CompressNAS : A Fast and Efficient Technique for Model Compression using Decomposition

Sudhakar Sah†, Nikhil Chabbra†, Matthieu Durnerin †
† STMicroelectronics

Abstract

Deep Convolutional Neural Networks (CNNs) are increasingly difficult to deploy on microcontrollers (MCUs) and lightweight NPUs (Neural Processing Units) due to their growing size and compute demands. Low-rank tensor decomposition, such as Tucker factorization, is a promising way to reduce parameters and operations with reasonable accuracy loss. However, existing approaches select ranks locally and often ignore global trade-offs between compression and accuracy. We introduce *CompressNAS*, a MicroNAS-inspired framework that treats rank selection as a global search problem. *CompressNAS* employs a fast accuracy estimator to evaluate candidate decompositions, enabling efficient yet exhaustive rank exploration under memory and accuracy constraints. In ImageNet, *CompressNAS* compresses ResNet-18 by 8× with less than 4% accuracy drop, in COCO, we achieve 2× compression of YOLOv5s without any accuracy drop and 2× compression of YOLOv5n with 2.5% drop.

Keywords: TinyML, Lightweight CNNs, EdgeAI, Model ompression

1. Introduction

Deep convolutional neural networks (CNNs) have become the backbone of modern computer vision, powering applications ranging from classification and detection to segmentation. Yet, these accuracy gains are obtained at the expense of rapidly increasing model depth, parameter count, and computational demand. This poses a major obstacle for deployment on resource-constrained platforms such as microcontrollers (MCUs) and lightweight neural processing units (NPUs), where limited memory, strict energy budgets, and real-time constraints dominate [1, 2]. Addressing this mismatch between model complexity and hardware constraints has spurred extensive research in neural network compression. Classical compression methods include pruning redundant weights or filters [3, 4], quantizing weights and activations to low precision [5, 6], and distilling knowledge from large teacher models into smaller students [7]. Although effective, these approaches often require costly retraining and may not guarantee stable accuracy under strict deployment budgets. An alternative line of work leverages low-rank tensor decomposition to directly factorize convolutional kernels, thereby reducing multiply–accumulate (MAC) operations and memory footprint. Pioneering methods based on Canonical polyadic (CP), Tucker, and tensor-train (TT) decompositions have demonstrated promising results for compressing convolution layers with minimal accuracy loss [8–11]. Among these, Tucker decomposition has emerged to be particularly attractive due to its ability to control ranks across different tensor modes, balancing compression and expressiveness [12].

A central challenge in decomposition-based compression is selecting appropriate ranks for each layer. Current pipelines often rely on analytic estimators such as Empirical Variational Bayes Matrix Factorization (EVBMF) [13] or heuristics based on local reconstruction error [8, 10]. While lightweight, these estimators optimize local approximation fidelity rather than global task performance, and they do not naturally account for inter-layer dependencies or

*sudhakar.sah@gmail.com

end-to-end constraints such as latency or memory budgets. This gap motivates search-based strategies that can identify globally consistent rank configurations.

Recent advances in neural architecture search (NAS) suggests to treat rank selection as an architectural design problem. NAS methods have shown success in identifying compact architectures under hardware-aware objectives [14, 15]. In parallel, zero-cost (ZC) proxies have been developed to estimate network accuracy without training by using gradient- or saliency-based measures from a single forward or backward pass [16–18]. These proxies correlate strongly with final trained accuracy, and recent work has explored ensembling and symbolic-regression approaches to further improve their predictive power [19–21]. Despite this progress, their application to rank selection in tensor decompositions remains underexplored.

This paper proposes *CompressNAS*, a microNAS framework for Tucker decomposition–based CNN compression. *CompressNAS* formulates rank selection as a global search problem. Our contributions are:

- We introduce a lightweight estimator to approximate the decomposition impact on accuracy and compression ratio enabling fast evaluation without finetuning.
- We develop an efficient exhaustive search technique that identifies globally consistent rank configurations under model size and accuracy constraints, suitable for MCU/NPU deployment.
- We propose an Integer Linear Programming (ILP) search based NAS given the hardware budget.

To the best of our knowledge, prior work on decomposition has not reported extreme compression (order of 10×) for classification models such as ResNet, nor any promising results for detection models such as YOLOv5.

2. Related Work

Tensor Decomposition for CNN Compression. Recent work has renewed interest in tensor factorizations for compact CNNs. Beyond classical CP/TT approaches, Tucker and its variants have seen steady progress. Knowledge-Based Systems (2022) introduced Tucker with nonlinear response modeling for accuracy-aware compression [22]. Hierarchical Tucker-2 (HT-2) further reduces storage by recursively factorizing the core, improving efficiency on ImageNet-scale models [23]. Concurrently, alternating optimization focussing under Tucker was proposed to combine fine-tuning with decomposition steps [24]. Complementary efforts studied hardware-aware acceleration pipelines for Tucker-compressed CNNs [25] and training-in-the-low-rank space to stabilize optimization [26].

Automatic and Budget-Aware Rank Selection. Selecting Tucker ranks remains central to the accuracy–efficiency trade-off. Early heuristics (e.g., VBMF and sensitivity) have been superseded by learning- or search-based strategies. BATUDE [27] integrates *automatic* rank selection into training under explicit budget constraints, yielding globally consistent rank configurations across layers [27]. More recently, unified frameworks cast rank selection as a continuous search with composite compression losses, enabling training-free or data-light rank discovery [28]. Layer-interaction–aware schemes have also appeared, arguing that ranks should be chosen with cross-layer coupling in mind [29].

Zero-Cost Proxies and NAS-Inspired Rank Search. ZC proxies evaluate architectures without training and using just one minibatch or even a single forward/backward pass, e.g., SynFlow, SNIP, GraSP, and their learned/ensembled variants [17, 19, 20]. These proxies have matured with aggregation and symbolic-regression–based design to improve model rank correlation with final accuracy [19, 21]. Despite this progress, their application to *tensor-rank selection* for Tucker remains underexplored: most Tucker works still rely on EBVMF [13], local reconstruction-error heuristics, per-layer sensitivities, or budget-constrained training

loops [27, 28] without considering the global impact on parameter reduction and accuracy trade-off. Our work bridges these lines by framing Tucker rank selection as a NAS problem and leveraging a custom (but simple) ZC proxy to guide ranks across modes and layers, avoiding expensive inner-loop fine-tuning while preserving global budget feasibility.
Efficient models. Lightweight CNNs such as MobileNet [30], SqueezeNet [31], ShuffleNet [32], and EfficientNet [33] have been widely adopted for deployment in resource-constrained environments. These models achieve competitive accuracy with significantly fewer parameters and reduced computational cost compared to conventional architectures. MobileNets employ depthwise separable convolutions to lower FLOPs, SqueezeNet introduces fire modules for parameter efficiency, ShuffleNet leverages channel shuffling with grouped convolutions to enhance information flow, and EfficientNet uses compound scaling to balance network depth, width, and resolution. Together, these architectures form the basis of much of the current research in efficient model design and compression.

BATUDE [27] leverages Tucker decomposition for compressing CNN models while providing budget-aware optimization. It performs layer-wise rank selection using heuristics and is suitable for memory-constrained deployment. However, the rank selection is fixed once computed and does not support flexible reuse for multiple compression levels. Accuracy-Preserving Neural Network Compression via Tucker Decomposition [**kim2016tucker**] focuses on solving a training-decomposition subproblem iteratively but lacks a global search mechanism and configurable reuse across multiple compression levels.Unified Framework for Neural Network Compression via Decomposition and Optimal Rank Selection [34] proposes a fine-grain search strategy to select optimal ranks globally across the network. While effective in exploring the compression–accuracy trade-off, it requires repeated optimization for different compression targets, which increases computational overhead. AutoRank [35] and AdaptiveTT [36] select decomposed ranks automatically but it needs finetuning. Also, the results are demonstrated on very simple and old models.

CompressNAS combines the benefits of prior methods by performing global rank search, supporting budget-aware and fine-grain optimization, and introducing a configurable optimization framework: the search is executed once and reused to generate models at different compression levels. Unlike prior approaches, *CompressNAS* reduces optimization overhead, provides flexibility in compression targets, and ensures practical deployment in resource-constrained scenarios.Given a 4D convolutional kernel $\mathcal{W} \in \mathbb{R}^{O \times I \times H \times W}$, where O and I denote the number of output and input channels respectively, and $H \times W$ represents the spatial kernel size, Tucker decomposition factorizes $\mathcal{W}$ into a smaller core tensor $\mathcal{G}$ and projection matrices along each mode as shown in equation 2.1.

$$\mathcal{W} \approx \mathcal{G} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \times_4 U^{(4)}, \tag{2.1}$$

where $\times_i$ denotes the i-mode product and $U^{(j)}$ are factor matrices. This decomposition reduces redundancy in the kernel representation and allows replacing a single high-dimensional convolution with a sequence of lower-rank convolutions, thereby reducing both parameter count and floating-point operations (FLOPs). When applied to CNN layers, Tucker decomposition is typically performed along the input and output channel dimensions while keeping the spatial kernel sizes intact. This strategy enables significant compression without severely impacting accuracy. Empirical studies have shown that Tucker-based factorization of convolutional layers leads to substantial acceleration while maintaining competitive performance [10].

3. Tucker Decomposition

Figure 1 shows an example of the decomposition of a layer from CNN model. A layer with 32 input channels and 64 output channels is decomposed into three layers. 1. A layer

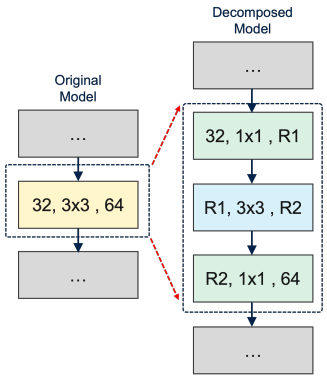

Figure 1. Decomposition of a CNN layer : $R1$ and $R2$ are selected ranks for decomposed layers

with kernel size 1×1 and 32 input channels and $R1$ output channels. 2. Second layer (also called core layer) with input channel as $R1$ and output channels as $R2$ where $R1$ and $R2$ are selected input and output ranks. 3. Final layer with input channels as $R2$ and output channel as 64. The reduction is number of parameters is calculated using Equation 3.1. Using same input and output rank as 8 by replacing $R1$ and $R2$ by 8 in Equation 3.1 can save 17,088 parameters.

$$\text{Number of Parameters} = 32 \cdot 3 \times 3 \cdot 64 - \left(32 \cdot 1 \times 1 \cdot R_1 + R_1 \cdot 3 \times 3 \cdot R_2 + R_2 \cdot 1 \times 1 \cdot 64\right) \quad (3.1)$$

Tucker decomposition leverages EVBMF [10] to automatically determine the ranks for each layer automatically. By estimating optimal (but local) ranks for each layer, number of parameters and FLOPS can be reduced significantly while preserving accuracy. However, EVBMF performs layer-wise, independent rank estimation and does not consider global trade-offs across layers, which can lead to suboptimal accuracy–compression balance compared to methods that perform network-wide rank search. Despite this limitation, EVBMF-based Tucker decomposition remains a widely used baseline for low-rank compression due to its simplicity and automatic rank selection. However, model compression using Tucker decomposition is still an open global rank optimization problem with accuracy/compression tradeoff.

4. CompressNAS Architecture

4.1. Network Proposals

Figure 2 shows the complete architecture of *CompressNAS*. *CompressNAS* adopts a microNAS-inspired approach in which impact of optimization of each layer is considered independently. For convolutional layers, the number of decomposition proposals is determined by the number of output channels. We use an exhaustive search strategy to find the best rank by generating rank proposals beginning from a configurable value (default: 8 channels) and increment in configurable steps of 8 (e.g., 8, 16, ..., up to the maximum number of output channels) or start with 4 channels and increment at the internal of 2,4,8 etc. Lower granularity of filter means larger search space and theoretically fine grain optimized model. For each rank proposal, Tucker decomposition [10] is applied to the target layer, and the original layer is replaced by decomposed layer as shown in figure 1. Each substitution yields a model candidate, i.e., one decomposed layer with one rank proposal corresponds to one model proposal. For every candidate model, we estimate its impact on

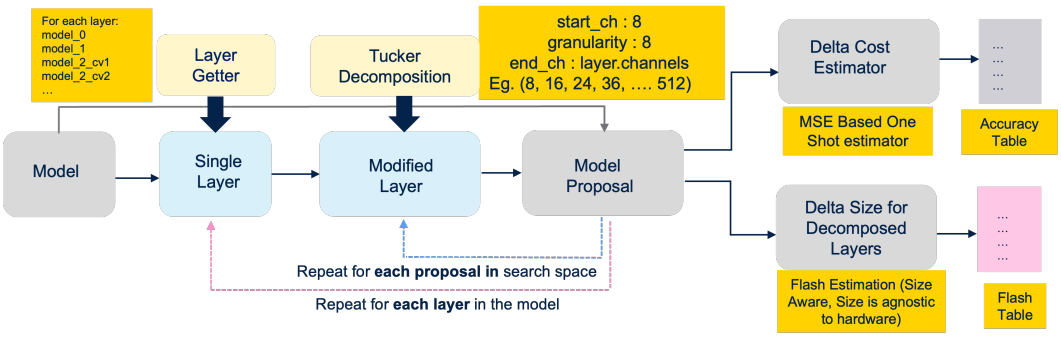

Figure 2. CompressNAS : Model Proposal Generation and Profiling

accuracy (Δacc) and memory footprint or flash size ($\Delta flash$). This process is repeated across all the convolutional and linear layers. For instance, a network with 20 candidate layers and 10 rank proposals per layer creates 200 candidate models. Since the number of channels for each layer is not constant, proposals per layer is variable. The output of this procedure is summarized in two lookup tables: one for Δacc and another for $\Delta flash$

4.2. **Accuracy Estimator**

Accurately estimating the performance of a model after layer decomposition is critical to evaluating the effectiveness of candidate architectures. We investigated a variety of existing zero-cost proxies, including NASWOT [16], GraSP, SNIP, and ZiCo [17], thereby covering both activation-based and gradient-based approaches. Figure 3 illustrates the difference between the proxy scores of the reference model and its decomposed variants across different ranks (x-axis) for the first seven layers of the YOLOv5n model. The leftmost point in each plot corresponds to the reference model and, as expected, yields a zero difference. Subsequent points represent the score differences of candidate models after decomposition at increasing ranks.Ideally, higher ranks should correspond to higher zero-cost proxy scores, reflecting improved accuracy of the decomposed model. However, the graph indicates large inconsistencies, failing to capture the expected monotonic trend making these proxies unreliable for estimate the model accuracy of decomposed models.

As an alternative, we employed a mean squared error (MSE)-based proxy, computed by treating the feature vector output of the decomposed layer as the prediction and the feature vector output of the corresponding reference layer as the target, as illustrated in Figure 4. Unlike existing zero-cost proxies, the MSE-based estimator consistently exhibits the expected behavior, with higher scores observed at higher ranks as shown in figure 3. This suggests that the proposed proxy provides a more reliable measure of the impact on accuracy of decomposition compared to existing methods.

4.3. **Flash Estimator**

After every layer replacement, the model is exported to ONNX [37] and the difference between the reference model and the candidate model provides $\Delta flash$ for a candidate model. We also explored a much simpler hardware agnostic approach to make this process extremely fast by using theoretical calculation of number of parameters for candidate layer and reference layer. For eg. a layer with M output channels, N input channel, k as kernel size and R as decomposed rank, the delta flash can be calculated using equation 4.1. We can use either of these approaches but in case of MCUs, hardware aware method is more reliable.

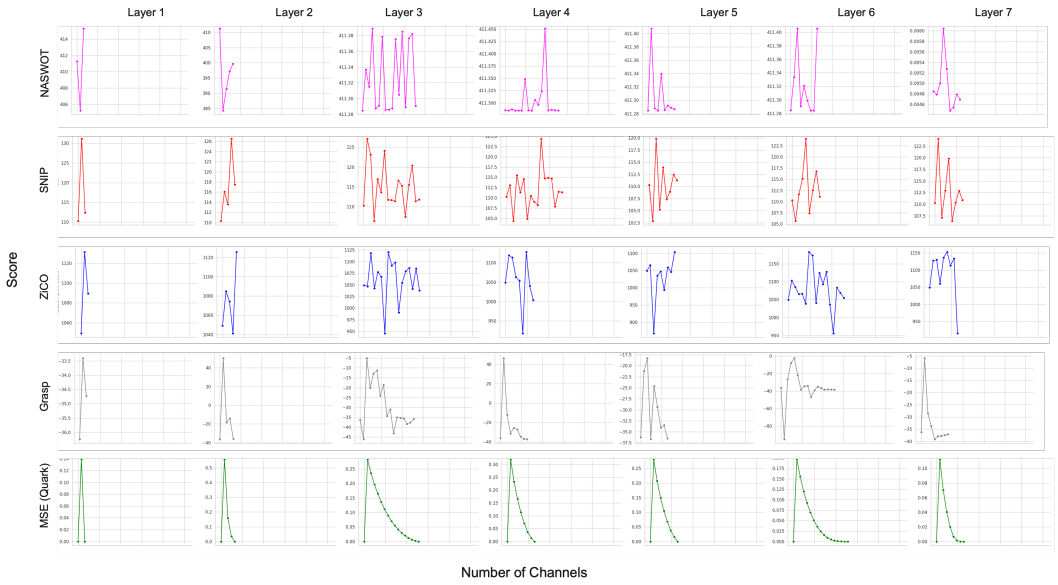

Figure 3. Comparison of different zero cost estimator scores for 7 initial layers of YOLOv5n

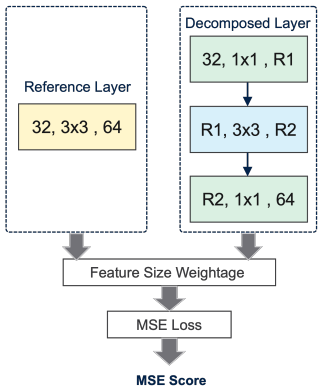

Figure 4. Mean Square Error based Accuracy Estimator

$$\Delta flash = NMk^2 - \left(NR \cdot 1{\times}1 + R^2 \cdot 3{\times}3 + RM \cdot 1{\times}1\right) \tag{4.1}$$

4.4. **Neural Architecture Search**

Using the accuracy estimator, the flash estimator, and MicroNAS strategy, we build two tables: **accuracy table** and **flash table**. In each table, rows represent convolution layers, columns represent rank values, and each cell coressponds to the change (Δ) in accuracy or flash compared to the reference model for that rank. We then formulate the problem of searching for compressed model as an Integer Linear Programming (ILP) [38] optimization problem. The objective is to maximize accuracy and while adhering to flash constraints as equation 4.2. In the ILP formulation, one variable vector represents the indices of the layers where decompositions are applied, and the other represents the candidate (rank for decomposition) for those layers. We used the open-source PuLP library [39] to solve this ILP

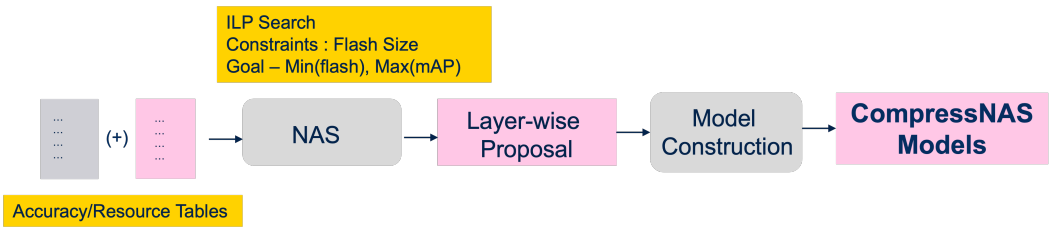

Figure 5. CompressNAS : Optimized Model Creation

problem. The result is a ranking list of the top k model proposals. ILP search provides top model candidates satisfying the flash and accuracy budget. The result is stored as proposed rank for each layer. The search space has number of filers starting from 8 to maximum number of channels for that layer increasing with the granularity of 8 (eg. for a layer with maximum channels as 64, the search space would be 8, 16, 24, 32, 40, 48, 56, 64 and the last one being the reference layer).

$$\begin{aligned} Accuracy = \max \quad & \sum_{(i,j)\in E} \Delta\text{accuracy}_{ij} \\ \text{s.t.} \quad & \sum_{(i,j)\in E} \Delta\text{flash}_{ij} \leq flash_{\max}. \end{aligned} \tag{4.2}$$

4.5. Optimized Model Architecture

This is the final step of model optimization and as depicted in Figure 5, *CompressNAS* uses the reference model and layer wise rank proposal to iteratively replace each layer with decomposed layer resulting in *CompressNAS* Optimized Model. This model needs full retraining to achieve highest possible accuracy. In case a new model is needed with different flash and/or accuracy constraint, the optimization process does not need to be repeated, instead, only ILP search is performed which takes few seconds to generate a new proposal but this model again needs full retraining.

5. Results

We compared performance of classification models on ImageNet [40] and Flowers102 [41] datasets using default timm [42] training pipeline. All the imagenet models are trained from scratch with starting learning rate 0.01 for 300 epochs at 224 input resolution and 256 effective batch size on imagenet dataset. Since the starting point is trained and decomposed model, most of the experiments didn't need 300 epochs of training but we used that as default setting. Usual training time on imagenet dataset is 24 hours on 8 GPUs. Object detection models are trained using ultralytics [43] library on MS COCO [44] datasets. All the models are trained on 640 resolution for 300 epochs with starting learning rate as 0.01 and effective batch size of 128. Usual training time is 16 hours on 8 GPU machine.

5.1. Optimized Models

Tables 1–3 compare *CompressNAS* with vanila Tucker Decomposition [8, 10] and Tucker with Neural Architecture Search (NAS) [45] under varying compression budgets. The NAS method uses search algorithm to decide whether to decompose a particular layer or not by estimating the impact on accuracy by the means of finetuning for few epochs. The NAS also helps to find the best configuration for a given flash constraint.

Table 1. Performance of Classification models trained on Imagenet datasets under different compression budgets. (*with NAS, $^+$without NAS, NAS uses finetuning for each layer and it is very slow)

Model	Params (M)	Top-1 (%)	Compression
ResNet18	11.68	70.50	1.00x
Tucker*	1.66	66.62	7.03×
Tucker$^+$	1.15	58.79	10.15×
CompressNAS	1.50	66.07	7.78×
ResNet34	21.79	76.41	1.00×
Tucker$^+$	1.01	57.35	21.57×
CompressNAS	1.08	59.42	20.18×
MobilenetV2	3.50	71.36	1.00×
Tucker*	1.56	26.45	2.24×
CompressNAS	1.57	38.58	2.24×

Table 2. Performance of YOLOv5 models trained on COCO dataset under different compression budgets. (*without NAS)

Model	Params (M)	mAP	Compression
YOLO5n	1.95	24.20	1.00×
Tucker*	0.95	16.60	2.03×
CompressNAS	0.96	21.70	2.03×
YOLO5s	7.03	32.99	1.00×
Tucker*	2.99	25.90	2.35×
CompressNAS	3.49	32.90	2.00×
CompressNAS	2.11	25.80	3.33×

Table 3. Extreme compression results on ResNet18 with smaller dataset (Flowers102 [41])

Model	Params (M)	Top-1 (%)	Compression
Reference	10.73	91.96	1.00×
CompressNAS	5.36	92.06	2.00×
CompressNAS	3.21	91.54	3.33×
CompressNAS	1.07	89.32	10.00×
CompressNAS	0.85	88.43	12.5×

As shown in Table 1, on **ResNet-18 [46] (ImageNet)**, tucker with NAS achieves a 7.03× compression ratio but suffers a 4.9-point Top-1 accuracy drop, while tucker without NAS collapses to only 58.79% Top-1 accuracy at 10.15× compression. In contrast, *CompressNAS* sustains 66.1% Top-1 accuracy at a comparable 7.78× compression, showing better robustness under aggressive reduction. For **ResNet-34** [46], with NAS compresses the model to 21.6× but degrades Top-1 accuracy to 57.4%. *CompressNAS* achieves a similar 20.2× compression but retains a higher Top-1 accuracy of 59.4%, again demonstrating improved accuracy preservation. For **MobileNetV2** [47], Tucker severely underperforms (26.5% Top-1 at 2.24× compression). *CompressNAS* recovers performance, reaching 38.6% Top-1 accuracy at the same compression budget, however, absolute top-1 accuracy of optimized MobilenetV2 model is very low and this highlights the fact that these compression algorithms fail on hand crafted smaller models like MobileNetV2.

Table 2 shows results for **YOLOv5n** [43] trained on COCO [44] dataset, tucker based compression results in sharp mAP drops (e.g., from 24.2% to 16.6%). *CompressNAS* improves accuracy at identical compression (21.7% mAP at 2.03× for **YOLOv5n**, 32.9% at 2× for **YOLOv5s**) and remains competitive even under more aggressive reduction (3.33×). It is important to note that YOLO5s got compressed by 2× without any accuracy drop.

Finally, Table 3 shows **extreme compression results on Flowers102 [41] with ResNet-18** , *CompressNAS* maintains accuracy under 3.33× compression (91.5% vs. 91.9% baseline) and only gradually degrades under extreme 10× and 12.5× compression, where Top-1 drops to 89.3% and 88.4%, respectively.

5.2. **Optimization Efficiency**

A key advantage of *CompressNAS* is substantial reduction in overall optimization time, enabled by its low-cost accuracy and flash estimators plus the fast ILP search [39]. The accuracy and flash lookup tables can be generated on an AMD EPYC 7542 32-Core processor in under 30 minutes. The subsequent ILP search under any given constraint produces a solution within seconds. Consequently, the end-to-end optimization time for a model consists of one full training cycle plus approximately 30 minutes.

For repeated optimization of the same model with different objectives, the additional computation cost is negligible, as the lookup tables are already available. In contrast, traditional tucker-based rank estimation requires comparable training effort but yields significantly larger accuracy degradation. Relative to Neural Architecture Search (NAS) methods, CompressNAS further improves efficiency: state-of-the-art NAS approaches typically require 3–4 full training cycles to achieve results comparable to CompressNAS, with a search complexity of approximately $\mathcal{O}(n \cdot E)$, where E denotes the number of fine-tuning epochs per candidate. This stark difference arises from the methodology: NAS-based strategies rely on partial fine-tuning to approximate the impact of decomposition, whereas CompressNAS leverages zero-cost estimators in combination with the MicroNAS strategy to directly guide rank selection.

5.3. **State-of-the-Art Comparison**

Table 4. Comparison of state-of-the-art compression methods on ResNet-18. BA- Budget Aware, FGR - Fine Grained Ranks, CO- Configurable Optimization

Technique	**Top-1(%)**	**Top-5(%)**	**Compression**	**BA**	**FGR**	**CO**
Torchvision[46, 48]	69.75	89.08	1.00×			
Vanilla Tucker (2016) [10]	-	87.53	2.25×	✗	✗	✗
MUSCO (2019) [49]	-	88.78	2.42×	✗	✗	✗
Stable EPC (2020) [50]	-	88.93	3.09×	✗	✗	✗
BATUDE (2022)[27]	-	89.41	2.52×	✓	✗	✗
ORTOS Tucker (2024)[34]	70.88	89.87	3.03×	✗	✓	✗
APNN (2025)[24]	-	89.04	3.22×	✗	✗	✗
CompressNAS	**71.00**	**90.55**	**2.52**×	✓	✓	✓
CompressNAS	**70.78**	**90.01**	**3.03**×	✓	✓	✓

Table 4 compares state-of-the-art compression methods on ResNet-18, focusing on low-rank decomposition approaches (Vanilla Tucker [10], MUSCO [49], Stable EPC [50], ORTOS [34]) and our proposed method, CompressNAS. Metrics reported are Top-1 and Top-5 accuracy on ImageNet [40], along with compression ratio (parameter reduction).

CompressNAS achieves the Top-1 (71.84%) and Top-5 (90.54%) accuracies with a competitive 2.52× compression, demonstrating that its global search-based rank selection outperforms prior heuristic, decomposition-based, and recent fine-grain search methods [34]. Other methods, such as BATUDE [27] and Stable EPC [50], achieve similar compression with either comparable or lower accuracy, highlighting *CompressNAS*'s ability to balance efficiency and performance. ORTOS [34] and Accuracy preserving Neural Network Compression (APNN) [24] achieve approximately similar performance given that our experiment is for lower compression (all state of the art methods use different level of compression so

it is not practical to compare with all results with same compression). However, APNN proposed a joint optimization during model training which means the optimziation needs to run again for a different budget. The concept is same for budget aware compression technique [27] where fine grain rank selection is performed at the time of model training. Our optimization method runs once and the model can be generated for different hardware and constraint for different dataset.

In addition to accuracy preservation, *CompressNAS* supports budget-aware optimization [27] and fine-grain rank selection [28] while introducing a configurable optimization framework: the search is executed once and can be reused to generate models at different compression levels, enhancing practical deployment flexibility as shown in 4. We have not compared the performance of our model with [24].

6. Conclusion

In this work, we introduced *CompressNAS*, a microNAS-based decomposition framework leveraging zero-cost estimators for efficient rank selection and model compression. Across classification (ResNet-18, ResNet-34, MobileNetV2) and detection tasks (YOLOv5), *CompressNAS* achieves high compression ratios while retaining accuracy, either outperforming or at par with other tucker based decomposition methods presented in literature. On ResNet-18, it attains among the highest Top-1 and Top-5 accuracies compared to state-of-the-art approaches, demonstrating the effectiveness of global search-based rank selection. *CompressNAS* also supports budget-aware and fine-grain rank optimization, with a configurable framework allowing a single search to generate models at different compression levels [34], significantly reducing overhead. These results establish *CompressNAS* as a practical, scalable, and flexible solution for deploying models on resource-constrained devices, effectively balancing aggressive compression with real-world applicability.

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* 108.11 (2020), pp. 1981–2010.

[2] P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019.

[3] S. Han, H. Mao, and W. J. Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: *arXiv preprint arXiv:1510.00149* (2015).

[4] H. Li, A. Kadav, I. Durdanovic, H. Samet, and W. J. Dally. "Pruning filters for efficient ConvNets". In: *arXiv preprint arXiv:1608.08710* (2017). DOI: 10.48550/arXiv.1608.08710.

[5] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations". In: *Journal of Machine Learning Research* 18.1 (2017), pp. 6869–6898.

[6] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *CVPR*. 2018, pp. 2704–2713.

[7] G. Hinton, O. Vinyals, and J. Dean. "Distilling the Knowledge in a Neural Network". In: *NIPS Deep Learning and Representation Learning Workshop*. 2015.

[8] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. "Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition". In: *ICLR*. 2014.

[9] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation". In: *NeurIPS*. 2014, pp. 1269–1277.

[10] 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". In: *ICLR*. 2016.

[11] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov. "Tensorizing Neural Networks". In: *NeurIPS*. 2015, pp. 442–450.

[12] T. G. Kolda and B. W. Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51.3 (2009), pp. 455–500.

[13] S. Nakajima, M. Sugiyama, S. D. Babacan, and R. Tomioka. "Global Analytic Solution of Fully Observed Variational Bayes for Matrix Factorization". In: *Journal of Machine Learning Research* 14.1 (2013), pp. 1–37.

[14] T. Elsken, J. H. Metzen, and F. Hutter. "Neural Architecture Search: A Survey". In: *Journal of Machine Learning Research* 20.55 (2019), pp. 1–21.

[15] H. Cai, C. Gan, and S. Han. "Once-for-All: Train One Network and Specialize it for Efficient Deployment". In: *ICLR*. 2020.

[16] J. Mellor, J. Turner, A. Storkey, and E. Crowley. "Neural Architecture Search Without Training". In: *ICML*. 2021.

[17] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane. "Zero-Cost Proxies for Lightweight NAS". In: *International Conference on Learning Representations*. 2021. URL: https://arxiv.org/abs/2101.08134.

[18] H. Tanaka, D. Kunin, D. Yamins, and S. Ganguli. "Pruning Neural Networks Without Any Data by Iteratively Conserving Synaptic Flow". In: *NeurIPS*. 2020.

[19] J. Lee and B. Ham. "AZ-NAS: Assembling Zero-Cost Proxies for Network Architecture Search". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024.

[20] O. Kocak et al. "Ensembling Zero-Cost Proxies to Estimate Performance of Neural Architectures". In: *arXiv preprint arXiv:2505.09344* (2025). URL: https://arxiv.org/abs/2505.09344.

[21] S. Chaudhary et al. "Crafting Zero-Cost Proxy Metrics for NAS via Symbolic Regression". In: *NeurIPS Workshop / OpenReview*. 2024. URL: https://openreview.net/forum?id=ludEV7dK9G.

[22] A. Gabor and R. Zdunek. "Deep neural network compression by Tucker decomposition with nonlinear response". In: *Knowledge-Based Systems* (2022). Elsevier.

[23] R. Zdunek and A. Gabor. "Compressing convolutional neural networks with hierarchical Tucker-2 decomposition". In: *Applied Soft Computing* (2023). HT-2 for CNN compression.

[24] Y. Liu et al. "An Accuracy-Preserving Neural Network Compression via Tucker Decomposition". In: *IEEE Transactions on Sustainable Computing* (2025). Alternating optimization with Tucker.

[25] L. Xiang, M. Yin, C. Zhang, A. Sukumaran-Rajam, P. Sadayappan, B. Yuan, and D. Tao. "TDC: Towards Extremely Efficient CNNs on GPUs via Hardware-Aware Tucker Decomposition". In: *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '23)*. ACM, 2023, pp. 260–273. DOI: 10.1145/3572848.3577478.

[26] Z. Li et al. "ELRT: Efficient Low-Rank Training for Compact Convolutional Neural Networks". In: *arXiv preprint arXiv:2401.10341* (2024). URL: https://arxiv.org/abs/2401.10341.

[27] Y. Wang et al. "Budget-Aware Neural Network Compression Based on Tucker Decomposition with Automatic Tensor Rank Selection". In: *AAAI Conference on Artificial Intelligence*. 2022.

[28] Q. Zhang et al. "Unified Framework for Neural Network Compression via Decomposition and Automatic Rank Selection". In: *arXiv preprint arXiv:2409.03555* (2024). URL: https://arxiv.org/abs/2409.03555.

[29] M. Kokhazadeh, G. Keramidas, V. Kelefouras, and I. Stamoulis. "A CNN Compression Methodology for Layer-Wise Rank Selection Considering Inter-Layer Interactions". In: *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE. 2025, pp. 1–7.

[30] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[31] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[32] X. Zhang, X. Zhou, M. Lin, and J. Sun. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 6848–6856.

[33] M. Tan and Q. Le. "EfficientNet: Rethinking model scaling for convolutional neural networks". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019, pp. 6105–6114.

[34] A. Aghababaei-Harandi and M. Amini. "Unified Framework for Neural Network Compression via Decomposition and Optimal Rank Selection". In: *arXiv preprint* abs/2409.03555 (2024). URL: https://arxiv.org/abs/2409.03555.

[35] M. Javaheripi, M. Samragh, and F. Koushanfar. "AutoRank: Automated Rank Selection for Effective Neural Network Customization". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.4 (Dec. 2021), pp. 611–619. DOI: 10.1109/JETCAS.2021.3127433.

[36] S. Luo, M. Liu, P. Sun, Y. Yu, S. Ren, and Y. Bai. "An Adaptive Tensor-Train Decomposition Approach for Efficient Deep Neural Network Compression". In: *arXiv preprint arXiv:2408.01534* (2024).

[37] J. Phillips, N. Gimelshein, et al. "ONNX: Open Neural Network Exchange". In: *Proc. 2nd SysML Conference*. https://onnx.ai. 2019.

[38] C. Trauth Jr and R. Woolsey. "Integer linear programming: a study in computational efficiency". In: *Management Science* 15.9 (1969), pp. 481–493.

[39] S. Mitchell, S. M. Consulting, M. O'Sullivan, and I. Dunning. *PuLP: A Linear Programming Toolkit for Python*. 2022. URL: https://optimization-online.org/?p=11731.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2009), pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[41] M.-E. Nilsback and A. Zisserman. "Automated flower classification over a large number of classes". In: *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing* (2008), pp. 722–729. DOI: 10.1109/ICVGIP.2008.47. URL: https://doi.org/10.1109/ICVGIP.2008.47.

[42] R. Wightman. *PyTorch Image Models (timm)*. https://github.com/rwightman/pytorch-image-models. 2019. DOI: 10.5281/zenodo.4414861.

[43] G. Jocher, A. Chaurasia, and J. Qiu. *YOLO by Ultralytics*. 2023. DOI: 10.5281/zenodo.7347926. URL: https://github.com/ultralytics/ultralytics.

[44] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. "Microsoft COCO: Common Objects in Context". In: *European Conference on Computer Vision (ECCV)*. Springer, 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48.

[45] A. Sankaran, O. Mastropietro, E. Saboori, Y. Idris, D. Sawyer, M. AskariHemmat, and G. B. Hacene. "Deeplite Neutrino: An End-to-End Framework for Constrained Deep Learning Model Optimization". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 17. 2021, pp. 15878–15880.

[46] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[47] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 4510–4520.

[48] TorchVision Contributors. *TorchVision: PyTorch's Computer Vision library*. https://pytorch.org/vision/stable/models.html. Accessed: 2025-08-28. 2016.

[49] J. Gusak, M. Kholiavchenko, E. Ponomarev, L. Markeeva, A. Cichocki, and I. Oseledets. "Automated Multi-Stage Compression of Neural Networks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2019, pp. 2434–2440.

[50] H. Phan, T. Nguyen, T.-T. Do, I. McLoughlin, A. Cichocki, and M. de Vos. "Stable Low-Rank Matrix Decomposition for Compression of Convolutional Neural Networks". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2020.