AutoRank: Automated Rank Selection for Effective Neural Network Customization

Mojan Javaheripi, *Graduate Student Member, IEEE*, Mohammad Samragh, *Student Member, IEEE*, and Farinaz Koushanfar, *Fellow, IEEE*

***Abstract*— Tensor decomposition is a promising approach for low-power and real-time application of neural networks on resource-constrained embedded devices. This paper proposes AutoRank, an end-to-end framework for customizing neural network decomposition using cross-layer rank-selection. For many-layer networks, determining the optimal decomposition ranks is a cumbersome task. To overcome this challenge, we establish a state-action-reward system that effectively absorbs inference accuracy and platform specifications into the rank-selection policy. Our proposed framework brings platform characteristics and performance in the customization loop to enable direct incorporation of hardware cost, e.g., runtime and memory footprint. By means of this hardware-awareness, AutoRank customization engine delivers high accuracy decomposed deep neural networks with low execution cost. Our framework minimizes the engineering cost associated with rank selection by providing an automated API for AutoRank that is compatible with popular deep learning libraries and can be readily used by developers.**

***Index Terms*— Computational and artificial intelligence, neural networks, artificial neural networks.**

I. Introduction

With the advancements in data acquisition tools and the growing desire for smart applications, deep neural networks (DNNs) are increasingly adopted in various autonomous platforms, e.g., self-driving cars, drones, and robotics. In order to satisfy the platform resource constraints as well as real-time requirements, a line of research focuses on developing algorithms for model compression with the goal of reducing the computational complexity/memory footprint of DNNs. Examples of explored approaches include using quantized/low-precision weights [1], [2], parameter pruning [3], [4], developing efficient network architectures [5], and tensor decomposition [6]. Each proposed DNN customization method for constrained settings is associated with certain benefits and limitations. For example, pruning can lead to a high model compression rate, but it requires custom hardware accelerators (e.g., FPGA or ASIC) to effectively leverage the existent sparsity patterns in the model [7]. Efficient architectures can benefit execution on general purpose processors [5], but designing such networks for various tasks and datasets incurs a high engineering cost. Tensor decomposition and low-rank approximation of DNN parameters allow for efficient execution on CPU/GPU platforms; nevertheless, determining the optimal approximation ranks that conform to the accuracy requirements and hardware constraints is a standing challenge.

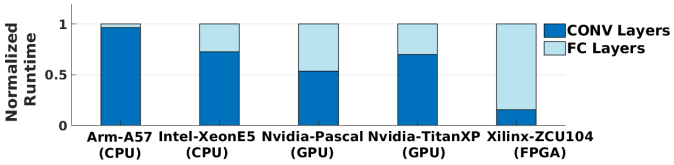

Fig. 1. Normalized runtime breakdown of the AlexNet model evaluated on different platforms.

To ensure an effective DNN compression using tensor decomposition, several challenges need to be addressed. A number of existing methods perform the decomposition one layer at a time, thus requiring per-layer fine-tuning of the network which incurs a significant retraining cost [8]. Authors of [6] propose to perform whole-network compression. Their method overlooks the DNN inter-layer dependencies, which directly affects accuracy. Moreover, the rank-selection strategy must be designed in compatibility with the ultimate goal of deploying the decomposed model on the desired hardware platform. As such, it is essential to take into account the underlying hardware specifications when configuring the decomposition, a concept that is missing in prior works [9], [10]. To illustrate the importance of platform-aware decomposition, Fig. 1 presents the relative runtime associated with convolution (CONV) and fully-connected (FC) layers of the popular AlexNet [11] architecture, measured on different platforms.[1] As seen, the FPGA platform is strictly memory-bounded, hence, FC layers need to be prioritized for decomposition.

This paper proposes AutoRank, an intelligent, automated framework that performs cross-layer customized DNN decomposition for any given network architecture and hardware platform pair. Inspired by Reinforcement Learning, we establish a state-action-reward system that incorporates inference quality and hardware specifications into the rank-selection policy to

Manuscript received April 30, 2021; revised October 18, 2021; accepted October 27, 2021. Date of publication November 10, 2021; date of current version December 13, 2021. This work was supported in part by the National Science Foundation (NSF) under Award 2016737, in part by the Semiconductor Research Corporation (SRC) under Award 2899.001, in part by the Intelligence Advanced Research Projects Activity (IARPA) under Award 2018-18022100004, and in part by the Intel Private AI Institute. This article was recommended by Guest Editor H. Homayoun. *(Corresponding author: Mohammad Samragh.)*

The authors are with the Department of Electrical and Computer Engineering, UC San Diego, San Diego, CA 92093 USA (e-mail: mojan@ucsd.edu; msamragh@ucsd.edu; farinaz@ucsd.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2021.3127433.

Digital Object Identifier 10.1109/JETCAS.2021.3127433

[1]The CPU/GPU measurements are obtained from Pytorch. The FPGA measurement is by ChaiDNN [12] with 8-bit, 128-DSP setting.

ensure a high performance in terms of inference accuracy and runtime. AutoRank outperforms hand-crafted and heuristic rank-selection methods on standard benchmark networks as it achieves a higher compression rate, better preservation of accuracy, and elimination of customization re-engineering. The contributions of AutoRank are as follows:

- Introducing AutoRank, a holistic framework for platform-aware DNN customization by parameter decomposition.
- Devising a resource-profiling scheme to automate the process of platform characterization for DNN decomposition.
- Proposing the first hardware-aware automated policy that performs DNN decomposition by simultaneously optimizing for inference accuracy and hardware performance.
- Development of an API for fast adaptation of AutoRank to smart DNN-based applications.
- Evaluating AutoRank on the challenging ImageNet classification benchmark. Our method achieves an average of $4.88\times$ measured speedup using platform-aware decomposition with an average of 0.62% top-5 accuracy decrease.

II. Related Work

DNNs are known to be over-parameterized, meaning that the trained network architectures can often be compressed without a significant decrease in the inference quality [13], [14]. While such redundancies are beneficial for faster convergence when training the model in high-end cloud servers, executing complex DNN architectures on embedded devices is a challenge. To alleviate this problem, many researchers have worked on various network compaction methodologies [15]. Among these techniques, we focus on tensor decomposition, which is particularly useful since the corresponding compressed DNN does not require specialized hardware, e.g., FPGA or ASIC, to ensure efficiency.

The effectiveness of Tensor decomposition for DNN acceleration has been shown in a line of contemporary research [9], [16]. Authors of [8] propose to utilize CP decomposition to accelerate CONV layers. They propose a one-layer-at-a-time strategy where the DNN is fine-tuned after decomposing each layer. The method proposed in [6] is the most relevant work to AutoRank, where the authors utilize Tucker decomposition to reduce the computations in CONV layers. Their approach follows a one-time, whole-network, compression and re-training strategy. For rank selection, the authors suggest utilizing Variational Bayesian Matrix Factorization, which essentially aims at minimizing the L^2-norm of the tensor reconstruction error. We argue that this approach is sub-optimal since the L^2-norm does not necessarily reflect the inference accuracy of the DNN. In addition, such rank selection is oblivious to the underlying hardware specifications and merely targets the number of computations. To address these challenges, we propose to formulate rank-selection as a state-action-reward system and iteratively assign the ranks in each layer such that maximum hardware performance is obtained with a minimal decrease in the inference accuracy. Our optimization methodology is devised to easily accommodate various hardware platforms and performance metrics.

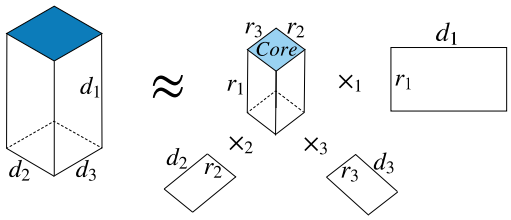

Fig. 2. Illustration of Tucker-3 decomposition for a 3-way tensor. The reconstruction tensor is computed by performing n-mode multiplication in each of the 3 directions [21].

On a separate track of research, reinforcement learning has been used [17] to automate hyper-parameter selection for channel pruning [18].

Similar to [17], we define a reward function for decision making; however, our methodology is different in that AutoRank is an end-to-end framework which directly incorporates hardware cost into the decision policy. In addition, since the state transitions in AutoRank are *deterministic*, our methodology does not require training an RL agent and incurs much lower overhead.

III. Preliminaries

Tensor is a general term used to describe multi-dimensional objects in mathematics. For instance, a scalar, a vector, and a matrix are 0-way, 1-way, and 2-way tensors, respectively. Tensor decomposition can be viewed as a methodology for representing high-dimensional objects as a combination of lower-way tensors. Well-known approaches for generic decomposition include the Canonical Polyadic decomposition (also known as the CANDECOMP/PARAFAC model) [19] and Tucker decomposition [20]. In this paper, we leverage Tucker decomposition to reduce the dimensionality of the trained parameters within conventional DNNs for efficient execution on embedded devices. Tucker decomposition aims to model a d-way tensor by means of smaller orthogonal components. The components, called projection matrices, are interconnected via a core tensor and each component represents one direction of the original tensor. Decomposition is performed with the objective of minimizing the squared error between the original tensor and the reconstruction by small components. Fig. 2 illustrates a Tucker-3 decomposition performed on a 3-way tensor and the resulting core tensor and three projection matrices. A lossy reconstruction of the original tensor is achieved by performing n-mode multiplication (denoted by $\times_n$) [21] between the core tensor and projection matrices where $n \in \{1, 2, 3\}$.

In a conventional DNN, the trainable parameters of CONV layers form 4-way tensors denoted by $\mathbf{W} \in \mathbb{R}^{k\times k\times c\times f}$, where k is the window size, c is the number of input channels, and f is the number of output channels. A generic Tucker decomposition would decompose $\mathbf{W}$ in all 4 ways; However, for the majority of DNNs studied in the literature, the window size (k) is relatively small and therefore decomposing in the k directions does not lead to better performance. As such, we perform a Tucker-2 decomposition in the direction of input (c) and output (f) channels for the CONV layers. By means of

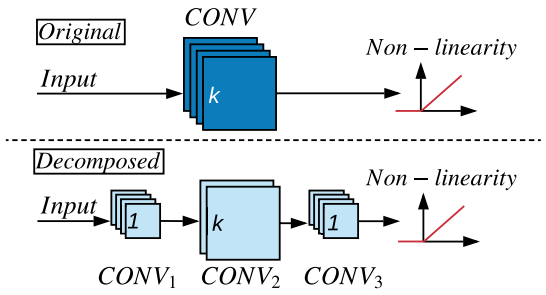

Fig. 3. Illustration of Tucker-2 decomposition on a CONV layer. The computational complexity in the decomposed layer (bottom) is lower than that of the original (top) layer.

such decomposition, a CONV layer can be represented as three consecutive layers depicted in Fig. 3. Here, the core tensor is the middle CONV layer and the first and last CONVs are the point-wise projection matrices.

IV. AutoRank Global Flow

A schematic overview of AutoRank framework is illustrated in Fig. 4. In order to generate a decomposed DNN configuration that is customized to the pertinent hardware specifications, AutoRank sequentially performs three interlinked stages, namely, hardware profiling, automated rank-selection, and fine-tuning. Stages of AutoRank pipeline are specifically designed such that they entirely separate the users from complications of efficient DNN inference on resource-constrained embedded devices.

A. Hardware Profiling

AutoRank utilizes a Hardware Profiling unit (Section V-A) that investigates individual layers by examining a set of pre-defined rank configurations, specifically selected according to the layer's input/output dimensionality. The result of this stage is a list of layer configurations (ranks) and their corresponding runtimes which is later used by the automated policy maker in the Automated Rank-Selection stage.

B. Automated Rank-Selection

AutoRank framework eliminates the engineering cost associated with developing heuristics or performing hand-optimizations by means of an intelligent network modifier, i.e., the Automated Rank-Selection module (Section V-B). This module leverages a novel and fully-automated algorithm primarily based upon capturing the trade-off between inference accuracy and a specific cost measure, e.g., inference runtime. The rank-selection module customizes the decomposition ranks for all layers of the given DNN by iteratively performing four tasks:

(i) The space of future-states is spanned where each state is a possible global rank configuration for DNN layers.
(ii) For each state, the module runs inference on a small subset of unseen data (validation set) to measure accuracy.
(iii) A customized reward is calculated such that it penalizes loss of accuracy and favors cost enhancement, e.g., overall runtime reduction.
(iv) The state rendering the maximum reward is chosen and the pertinent layer is decomposed accordingly. The future search-space is then updated by removing redundant states, i.e., those with a higher cost measure.

As the iterations proceed, the cost measure is decreased at the expense of a decrease in the inference accuracy. The algorithm terminates when the user-provided accuracy threshold is reached. The output of the rank-selection module is a list of customized rank configurations that lead to efficient DNN architectures where the efficiency is described in terms of the pre-defined cost measure. Based on the underlying hardware constraints, this cost measure can be defined as the total number of operations per-inference (FLOPs), runtime, power, or model memory footprint, allowing AutoRank to be compatible with various optimization objectives.

C. Fine-Tuning

Once the cross-layer configuration ranks are selected, the DNN is decomposed accordingly and fine-tuned to recover the accuracy degradation associated with the rank-selection algorithm (Section V-C). In this stage, only a few training epochs are sufficient to restore the accuracy. This incurs much lower computational cost compared to the cost of training the original uncompressed DNN.

V. Methodology

A. Hardware Profiling

An efficient DNN compression scheme is one that takes into account the implications of the compaction method on physical performance. To this end, AutoRank identifies the underlying hardware and incorporates platform-specific metrics into the rank selection policy. Such automated hardware identification allows for a customized rank selection that is specifically tailored to conform to the platform constraints, e.g., computing power, memory bandwidth, or speed (runtime). In order to fully automate such a rank-selection process, AutoRank first gathers abstract information regarding the physical performance during execution of the desired DNN architecture and its decomposed variants. The gathered information is later used in the automated rank-selection module (Section V-B) to customize the decomposed architecture per hardware.

1) Problem Statement: The first step towards hardware profiling is to specify the hardware optimization goal which can take the form of minimizing runtime, memory footprint, or power consumption. We perform hardware profiling by measuring the cost (e.g., runtime) associated with the execution of individual layers in a given DNN as explained below.

2) Per-Layer Performance Identification: For each layer in a given DNN, we quantize the possible decomposition ranks in each direction into b bins. The number of bins directly controls the granularity of the rank selection search-space. A higher number of bins may result in decomposition configurations with similar hardware cost. A lower number of bins enables a faster search but may result in sub-optimal results. We found that $b = 8$ provides a good balance between the aforesaid properties for the benchmarks studied in this paper. For a

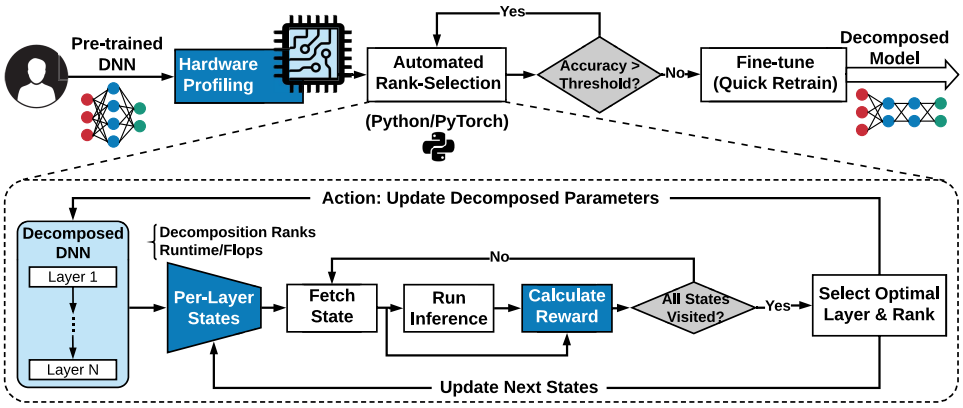

Fig. 4. An overview of AutoRank framework consisting of three sequential stages, namely, hardware profiling, automated rank-selection, and fine-tuning.

CONV layer with c input and f output channels, the quantized rank can have b^2 possiblities:

$$R \in \{(r_c, r_f) | r_c \in \{\frac{c}{b}, \ldots, c\}, r_f \in \{\frac{f}{b}, \ldots, f\}\}. \quad (1)$$

For each of these configurations, AutoRank executes decomposed layer and measures the cost. The gathered information is then forwarded to the automated rank-selection module (Section V-B) as $cost(R^l)$, where the l superscript stands for the layer index.

3) Overhead Analysis: The overhead of hardware profiling is mainly due to the evaluation of the corresponding cost measure per layer. For an L-layer DNN, the search overhead is approximately $b^2 \times L \times t_{avg}$ where b is the number of quantized ranks per direction and t_{avg} is the average time required to compute the output of one DNN layer. Note that there is no need to perform parameter decomposition at this stage as the costs do not depend on parameter values, i.e., random tensors will also result in correct profiling. We emphasize that the timing overhead of AutoRank resource profiling is negligible compared to DNN training.

B. Automated Rank-Selection

The quest for higher accuracy has led moderns DNNs towards deeper architectures that consist of many stacked layers. In order to compress these complex architectures without a significant loss of accuracy, one is required to carefully select the per-layer decomposition ranks. When applying Tucker decomposition, the network compression rate and the inference accuracy are directly determined by the per-layer decomposition ranks, $R^l = (r_c, r_f)$.

DNN decomposition can be particularly cumbersome for sophisticated networks since the space of possibilities grows exponentially with the number of layers. As a result, hand-crafted customization is sub-optimal and tedious. Another approach would be to perform a brute-force search among all possible rank configurations in DNN layers. While this methodology gives the absolute optimal solution, it has an extremely high computational cost and is infeasible in real-life scenarios. One attractive solution to perform an effective search is to utilize Reinforcement Learning (RL) [17]. However, even vanilla RL solutions can be costly and require excessive hardware resources and time to train.

To overcome the aforementioned challenges, we propose an iterative algorithm inspired by RL that aims to determine the above-mentioned ranks across all layers, i.e., $\{R^1, \ldots, R^L\}$ for an L-layer DNN. The objective of this algorithm is to minimize a physical cost measure (e.g. runtime, power, memory footprint, etc.) under a certain inference accuracy requirement. Our proposed automated rank-selection incurs significantly lower cost than pure RL-based solutions while successfully achieving efficient DNN inference on embedded hardware. We formulate the rank customization problem using a state-action-reward system described below.

1) State: A state (S) corresponds to a list of per-layer decomposition ranks, $\{R^1, \ldots, R^L\}$, for all DNN layers. Each state renders a certain accuracy, acc_S, and a certain overall hardware cost, $cost_S = \sum_{l=1}^{L} cost(R^l)$, where the per-layer hardware costs are obtained by resource profiling (Section V-A).

2) Action: An action A_{R^l} decomposes the weights of the l-th layer with the selected ranks R^l. Each action leads to a next state, S', with a certain total cost ($cost_{S'}$) and accuracy ($acc_{S'}$).

3) Reward: To asses an action (A) at a given state (S), we formulate the following reward:

$$reward(A, S) = \frac{cost_S - cost_{S'}}{exp(acc_S - acc_{S'})}. \quad (2)$$

The numerator of this reward function encourages reduction in the underlying hardware cost. The denominator is an exponential term that penalizes decrease in the inference accuracy. As such, the reward function models the ultimate goal of efficient implementation with minimal accuracy degradation.

4) Iterative Rank-Selection: AutoRank performs a step-by-step rank selection by iteratively choosing the optimal actions that render the highest rewards. Algorithm 1 presents a pseudo-code for the proposed rank-selection policy. Initially, all DNN layers are set to have full-rank weight parameters, i.e., no decomposition is applied (line 1). At each step, possible actions are evaluated where each action selects one layer (l) and changes its current rank (R^l) to one of the possible quantized ranks (R^l_{next}, line 11). For each of these actions, the

Algorithm 1 Automated Rank Selection

Inputs: pre-trained DNN model (M), number of per-way decomposition bins (b), minimum accuracy threshold (θ), list of per-layer profiling information ($cost$), validation data and labels ($XY = \{(x_1, y_1), \ldots\}$).

Output: list of rank configurations ($\{S_1, S_2, \ldots\}$) that capture the optimal accuracy-runtime trade-off.

1: $S \leftarrow \{R^1 = (c^1, f^1), \ \ldots, \ R^L = (c^L, f^L)\}$ ▷ Full-ranks
2: $acc_S = Accuracy(M, XY|S)$ ▷ Run inference.
3: $configs \leftarrow \{S\}$
4: **for** $l \in \{1, \ldots, L\}$ **do** ▷ List per-layer quantized ranks.
5: $ranks^l \leftarrow \{(r_c, r_f)|r_c \in \{\frac{c^l}{b}, \ldots, c^l\}, r_f \in \{\frac{f^l}{b}, \ldots, f^l\}\}$
6: **end for**
7: **while** $acc_S > \theta$ **do**
8: **for** $l \in \{1, \ldots, L\}$ **do**
9: **for** $R^l_{next} \in ranks^l$ **do**
10: $A: \ R^l \leftarrow R^l_{next}$
11: $S' \leftarrow \{R^1, \ R^2, \ \ldots, \ R^l = R^l_{next}, \ \ldots, R^L\}$
12: $acc_{S'} = Accuracy(M, XY|S')$
13: $reward(A, S) = \frac{cost(R^l) - cost(R^l_{next})}{exp(acc_S - acc_{S'})}$
14: **end for**
15: **end for**
16: $A^* \leftarrow \arg\max_A \ reward(A, S)$ ▷ Get optimal action.
17: $R^{l^*}_{next} \leftarrow R^l_{next}|A^*$ ▷ Decompose with selected rank.
18: $S = \{R^1, \ R^2, \ \ldots, \ R^l = R^{l^*}_{next}, \ \ldots, R^L\}$
19: $acc_S = Accuracy(M|S)$
20: **for** $R^{l^*}_{next} \in ranks^{l^*}$ **do** ▷ Reduce search space.
21: **if** $cost(R^{l^*}_{next})$ **then**
22: $ranks^{l^*} \leftarrow ranks^{l^*} - \{R^{l^*}_{next}\}$
23: **end if**
24: **end for**
25: $configs \leftarrow \{configs\} + \{S\}$ ▷ Store per-layer ranks.
26: **end while**
27: **return** $configs$

corresponding inference accuracy is measured by evaluating the decomposed DNN on a small subset of validation data. Next, all rewards are calculated using Eq. 2 and the action with the highest reward is applied (line 16). As the iterative algorithm proceeds, both total cost ($cost_S$) and inference accuracy (acc_S) decrease, resulting in a trade-off between inference accuracy and the pre-defined hardware performance measure.

5) Search Space Reduction: At the end of each iteration, all actions resulting in a higher cost than the optimal action are eliminated from the search space (line 22). As such, the search overhead is diminished over iterations. Note that AutoRank does not perform network re-training (parameter fine-tuning) in between the algorithm steps. Therefore, the computational overhead of AutoRank is drastically smaller than that of RL-based techniques.

6) Overhead Analysis: AutoRank requires one-time parameter decomposition for all layers, which is pre-computed before executing Algorithm 1. During the rank selection process, for each evaluation of the reward function, the algorithm runtime is dominated by two main components. First, the weight parameters of the selected layer should be set to the (pre-computed) decomposed parameters (line 11). Second, the inference accuracy should be computed for a small batch of data. Both of the aforementioned operations incur negligible computational overhead compared to DNN training.

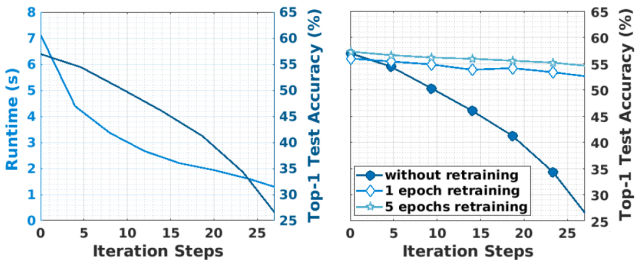

Fig. 5. Left: inference accuracy and runtime trade-off versus the iteration steps. Right: effect of re-training on the validation accuracy for different configurations.

C. Fine-Tuning

As discussed in Section V-B, the output of our rank-selection policy is a set of per-layer rank configurations, each of which is associated with a certain physical cost and inference accuracy. Fig. 5 (left) demonstrates the accuracy and runtime for different rank configurations obtained by applying Algorithm 1 to AlexNet architecture. As mentioned earlier, to minimize the search overhead, no re-training is performed amidst algorithm iterations. Upon completion of the rank-selection procedure, AutoRank re-trains the chosen decomposed DNN configuration to restore the accuracy loss. Fine-tuning is performed by means of standard back-propagation routines. In Fig. 5 (right), we show that the inference accuracy can be restored with minimal degradation by fine-tuning the selected decomposed DNN for as few as 1 epoch. As we show in our experiments, the accuracy that is achieved immediately after customization is highly correlated with the one after fine-tuning.

VI. Experiments

To corroborate the effectiveness of our proposed methodology, we evaluate two well-known DNN architectures, namely, Alexnet (5 CONV and 3 FC layers) and VGG-16 (13 CONV and 3 FC layers). Our experiments are performed on the ISLVRC-12 (ImageNet) visual dataset consisting of 1000 classes. We use Pytorch for DNN description and fine-tuning. For Tucker decomposition, we utilize the Tensorly package [22]. The first step for evaluating AutoRank is to extract hardware-specific information by means of the hardware profiling tool explained in Section V-A. We randomly sample 1000 images from the ImageNet validation data, which are then fed to the rank-selection algorithm together with the extracted hardware profiling reports. The minimum (top-1) accuracy threshold (θ in Algorithm 1) is set to 25% and the number of per-mode quantized ranks (b in Algorithm 1) are set to 8 in our experiments. Once the decomposed DNN is

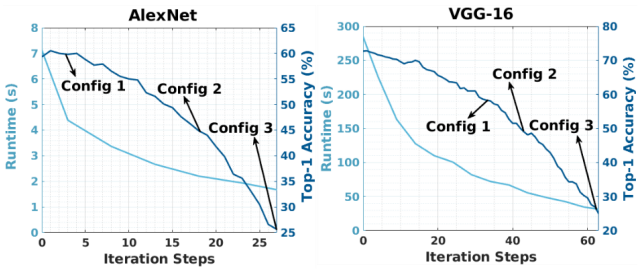

Fig. 6. Trade-off curve between runtime and accuracy obtained by Algorithm 1 for AlexNet (left) and VGG-16 (right). The reported accuracies are achieved without fine-tuning. Runtimes are measured for single image inference on ARM-A57 CPU. Arrows show the selected configurations to be fine-tuned.

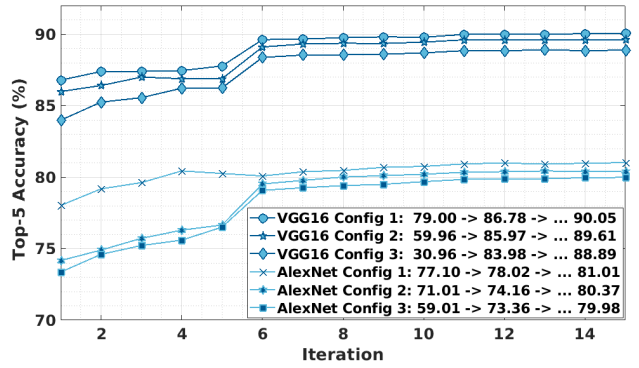

Fig. 7. Fine-tuning curves of selected configurations in Fig. 5.

configured, we deploy the model on an embedded board with an ARM-A57 processor to measure runtime and power.

A. Runtime and Accuracy Analysis

We run AutoRank algorithm with the cost in Eq. 2 set to measured runtime. The result of the algorithm is a set of per-layer rank configurations that capture the trade-off between runtime and accuracy as shown in Fig. 6. The accuracy of AutoRank decomposed DNNs can be further improved by fine-tuning, which takes place after Algorithm 1. To show this effect, we select several candidate configurations (shown by arrows) and fine-tune the corresponding DNNs for 15 epochs. The training curves are presented in Fig. 7. We note that the final accuracy is correlated with the initial validation accuracy. Such correlation allows us to run AutoRank without fine-tuning the model throughout the search iterations, ensuring a fast rank-selection process.

1) Effect of Hardware Profiling: It is often observed in prior work [6], [23] that the measured speedup on compressed DNNs is lower than the expected "theoretical" speedup defined based on the number of floating-point operations (FLOPs) required for inference. An interesting property of AutoRank is that it achieves higher measured speedup compared to what is expected in theory. Fig. 8 shows the two speedups for the evaluated benchmarks. AutoRank directly incorporates implicit hardware-related factors (e.g., memory access) that impact runtime rather than solely relying on the number of FLOPs. This hardware-aware customization greatly boosts the

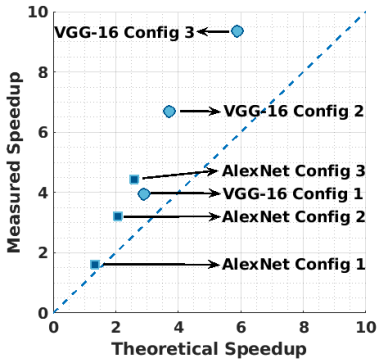

Fig. 8. Theoretical and measured speedups. AutoRank customized models achieve higher actual speedup, compared to theory, in all benchmarks.

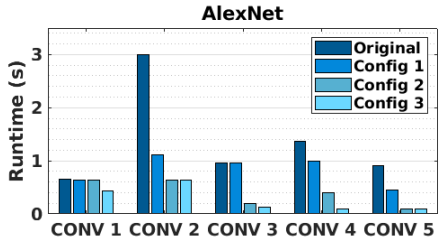

Fig. 9. breakdown of runtime for AlexNet CONV layers.

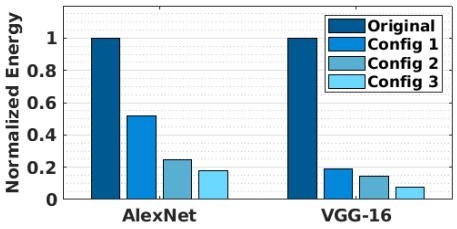

Fig. 10. Normalized energy for AlexNet and VGG-16.

performance on constrained processors such as the evaluated CPU.

2) Discussion: To qualitatively study AutoRank and provide insights, we show the per-layer runtime of the AlexNet selected configurations in Fig. 9. In the original model (the dark bars), the second layer (CONV-2) renders the highest runtime. AutoRank automatically detects this characteristic at the early stages of the iterative rank selection, resulting in Config 1. Unlike the baseline model which has a high variation in layer runtimes, AutoRank tends to enforce a uniform runtime across layers, which can facilitate pipelining the layers for a higher throughput. After Config 1 is obtained, AutoRank iteratively advances to Config 2 and 3. Our algorithm automatically detects that CONV 1 and CONV 2 contribute significantly to the inference accuracy; therefore, to maintain the inference accuracy, AutoRank opts not to further compress the first two layers and decomposes the deeper layers instead.

B. Power and Energy Analysis

To examine the energy efficiency of AutoRank runtime-optimized configurations, we measure the power consumption

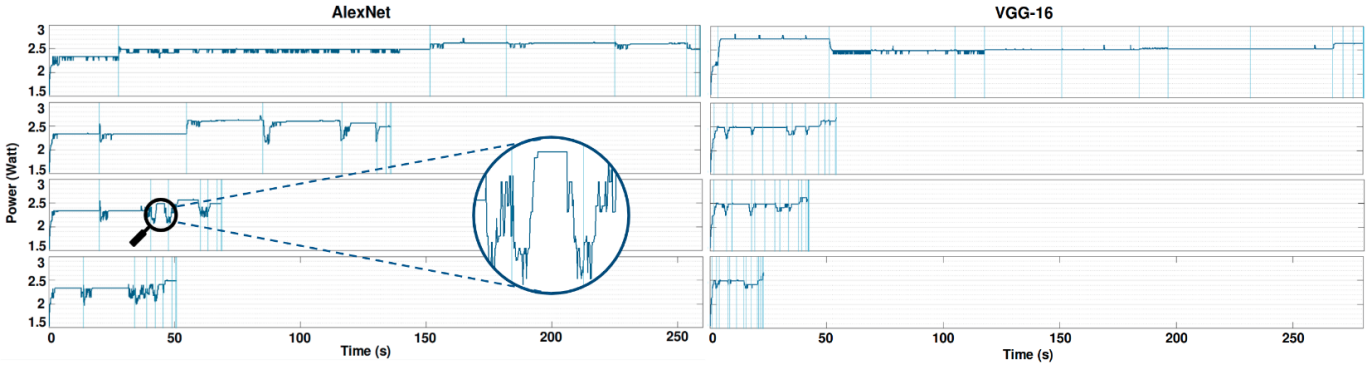

Fig. 11. Power monitoring for AlexNet (left) and VGG-16 (right). The top row shows power curves for the original uncompressed model. The bottom three rows correspond to Config-1 through 3, respectively. In this example, the AlexNet model runs a batch of 32 images while the VGG-16 model performs single image inference.

TABLE I
SUMMARY OF SELECTED RUNTIME-OPTIMIZED MODELS

Network	Fine-tuning Epochs		Top-5 acc (%) 0	1	15	FLOPs ($\times 10^8$)	Runtime (s)	Theoretical Speedup	Measured Speedup
AlexNet	**Baseline**			81.05		7.14	7.13	-	-
	AutoRank	Config 1	77.10	78.02	81.01	5.43	4.39	1.31×	1.62×
		Config 2	71.01	74.16	80.37	3.49	2.21	2.04×	3.23×
		Config 3	59.01	73.36	79.98	2.77	1.61	2.58×	4.43×
	Kim et al. [6]		23.39	74.74	78.33	2.72	2.11	2.63×	3.37×
VGG-16	**Baseline**			90.1		138.3	285	-	-
	AutoRank	Config 1	79.00	86.78	90.05	49.5	72.15	2.79×	3.95×
		Config 2	59.96	85.97	89.61	31.7	42.55	4.36×	6.70×
		Config 3	30.96	83.98	88.89	23.6	30.48	5.86×	9.35×
	Kim et al. [6]		34.06	78.68	89.4	31.4	42.86	4.40×	6.64×

of Jetson TX2 embedded board during model inference. Measurements are obtained from three rails corresponding to CPU, SOC, and Memory. Fig. 10 summarizes AutoRank energy usage (normalized by the uncompressed model's energy). To provide a more detailed analysis, we present the instantaneous power consumption in Fig. 11. Here, the start and end times for execution of DNN layers are shown by the vertical lines on each figure. As seen in the magnified curve, decomposed convolutions at the beginning and end of layer execution have lower power consumption. This is due to the lower cost of point-wise (1×1) convolutions (corresponding to $CONV_1$ and $CONV_3$ in Fig. 3) that take place in decomposed layers. This effect along with lower overall runtime allows AutoRank to achieve significant energy saving as shown in Fig. 10.

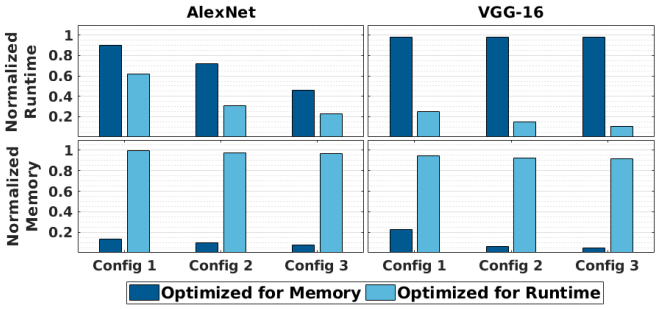

Fig. 12. Comparison between memory-oriented (dark blue) and runtime-oriented (light blue) rank configurations. The latter achieves better runtime while the former gains higher memory compression. At each Config X, the corresponding pairs have equivalent classification accuracy.

C. Memory Analysis

Similar to our runtime-oriented rank-selection in Fig. 6, we generate a set of configurations for memory-optimized decomposition. In this case, the cost in the denominator of Eq. 2 is defined as the total number of parameters in the weight tensors which is directly translated to the pertinent memory footprint. We then select three candidate configurations that achieve the same level of accuracy as the run-time optimized configurations selected. Fig. 12 presents the comparison between the acquired memory/runtime-optimized AutoRank models in terms of runtime and memory. As can be seen, the memory-oriented optimization achieves a higher compression rate than the runtime-based policy; however, the ranks selected for memory compression lead to a high runtime. In this case, we observe that AutoRank aggressively targets fully-connected layers and approximates them with lower ranks to save memory, but does not apply severe decomposition to the compute-intensive convolution layers. This experiment further validates the effectiveness of hardware-aware optimization. It also demonstrates the automatic adaptation of AutoRank compression policy to the underlying hardware cost by means of our reward function (Eq. 2).

D. Summary and Comparison With Prior Art

Table I summarizes the corresponding runtime improvement and reduction in FLOPs achieved for the AlexNet and

TABLE II

RANK CONFIGURATIONS FOR DECOMPOSED ALEXNET (TOP) AND VGG-16 (BOTTOM) PRODUCED BY AUTORANK. HERE, **C** DENOTES A CONV LAYER, $(\mathbf{r_c}, \mathbf{r_f})$ PRESENTS THE DECOMPOSITION RANKS, AND A DASH MEANS THAT THE LAYER IS NOT DECOMPOSED IN THAT DIRECTION

	AlexNet				
	C1	C2	C3	C4	C5
Config 1	-	(32,-)	-	(288,224)	(160,192)
Config 2	-	(24,144)	(96,240)	(144,192)	(96,96)
Config 3	(-,40)	(24,144)	(72,192)	(96,96)	(96,96)

	VGG16												
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
Config 1	(-,16)	(24,16)	(24,48)	(48,64)	(80,128)	(96,96)	(96,128)	(96,320)	(192,256)	(192,256)	(320,448)	(320,320)	(348,348)
Config 2	(-,16)	(24,16)	(24,48)	(48,64)	(32,96)	(96,96)	(96,96)	(96,256)	(128,256)	(192,192)	(192,384)	(320,320)	(192,192)
Config 3	(-,16)	(8,16)	(16,32)	(32,48)	(32,64)	(64,96)	(64,96)	(64,192)	(128,128)	(128,192)	(128,192)	(192,256)	(192,192)

VGG16 networks, respectively. The corresponding decomposition ranks for each of the configurations are enclosed in Table II. We report our top-5 test accuracy for candidate configurations (shown in Fig. 6) prior to fine-tuning, after 1 epoch of re-training, and after 15 epochs of re-training. For AlexNet and VGG-16 networks, AutoRank achieves up to $4.43\times$ and $9.35\times$ reduction in runtime, with a respective top-5 accuracy degradation of less than 1.07% and 1.21%.

To better demonstrate the effect of cost-aware rank selection, we compare AutoRank with the original Tucker decomposition methodology [6]; in that work, the ranks are selected such that the energy of per-layer tensor approximations is higher than a pre-defined value. While the aforementioned method can achieve a low reconstruction error for single-layer tensor approximation, it does not model the inter-layer correlation for whole-network compression, resulting in low accuracy rates immediately after compression (corresponding to 0 fine-tuning epoch in Table I). As seen, [6] reduces the top-5 accuracy of AlexNet and VGG-16 to 23.39% and 34.06%, respectively. AutoRank, on the other hand, allows us to preserve this accuracy even without fine-tuning: in Config 1, we can obtain 77.10 top-5 accuracy and 1.62 runtime improvement for AlexNet. Similarly for VGG-16, we achieve 3.95 runtime improvement with a top-5 accuracy of 79.00 in Config 1.

We can fine-tune the models to improve the accuracy. We achieve faster training convergence compared to the prior work. For example, considering the fine-tuning of VGG-16 in Config 2, AutoRank achieves 85.97% top-5 accuracy after 1 epoch and eventually obtains 89.61% after 15 epochs, whereas [6] achieves 78.68% and 89.4% after 1 and 15 epochs, respectively. This is a direct result of starting the fine-tuning stage from a good initial state with carefully-selected ranks that do not severely impact the original classification accuracy.

E. Algorithm Overhead

On a personal computer with an Intel-Xeon-5 CPU and Nvidia-Titan-XP GPU, our search policy takes ($\approx$ 1 GPU-hour) for AlexNet and ($\approx$ 18 GPU-hours) for VGG-16. On the same machine, training the original uncompressed DNNs takes ($\approx$ 27 GPU-hours) for AlexNet (assuming 80 epochs for convergence [11]), and ($\approx$ 225 GPU-hours) for VGG-16 architecture (assuming 75 epochs for convergence [24]). Therefore, the overhead of AutoRank customization algorithm is quite negligible compared to the training cost: $\sim 3.7\%$ and $\sim 8\%$ for AlexNet and VGG-16, respectively. Furthermore, we emphasize that AutoRank extracts *multiple* configurations, i.e, 33 for AlexNet and 63 for VGG-16 in our experiments, each of which represents a specific accuracy and hardware constraint. Thus, the amortized (per-configuration) computational costs of AutoRank are much lower than the above-mentioned values.

VII. CONCLUSION

This paper proposes a fully automated framework for hardware-aware compression of DNNs via tensor decomposition. We devise an intelligent rank-selection module that adaptively selects the best configuration of decomposition ranks across DNN layers such that the decomposed model optimally executes on a desired hardware platform. Our automated rank-selection engine accommodates various embedded hardware constraints, e.g., runtime, memory footprint, and power. In order to efficiently model the limitations of the resource-constrained platforms, we leverage a hardware profiling module which generates performance reports to be used for policy making by the rank-selection engine. AutoRank automated rank-selection incorporates a state-action-reward scheme inspired by RL to select several configuration of per-layer ranks, each of which demonstrate a certain trade-off between model accuracy and compression rate. AutoRank achieves higher practical performance on different DNN architectures compared to prior work. Our evaluations on the challenging ImageNet dataset together with our measurements from an embedded processor fully corroborate the effectiveness of AutoRank methodology.

REFERENCES

[1] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," 2016, *arXiv:1609.07061*.

[2] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[3] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 806–814.

[4] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. NIPS*, 2016, pp. 2074–2082.

[5] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[6] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*.
[7] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.
[8] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," 2014, *arXiv:1412.6553*.
[9] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.
[10] C. Tai, T. Xiao, Y. Zhang, X. Wang, and E. Weinan, "Convolutional neural networks with low-rank regularization," 2015, *arXiv:1511.06067*.
[11] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[12] *ChaiDNN: HLS Based Deep Neural Network Accelerator Library for Xilinx Ultrascale + Mpsocs*. Accessed: Sep. 2018. [Online]. Available: https://github.com/Xilinx/CHaiDNN
[13] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, *arXiv:1207.0580*.
[14] M. Denil *et al.*, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.
[15] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017, *arXiv:1710.09282*.
[16] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014, *arXiv:1405.3866*.
[17] Y. He, J. Lin, Z. Liu, H. Wang, L. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. ECCV*, 2018, pp. 784–800.
[18] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, vol. 2, no. 6, pp. 1389–1397.
[19] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009.
[20] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.
[21] C. D. M. Martin, "Tensor decompositions workshop discussion notes American Institute of Mathematics (AIM) Palo Alto, CA," 2004, pp. 19–23.
[22] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, "Tensorly: Tensor learning in Python," 2018, *arXiv:1610.09555*.
[23] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning," in *Proc. IJCAI*, 2018, pp. 2425–2432.
[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

Mojan Javaheripi (Graduate Student Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology in 2017. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of California at San Diego. Her research interests include co-developing machine learning algorithms and their corresponding specialized hardware with the goal of maximizing efficiency and performance.

Mohammad Samragh (Student Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology and the Ph.D. degree in electrical and computer engineering from the University of California at San Diego. His research interests include deep learning customization, safety and robustness of machine learning algorithms, hardware acceleration of learning algorithms, and privacy-preserving inference.

Farinaz Koushanfar (Fellow, IEEE) received the B.S. degree in electrical engineering from the Sharif University of Technology, the M.S. degree in electrical engineering from UCLA, and the M.A. degree in statistics and the Ph.D. degree in electrical engineering and computer science from UC Berkeley. She is currently a Professor and a Henry Booker Faculty Scholar of electrical and computer engineering with the University of California at San Diego, where she is directing the Adaptive Computing and Embedded Systems (ACES) Laboratory. Before joining the faculty at UCSD, she was a Professor of electrical and computer engineering at Rice University. Her research interests include embedded and cyber-physical systems design, embedded systems security, and design automation of domain-specific/mobile computing and machine learning applications. She is a fellow of the Kavli Foundation Frontiers of the National Academy of Engineering. She has received a number of awards and honors for her research, mentorship, and teaching, including the Presidential Early Career Award for Scientists and Engineers (PECASE) from the President Obama, the ACM SIGDA Outstanding New Faculty Award, MIT TR-35, and the Young Faculty/CAREER Awards from NSF, DARPA, ONR, and ARO.