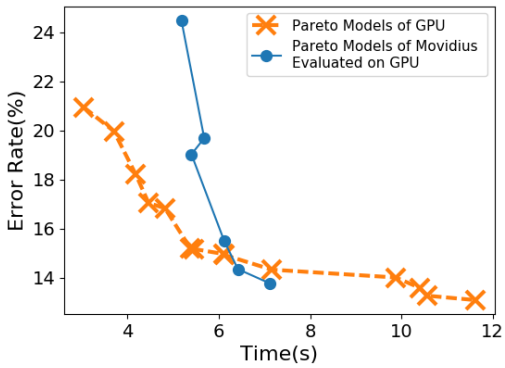

Fig. 1. Pareto-optimal models searched on Movidius are suboptimal on TITAN X GPU.

problem and leverage Bayesian optimization to search for the Pareto-optimal points. While Bayesian optimization has been used to obtain hardware-aware neural networks [12], it was only used to search for several hyper-parameters with a fixed network architecture. To the best of our knowledge, our work is the first to apply Bayesian optimization for neural architecture search. Furthermore, TEA-DNN does not require modeling the hardware platform and instead leverages the ability to directly measure energy and execution time on actual hardware. We summarize our contributions as follows:

- A time, energy and accuracy co-optimization framework for CNNs.
- Employing Bayesian optimization to search for CNN structures that yield Pareto-optimal operating conditions.
- We demonstrate how different device configurations can lead to different trade-off behaviors.
- We demonstrate that optimal models searched on one hardware platform are not optimal for another and thus reiterate the importance of hardware-aware NAS.

II. Related Work

A. Neural Architecture Search (NAS)

Early versions of NAS algorithms [7] employed recurrent neural networks (RNNs) to predict the architecture of a target CNN where the weights of the RNN are updated using reinforcement learning. [8] follows the same framework as proposed in [7], but instead of using a RNN to predict the entire network architecture, the algorithm only predicts the optimal structure for one convolutional module (or "cell"). Identical cells are then stacked multiple times to form the full network. [9] replaced reinforcement learning with progressive search, which can yield better models with fewer samples.

B. Hardware-Aware NAS

Explicitly incorporating hardware constraints into NAS has been an active research topic in recent years. HyperPower [12] approximates the power and memory consumption of a networks using linear regression and the approximated functions are then used in the acquisition function in Bayesian optimization to avoid sampling constraint-violated models. MnasNet [13] focused on searching optimal networks for mobile devices and used inference time as one of the objectives. DPP-Net [14] performs the search on different devices and considers more objective metrics, e.g., error rate, number of parameters, FLOPs, memory, and inference time.

Our work is closely related to MnasNet and DPP-Net in that we all search for the Pareto-optimal networks for a specific device. Our approach is unique in two aspects: 1) we perform true multi-objective optimization instead of combining several objectives into a single objective as done in MnasNet; 2) Unlike DPP-Net, we do not use any surrogate functions to approximate the optimization objectives. Instead, we directly measure the *real-world* values for all the three objectives (i.e., time, energy and accuracy). This eliminates the need to model the targeted hardware, which is a challenging task given the diversity of hardware platform configurations.

III. TEA-DNN Optimization Framework

A. System Overview

We formulate the neural network architecture search problem as a multi-objective optimization problem $\mathbf{min}_x(error(x), energy(x), time(x))$ where we wish to find a network architecture parameterized by x (see Section III-B for details) that minimizes classification *error*, *energy* consumption, and inference *time*. We do not assume a closed-form model for energy consumption or inference time, but evaluate them directly on actual hardware to measure real-world performance. Networks were trained and evaluated on GPUs for efficiency as we assume that classification error is not affected by the specific hardware a network is run on. As formulated, this is an instance of a black-box optimization problem where the objective functions can only be evaluated (and are not differentiable), and where function evaluations (especially classification error, which requires training the model) are costly. Note that no single "best" point exists for a multi-objective optimization problem. A solution is instead defined by a Pareto optimal set of points, for which improvement in any objective function cannot be made without negatively affecting some other objectives.

We chose to employ a Bayesian optimization algorithm [15] (detailed in Section III-C) to solve this optimization problem. We provide a brief overview and refer the reader to the comprehensive review in [16]. Bayesian optimization algorithms perform a sequential exploration of the parameter space while building a surrogate probabilistic model to approximate the objective functions. This model is used to *select points* at which to next *evaluate* the objective functions, and the obtained function values are then used to *update the model*. The algorithm proceeds iteratively following this *select-evaluate-update* loop, such that points in the Pareto optimal set are selected more frequently as the algorithm progresses. We stopped the algorithm after a specified number of iterations or when a time limit is hit. A schematic overview of our search algorithm is shown in Fig. 2.

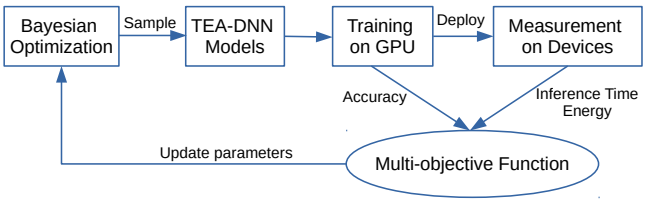

Fig. 2. System diagram of the proposed neural architecture search method.

TABLE I
SPECIFICATIONS OF THE DEVICES USED IN OUR EXPERIMENTS.

	GTX TITAN X	Jetson TX2	Movidius
Processing Unit	3072 CUDA cores	256 CUDA cores	Myriad 2 VPU
FLOPS	6.7T FP32	1.5T FP32	2T FP16
Memory	12GByte GDDR5	8GByte LPDDR4	4GBit LPDDR3
Mem. Bandwidth	336.6 GBytes/s	59.7 GBytes/s	4 GBits/s
Power	250 W	15 W	1 W

B. Search Space

We search over the subset of network architectures that can be described as repetitions of a modular network "cell", as proposed by [9]. The overall network architecture is predefined (Fig. 3(a)) and consists of cells with either stride 1 or 2. As a common heuristic, the number of filter channels is doubled after the stride 2 cells. As such, the network architecture is uniquely determined by the initial filter channel number F, the number of cell repeats N and the cell structure. F and N are hyper-parameters that are pre-specified and the cell structure is searched using Bayesian optimization.

Specifically, each cell is composed of 5 building blocks and each building block (illustrated in Fig. 3(b)) is parameterized by 4 parameters (I_1, I_2, O_1, O_2) for a 20-dimensional parameter space. I_1 and I_2 denote the inputs, and O_1 and O_2 specify the operations applied to the respective inputs. The input space of each building block consists of the outputs of all preceding blocks in the current cell as well as outputs from the two preceding cells. The operation space includes the following eight functions commonly used in top performing CNNs:

1) sep 3×3: 3×3 depthwise-separable convolution
2) sep 5×5: 5×5 depthwise-separable convolution
3) sep 7×7: 7×7 depthwise-separable convolution
4) identity: identity mapping
5) avg 3×3: 3×3 average pooling
6) max 3×3: 3×3 max pooling
7) 1×7 7×1: 1×7 convolution followed by 7×1 convolution
8) dil 3×3: 3×3 dilated convolution with dilation rate = 2

The search space described above has an order of 10^{14} ($2^2 \times 8^2 \times 3^2 \times 8^2 \times 4^2 \times 8^2 \times 5^2 \times 8^2 \times 6^2 \times 8^2 = 5.6 \times 10^{14}$). The outputs of the two operations are then combined by element-wise addition. The final output of the cell is the concatenation of all unused building block outputs.

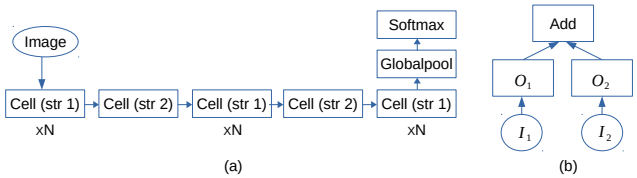

Fig. 3. The network used for CIFAR-10 (a) and the architecture of one building block (b).

C. Multi-Objective Bayesian Optimization

Bayesian optimization is a sequential model-based approach that approximates each objective function with a Gaussian process (GP) model. For a particular objective function (e.g., classification error) let $f(x)$ be its surrogate GP model, $x_{1:n}$ be the evaluated network architectures in the search space, $f_i = f(x_i)$ be the objective function value for network x_i, and y_i be the actual measured function value. The GP model assumes that $\mathbf{f} = f_{1:n}$ are jointly Gaussian with mean $\mathbf{m}$ and covariance $\mathbf{K}$ and observations $\mathbf{y} = y_{1:n}$ are normally distributed given $\mathbf{f}$:

$$\begin{aligned} \mathbf{f} &\sim N(\mathbf{m}, \mathbf{K}), \\ \mathbf{y}|\mathbf{f}, \sigma^2 &\sim N(\mathbf{f}, \sigma^2 \mathbf{I}). \end{aligned} \tag{1}$$

Each iteration of Bayesian optimization consists of 3 steps:

1) Selecting the next point (network architecture to evaluate) x_{n+1} by maximizing an acquisition function, which specifies a likely candidate that improves the objective(s). We used the PESMO (Predictive Entropy Search Multi-objective) [15] acquisition function in our experiments that chooses points which maximally reduce the entropy of the current posterior distribution given by the GPs over the Pareto set.
2) Evaluating the objective functions at x_{n+1}.
3) Updating the parameters $\mathbf{m}$ and $\mathbf{K}$ for the GP models.

To employ Bayesian optimization for neural architecture search, we use the 20-dimensional parameterization of the search space as described in Section III-B. Our three objective functions are the 1) error rate (i.e., $1 -$ accuracy), 2) inference time and 3) energy consumption, and we used the open-source PESMO implementation in Spearmint [15] for our experiments.

IV. Experimental Setup

We evaluate TEA-DNN models on different deep-learning hardware platforms, representing embedded and server-based systems. Table I summarizes the properties of these platforms.

A. Training Setup

In our experiments, models are trained and tested on the CIFAR-10 dataset [17], which is a popular benchmarking dataset for image classification. CIFAR-10 has 50,000 training images and 10,000 test images of dimension $32 \times 32 \times 3$. We removed 5,000 images from the training set for use as a validation set and train on the remaining 45,000 images. During the search process, each model is trained for 20 epochs with a batch size of 32. We use the RMSProp optimizer [18] with momentum and decay both set to 0.9. The learning rate is set to 0.01, and decayed by 0.94 every 2 epochs. Weight

decay is set to 0.00004. The data augmentation technique we used is as described in [9]. The initial channel number F is set to 24 and the number of cell repeats N is set to 2 in the search process.

B. Time and Energy Measurement

We describe how we measure inference time and energy consumption on each of the hardware platforms in Table I:

i **TITAN X GPU** [10]. We run the model on the 5,000 validation images with a batch size of 100 and report the total inference time. We use the NVIDIA Management Library (NVML) [19] to monitor power consumption during evaluation and compute energy consumption by integrating the collected power values over the total inference time.

ii **Jetson TX2** [20]. We run the model on the 5,000 validation images and report the total inference time. The batch size is set to 1 to match actual use scenario. We use the Python library provided by [21] to monitor power and compute energy consumption by integrating the collected power values over the total inference time.

iii **Movidius NCS** [11]. We use the profiling tool provided by the Movidius Neural Compute SDK [22] to obtain the inference time. We use a power meter [23] attached to the NCS to monitor power consumption and compute energy consumption by integration.

V. Results and Discussions

A. Evolution of Pareto Curve

To demonstrate the effectiveness of Bayesian optimization in searching time-energy-accuracy co-optimized DNN models, we plot the Pareto curves for the TITAN X GPU as the search progresses in Fig. 4. We see that the Pareto curve evolves towards the bottom-left corner as the search progresses, implying that models with better trade-offs are being found. By about 500 iterations, the curve does not change much suggesting that the optimization has converged.

B. Accuracy Benchmarking

We evaluate the performance of TEA-DNN models on the CIFAR-10 test subset. For each device, we select the model that achieves the best accuracy and train it for 300 epochs. The starting learning rate is set to 0.025 and decayed by 0.1 every 150 epochs. F is set to 48 and N is set to 3 (Section III-B). The results are reported in Table II. It can be seen that models searched on different devices achieve similar error rates. However, the numbers of parameters and multiplication-add operations on the Movidius NCS are half those on the GPU. Indeed, the extremely limited resources on the NCS caused TEA-DNN to explore network structures that lower the energy consumption on an embedded device, but would not affect the execution time or energy consumption on the higher-performance TITAN X GPU (CNNs with low number of parameters do not fully utilize the parallel resources in a GPU). This is further illustrated in Fig. 5 where the cell structures of the models evaluated in Table II are shown. It can be seen that compared with those for the TITAN X and Jetson, the Pareto-optimal structure for the Movidius NCS uses more sep 5×5 and sep 7×7 operations, as they utilize much fewer parameters than normal convolutions and thus allowing the use of a larger kernel size that helps to achieve high accuracy.

TABLE II
Performance on CIFAR-10 Test subset of TEA-DNN models.

	Error Rate (%)	#Parameters	#Mult-Adds
TITAN X GPU	7.16	15.3M	2.7B
Jetson TX2	7.23	10.1M	1.8B
Movidius NCS	6.99	7.3M	1.1B

C. Pareto Curve on Different Devices

We obtain the Pareto points for each pair of objectives (error-time, error-energy and energy-time) and illustrate the resulting curves for TITAN X, Jetson TX2 and Movidius in Fig. 6, Fig. 7 and Fig. 8, respectively. We see that time or energy has to increase to reduce the error rate for all hardware platforms, which is an intuitive result – deeper CNNs reduce error rates by using more compute operations. However, for the energy-time trade-off, the three platforms exhibit different behaviors. For the TITAN X, there is only one optimal point. For Jetson and Movidius, however, there is a trade-off between time and energy. This behavior is likely related to the following three factors: the relatively small input images, the amount of on-chip memory, and the bandwidth to off-chip memory. A model with fewer parameters and a larger number of activations can consume more time and less energy on a platform with limited on-chip memory—a substantial amount of time would be spent on waiting for memory-access requests to be serviced while computing units sit idle.

D. Cross-Device Evaluation of Pareto-Optimal Models

We evaluate whether a set of Pareto-optimal models searched for one platform is also Pareto-optimal for another. For brevity, we consider the error rate v.s. time trade-off. First, we evaluate Pareto-optimal models searched for the TITAN X GPU on the Jetson TX2 (Fig. 9(a)) and Movidius NCS (Fig. 9(b)). For the Jetson TX2, none of the Pareto-optimal models searched for the GPU is Pareto-optimal. For the Movidius NCS, only 3 out of the 9 models are Pareto-optimal. This clearly indicates that incorporating energy and execution time of the targeted platform is key in TEA-DNN.

In addition, we evaluate the set of Pareto-optimal models for the Jetson TX 2 and Movidius NCS on the TITAN X GPU, as illustrated in Fig. 10. For Jetson TX2 and Movidius NCS, there are both only two models that are Pareto-optimal on GPU. We also note that models searched for the embedded systems distribute within a limited time range on GPU, which suggests that the limited resources on embedded platforms yield CNNs architectures that cannot fully leverage the availability of compute resources in the high-performance GPU.

VI. Conclusions

In this work, we propose the TEA-DNN framework that employs Bayesian optimization to search for time-energy-accuracy co-optimized CNN models. We apply TEA-DNN

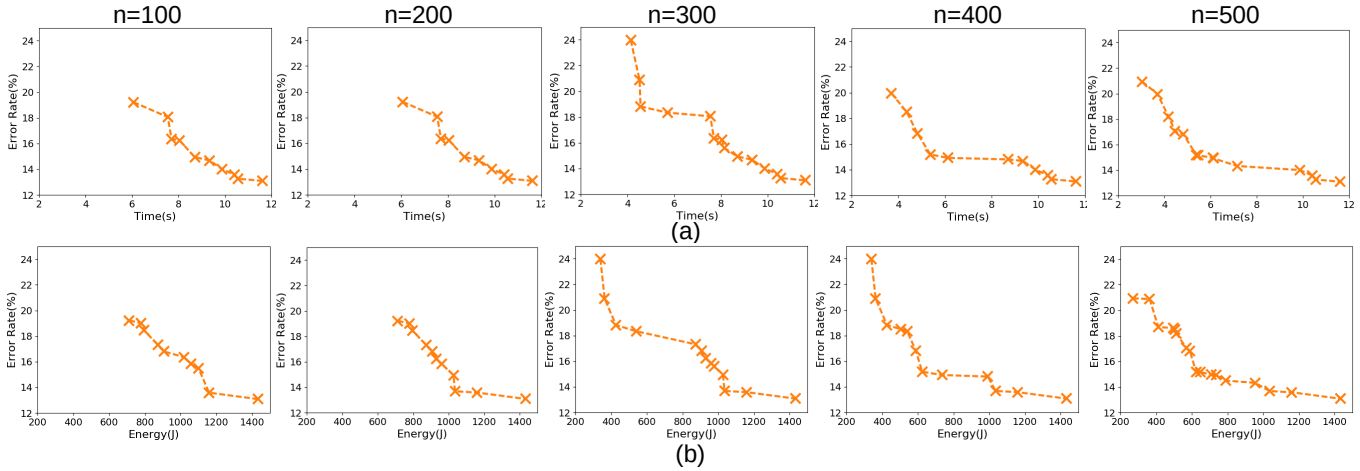

Fig. 4. Evolution of Pareto curves found by Bayesian optimization on TITAN X GPU. Figures are drawn for every 100 iterations. (a) Pareto curve of the error-time trade-off; (b) Pareto curve of the error-energy trade-off.

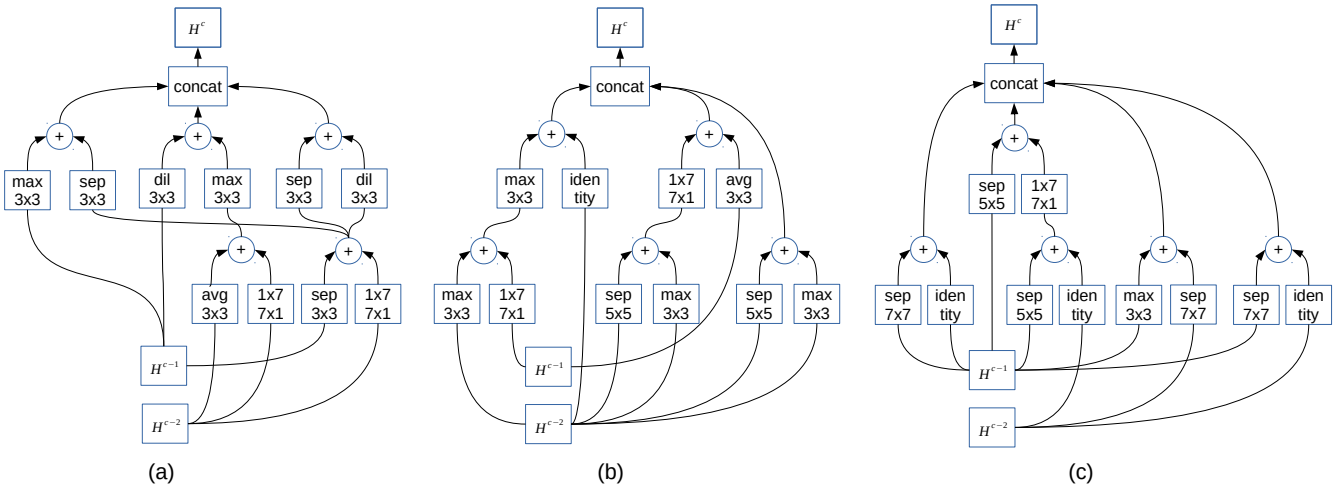

Fig. 5. The cell structures of the models evaluated in Table II. (a) TITAN X GPU; (b) Jetson TX2; (c) Movidius NCS.

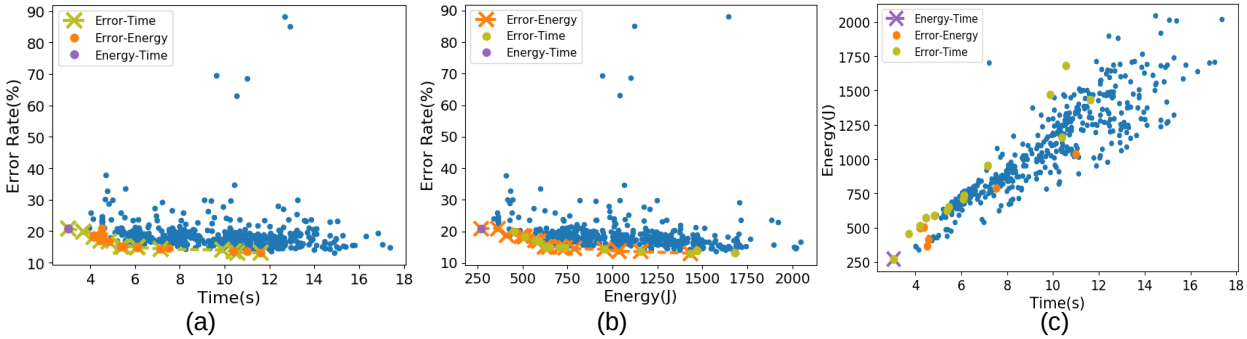

Fig. 6. Pareto curves on TITAN X GPU. The figures show the Pareto set (crosses) for the 500 searched results (blue dots). (a) Error rate v.s. time trade-off. (b) Error rate v.s. energy trade-off. (c) Energy v.s. time trade-off. In each subplot, we also show the Pareto points from the other two trade-offs.

on three different devices: TITAN X GPU, Jetson TX2 and Movidius NCS. Experimental results show that TEA-DNN can effectively find Pareto-optimal models within a few hundred iterations. By analyzing the Pareto curve of the search results, we demonstrate that different device configurations can lead to different trade-off behaviors. Cross-device evaluation of Pareto-optimal models demonstrates that optimal models searched for one hardware platform are not optimal for another and thus reiterates the importance of explicitly considering hardware characteristics in NAS.

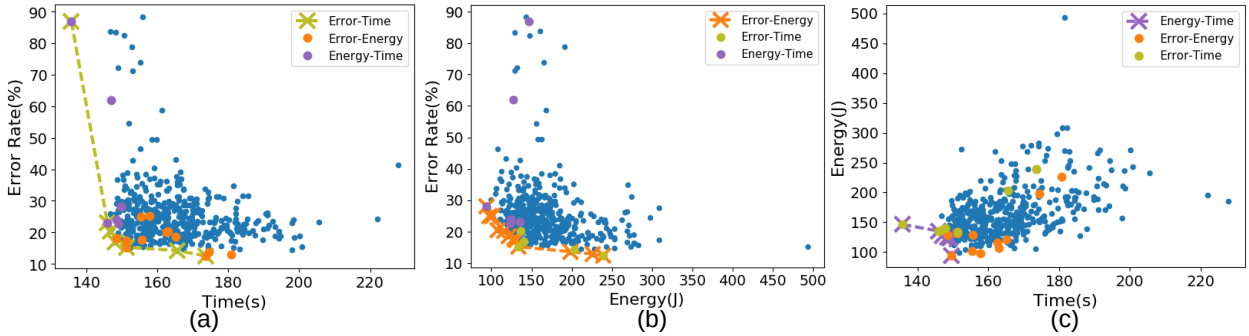

Fig. 7. Pareto curves on Jetson TX2. The figures show the Pareto set (crosses) for the 461 searched results (blue dots). (a) Error rate v.s. time trade-off. (b) Error rate v.s. energy trade-off. (c) Energy v.s. time trade-off. In each subplot, we also show the Pareto points from the other two trade-offs.

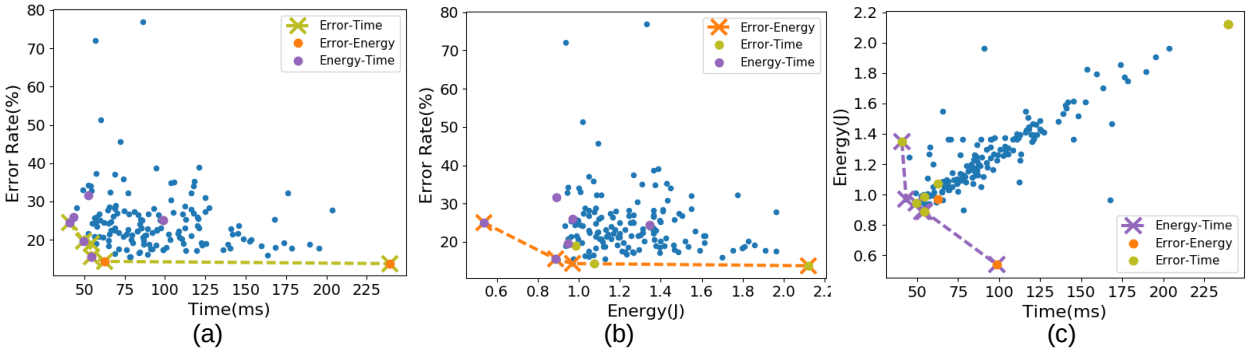

Fig. 8. Pareto curves on Movidius NCS. The figures show the Pareto set (crosses) for the 156 searched results (blue dots). (a) Error rate v.s. time trade-off. (b) Error rate v.s. energy trade-off. (c) Energy v.s. time trade-off. In each subplot, we also show the Pareto points from the other two trade-offs.

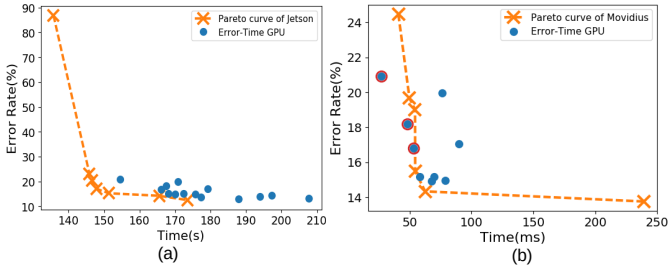

Fig. 9. Evaluating Pareto-optimal models searched for TITAN X GPU on Jetson TX2 (a) and Movidius NCS (b). Dots with red circles denote the new Pareto points. Note that as the Movidius NCS does not support dilated convolutions, we exclude models involving this operation from evaluation on the platform.

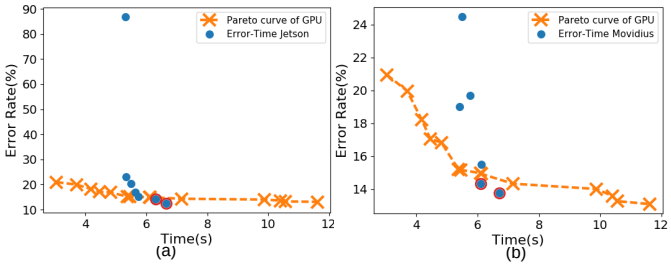

Fig. 10. Evaluating the Pareto-optimal models searched for Jetson TX2 (a) and Movidius NCS (b) on TITAN X GPU. Dots with red circles denote the new Pareto points.

REFERENCES

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[3] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning.," in *AAAI*, 2017, vol. 4, p. 12.

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

[5] X Zhang, X Zhou, M Lin, and J Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices. arxiv 2017," *arXiv preprint arXiv:1707.01083*.

[6] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q Weinberger, "Condensenet: An efficient densenet using learned group convolutions," *group*, vol. 3, no. 12, pp. 11, 2017.

[7] Barret Zoph and Quoc V Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[8] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le, "Learning transferable architectures for scalable image recognition," *arXiv preprint arXiv:1707.07012*, vol. 2, no. 6, 2017.

[9] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy, "Progressive neural architecture search," *arXiv preprint arXiv:1712.00559*, 2017.

[10] TITAN X GPU, *https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x*.

[11] Movidius Neural Compute Stick, *https://developer.movidius.com/*.

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

[13] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le, "Mnasnet: Platform-aware neural architecture search for mobile," *arXiv preprint arXiv:1807.11626*, 2018.

[14] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun, "Dpp-net: Device-aware progressive search for pareto-optimal neural architectures," *arXiv preprint arXiv:1806.08198*, 2018.

[15] Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams, "Predictive entropy search for multi-objective bayesian optimization," .
[16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, Jan 2016.
[17] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.
[18] Tijmen Tieleman and Geoffrey Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.
[19] NVIDIA Management Library (NVML), *https://developer.nvidia.com/nvidia-management-library-nvml*.
[20] Jetson TX2, *https://developer.nvidia.com/embedded/buy/jetson-tx2*.
[21] Lukas Cavigelli, "Convenient power measurements on the jetson tx2/tegra x2 board," 2018.
[22] Intel Movidius Neural Compute SDK, *https://movidius.github.io/ncsdk/*.
[23] Power-Z USB TD Tester, *https://www.unionrepair.com/how-to-use-power-z-usb-pd-tester-voltage-current-type-c-meter-km001/*.