AutoRank: Automated Rank Selection for Effective Neural Network Customization

Mohammad Samragh, Mojan Javaheripi, Farinaz Koushanfar
Department of Electrical and Computer Engineering, UC San Diego
{msamragh,mojan,farinaz}@ucsd.edu

ABSTRACT

Tensor decomposition is a promising approach for low-power and real-time realization 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 an intelligent agent based on 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 *learns* how to deliver 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 that is compatible with popular deep learning libraries.

1. INTRODUCTION

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. 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 [1]. Authors of [2] 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 [3,4].

This paper proposes AutoRank, an intelligent, automated framework that learns how to perform cross-layer DNN decomposition that is specifically customized for any given network architecture and hardware platform pair. We establish a state-action-reward system that incorporates inference quality and hardware specifications into the rank-selection policy to ensure a high performance in terms of accuracy and runtime. AutoRank outperforms hand-crafted and heuristic rank-selection methods on standard benchmark networks as it achieves a higher compression rate, better accuracy preservation, and elimination of customization re-engineering. The contributions of AutoRank are as follows: (1) Introducing AutoRank, a holistic learning-based framework for platform-aware DNN customization by parameter decomposition. (2) Devising a resource-profiling scheme to automate the process of platform characterization for DNN decomposition. (3) Proposing the first hardware-aware automated policy that performs DNN decomposition by simultaneously optimizing for inference accuracy and hardware performance. (4) Development of an API for fast adaptation of AutoRank to smart DNN-based applications. (5) Evaluating AutoRank on the challenging ImageNet classification benchmark. Our method achieves an average of 4.88× measured speedup using platform-aware decomposition with an average of 0.62% top-5 accuracy decrease.

2. PRELIMINARIES

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. In this paper we approximate each 4-way weight using Tucker-2 decomposition. We refer the reader to [2] for detailed explanation of Tucker-2 decomposition and their usage in DNN compression. By means of such decomposition, a CONV layer can be represented as three consecutive layers depicted in Fig. 1. The size of these convolution layers is determined by the approximation ranks. Larger ranks result in lower approximation error but come at the cost of a higher computational complexity. To utilize Tucker-2 decomposition for DNN compression, one needs to specify 2 approximation ranks for each convolutions layer. Finding the combination of ranks across all layers such that the overall execution cost s minimized and the inference accuracy is maximized is extremely challenging. In this paper, we select these ranks in an intelligent and automated manner.

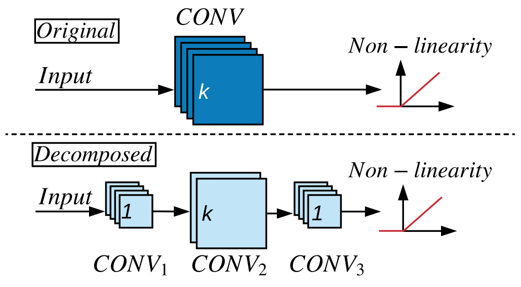

Figure 1: Tucker-2 decomposition for a CONV layer.

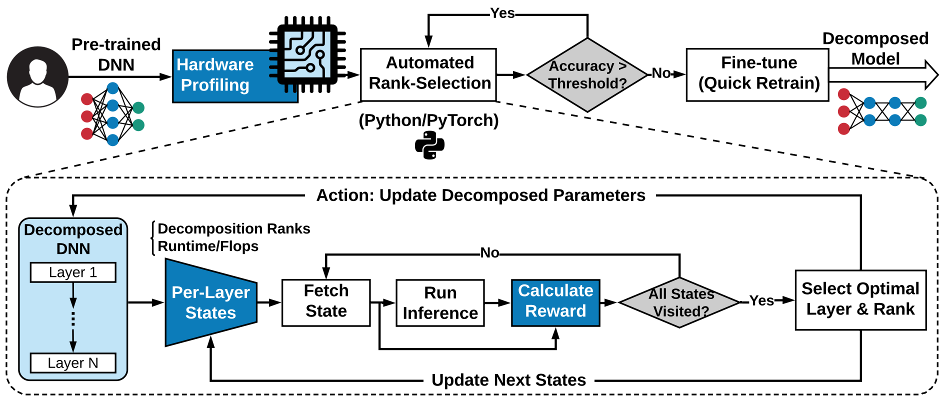

Figure 2: An overview of AutoRank framework consisting of three stages, namely, hardware profiling, automated rank-selection, and fine-tuning.

3. AutoRank GLOBAL FLOW

A schematic overview of AutoRank framework is illustrated in Fig. 2. 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.

3.1 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. This information is later used in the automated rank-selection module (Section 3.2) to customize the ranks per hardware.

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.

Per-Layer Performance Identification. For each DNN layer, we quantize the possible decomposition ranks in each direction into b bins. For a CONV layer with c input and f output channels, the quantized rank can have b^2 possiblities:

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

AutoRank executes the decomposed layer corresponding to each configuration and measures the cost, and uses this information in automated rank-selection (Section 3.2).

3.2 Automated Rank-Selection

When applying Tucker decomposition, the network compression rate and the inference accuracy are directly determined by the per-layer decomposition ranks, $R^l = (r_c, r_f)$. We propose an iterative algorithm that aims to determine the above-mentioned ranks across all layers, i.e., $\{R^1, \dots, R^L\}$ 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 Reinforcement Learning (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.

State: A state (S) corresponds to a list of per-layer decomposition ranks, $\{R^1, \dots, R^L\}$, 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 3.1).

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 ($acc_{S'}$).

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.

Iterative Rank-Selection. 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 corresponding inference accuracy is measured by evaluating

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 ($XY = \{(x_1, y_1), \dots\}$).
Output: list of rank configurations ($\{S_1, S_2, \dots\}$) that capture the optimal accuracy-runtime trade-off.

```
1:  $S \leftarrow \{R^1 = (c^1, f^1), \dots, R^L = (c^L, f^L)\}$        ▹ Full-rank layers.
2:  $acc_S = Accuracy(M, XY|S)$        ▹ Run inference
3:  $configs \leftarrow \{S\}$
4:  for $l \in \{1, \dots, L\}$ do        ▹ list per-layer quantized ranks
5:      $ranks^l \leftarrow \{(r_c, r_f) | r_c \in \{\frac{c^l}{b}, \dots, c^l\}, r_f \in \{\frac{f^l}{b}, \dots, f^l\}\}$
6:  end for
7:  while $acc_S > \theta$ do
8:      for $l \in \{1, \dots, L\}$ do
9:          for $R^l_{next} \in ranks^l$ do
10:             $A: R^l \leftarrow R^l_{next}$
11:             $S' \leftarrow \{R^1, R^2, \dots, R^l = R^l_{next}, \dots, R^L\}$
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:     $S = \{R^1, R^2, \dots, R^l = R^{l^*}_{next}, \dots, R^L\}$
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 $configs$
```

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). At each iteration, both total cost ($cost_S$) and inference accuracy (acc_S) decrease, resulting in a trade-off between inference accuracy and the hardware performance measure. 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.

3.3 Fine-tuning

To minimize the search overhead, no re-training is performed amidst AutoRank iterations. Upon completion of Algorithm 1, AutoRank re-trains the chosen decomposed DNN configuration to restore the accuracy loss. Fine-tuning is performed by standard back-propagation routines. As we show in our experiments, the accuracy that is achieved immediately after customization is highly correlated with the one after fine-tuning.

4. 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 [5]. The first step for evaluating AutoRank is to extract hardware-specific information by means of the hardware profiling tool explained in Section 3.1. 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 configured, we deploy the model on an embedded board with an ARM-A57 processor to measure runtime and power.

4.1 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. 3. 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. 4. 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.

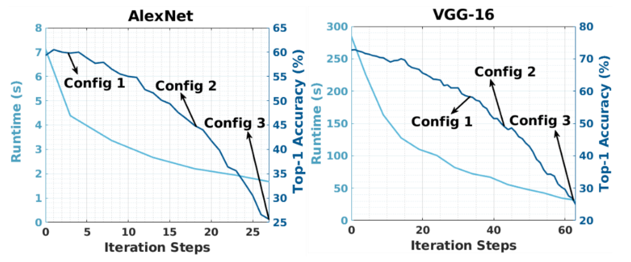

Figure 3: Trade-off curve between runtime and accuracy obtained by Algorithm 1 for AlexNet (left) and VGG-16 (right). Accuracies are reported before fine-tuning. Runtimes are measured for single image inference on ARM-A57 CPU. Arrows show the selected configurations to be fine-tuned.

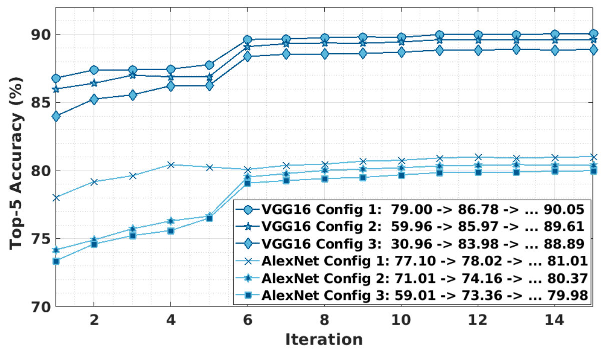

Figure 4: Fine-tuning selected configurations of Fig. 3.

Effect of hardware profiling. It is often observed in prior work [2,6] 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. 5 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 performance on constrained processors.

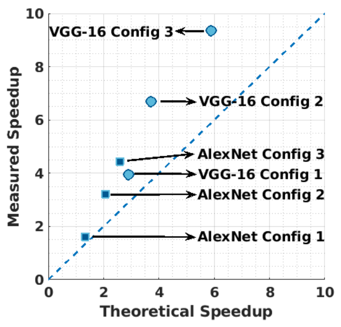

Figure 5: Theoretical and measured speedups. AutoRank customized models achieve higher actual speedup, compared to theory, in all benchmarks.

4.2 Power and Energy Analysis

To examine the energy efficiency of AutoRank runtime-optimized configurations, we measure the power consumption of Jetson TX2 embedded board during model inference. Measurements are obtained from three rails corresponding to CPU, SOC, and Memory. Fig. 6 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. 7. 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. 1) that take place in decomposed layers. This effect along with lower overall runtime allows AutoRank to achieve significant energy saving as shown in Fig. 6.

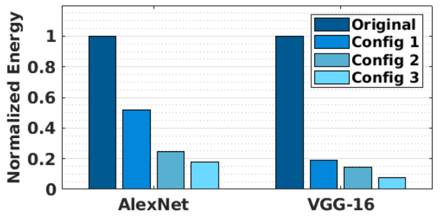

Figure 6: Normalized energy for AlexNet and VGG-16.

4.3 Memory Analysis

Similar to our runtime-oriented rank-selection in Fig. 3, 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. 8 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).

4.4 Summary and Comparison with Prior Art

Table 1 summarizes the corresponding runtime improvement and reduction in FLOPs achieved for the AlexNet and VGG16 networks, respectively. We report our top-5 test accuracy for candidate configurations (shown in Fig. 3) prior to fine-tuning, after 1 epoch of re-training, and after 15 epochs. For AlexNet and VGG-16, 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 [2]; 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 1). As seen, [2] 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 the accuracy prior to fine-tuning: in Config 1, we obtain 77.10% top-5 accuracy and $1.62\times$ runtime improvement for AlexNet. Similarly for VGG-16, we achieve $3.95\times$ speedup with 79.00% top-5 accuracy 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 [2] achieves 78.68% and 89.4% after 1 and 15 epochs, respectively. This is a direct result of starting the fine-tuning from a good initial state with carefully-selected ranks that do not severely impact the accuracy.

5. RELATED WORK

The effectiveness of Tensor decomposition for DNN acceleration has been shown in a line of contemporary research [3, 7]. Authors of [1] 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 [2] 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,

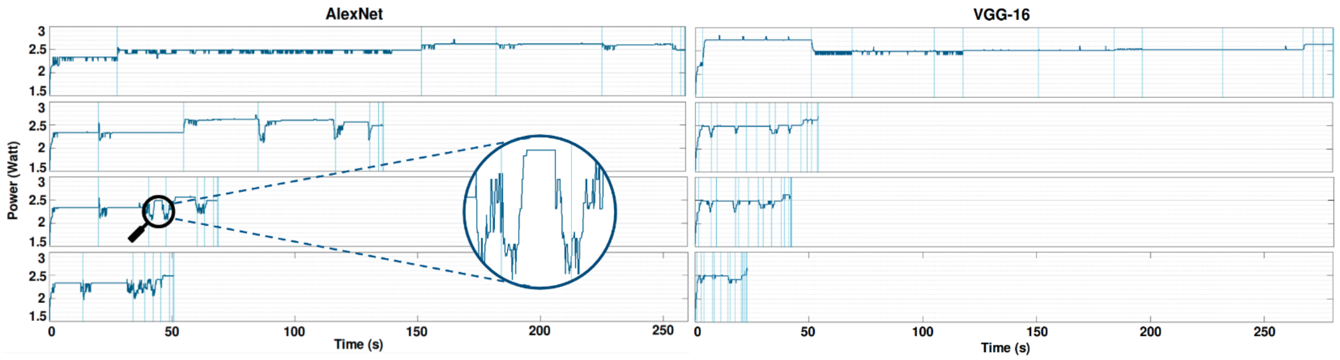

Figure 7: Power monitoring for AlexNet (left) and VGG-16 (right). The top row shows power curves for the original uncompressed model. The bottom three rows correspond to Config-1 through 3, respectively. In this example, the AlexNet model runs a batch of 32 images while the VGG-16 model performs single image inference.

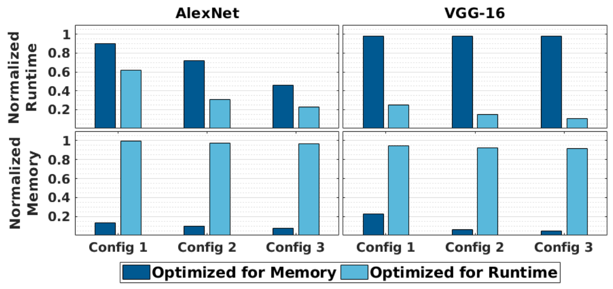

Figure 8: Comparison between memory-oriented (dark blue) and runtime-oriented (light blue) rank configurations. The latter achieves better runtime while the former gains higher memory compression. At each Config X, the corresponding pairs have equivalent classification accuracy.

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. AutoRank, on the other hand, directly incorporates inference accuracy and hardware execution cost in the rank-selection policy.

On a separate track of research, reinforcement learning has been used [8] to automate hyper-parameter selection for channel pruning [9]. Similar to [8], 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.

6. 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.

Table 1: Summary of selected runtime-optimized models.

Network	Fine-tuning Epochs		Top-5 acc (%)			FLOPs ($\times 10^8$)	Runtime (s)	Theoretical Speedup	Measured Speedup
			0	1	15				
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. [2]		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. [2]		34.06	78.68	89.4	31.4	42.86	4.40×	6.64×

7. REFERENCES

[1] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[2] Y. 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," *arXiv preprint arXiv:1511.06530*, 2015.

[3] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, pp. 1269–1277, 2014.

[4] C. Tai, T. Xiao, Y. Zhang, X. Wang, *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.

[5] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, "Tensorly: Tensor learning in python," *CoRR*, vol. abs/1610.09555, 2018.

[6] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning.," in *IJCAI*, pp. 2425–2432, 2018.

[7] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[8] Y. He, J. Lin, Z. Liu, H. Wang, L. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *ECCV*, pp. 784–800, 2018.

[9] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *International Conference on Computer Vision (ICCV)*, vol. 2, 2017.