

# Heterogeneous Factorizations of Convolutional Neural Networks via Tucker Decomposition

by

Frank Mokadem

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Master of Applied Science  
in  
System Design Engineering

Waterloo, Ontario, Canada, 2025

© Frank Mokadem 2025

## **Author's Declaration**

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

Convolutional Neural Network (CNN) remain the architecture of choice for computer vision tasks on compute-constrained platforms such as edge and personal devices, delivering both close to state-of-the-art performance metrics and linear inference complexity with respect to input resolution and number of channels. However, the deployment of larger and more complex CNN architectures is limited by the restrained memory offered by such platforms. This brings about a need to compress pretrained CNN into smaller models in number of parameters while controlling for degradation in performance.

This thesis tackles CNN compression using low rank approximation of convolution layers using Tucker Decomposition (TD). We introduce a new heuristics-based Neural Architectural Search procedure to select low rank configurations for the convolution tensors, which we call Heterogeneous Tucker Decomposition (HTD).

Standard low rank approximation using TD factorizes and approximates convolution layers using uniform ranks for all convolution tensors, then applies a few fine-tuning epochs to recover degradation in performance. An approach we show to be suboptimal against a heterogeneous selection of ranks for each convolution layer, followed by same number of fine-tuning epochs.

Our primary contribution is the development and evaluation of TD, which applies layer-specific compression rate (low rank divided by full rank) inferred from a Neural Architectural Search (NAS) process. Furthermore, we introduce a sampling heuristic to efficiently explore the search space of layer-specific compression rates, thus preserving performance while significantly reducing search time.

We present a mathematical formulation for the HTD optimization problem and an NAS algorithm to find admissible solutions. We test our approach on multiple varieties of CNN architectures: AlexNet, VGG16, and ResNet18, adapted for the MNIST classification task. Our findings confirm that HTD performs better than TD on all models tested. For the same compression rate, HTD enables to recover a higher precision after fine-tuning, with gains ranging from 1.2% to 5.8%. For equivalent accuracy targets, HTD delivers 15-30% higher compression rates than TD.

This thesis advances Neural Architectural Search by highlighting the efficacy of heterogeneous tensor decomposition approaches. It provides a robust framework for their implementation and evaluation, with significant implications for deploying convolutional deep learning models in resource-limited settings. Future work will explore incorporating low-rank constraints as a regularization objective during training, potentially enabling end-to-end compression-aware optimization.

## **Acknowledgements**

I would like to thank my family for their constant support and belief in me. My late father Amor, my mother Naziha and my brothers Farouk and Firas.

I would like to thank my supervisor Alexander Wong for his support and guidance.

I would like to thank the members of the VIP Lab at the University of Waterloo for a great work environment.

## **Dedication**

This is dedicated to the memory of my late father Amor Mokadem (1952 - 2013). A man who devoted his life for his family.

# Table of Contents

|   |            |
|---|------------|
| <b>Author's Declaration</b>                                   | <b>ii</b>  |
| <b>Abstract</b>   | <b>iii</b> |
| <b>Acknowledgements</b>                                       | <b>iv</b>  |
| <b>Dedication</b>   | <b>v</b>   |
| <b>List of Figures</b>  | <b>ix</b>  |
| <b>List of Tables</b>   | <b>x</b>   |
| <b>List of Abbreviations</b>                                  | <b>xi</b>  |
| <b>1 Introduction</b>   | <b>1</b>   |
| 1.1 Motivation . . . . .                                      | 1          |
| 1.2 Role of Tensor Decomposition in CNN Compression . . . . . | 2          |
| 1.3 Problem statement . . . . .                               | 2          |
| 1.4 Approach and contributions . . . . .                      | 3          |
| 1.5 Thesis organization . . . . .                             | 3          |

|          |  |           |
|----------|--|-----------|
| <b>2</b> | <b>Background and Tucker Decomposition</b>                     | <b>5</b>  |
| 2.1      | Foundations of convolutional neural networks . . . . .         | 5         |
| 2.2      | Tensors, Rank and Tucker Decomposition . . . . .               | 6         |
| 2.3      | Tucker decomposition: HOSVD algorithm and truncation . . . . . | 7         |
| 2.4      | Practical mapping to Conv2D layers . . . . .                   | 8         |
| 2.5      | Related work and positioning . . . . .                         | 9         |
| <b>3</b> | <b>Methodology</b>   | <b>11</b> |
| 3.1      | Problem Definition and Objective . . . . .                     | 11        |
| 3.2      | Probabilistic Rank Sampling . . . . .                          | 12        |
| 3.2.1    | Size-Based Heuristic . . . . .                                 | 12        |
| 3.2.2    | Sensitivity-Based Heuristic . . . . .                          | 12        |
| 3.2.3    | Combined Sampling Policy . . . . .                             | 12        |
| 3.3      | Algorithm in Pseudocode . . . . .                              | 13        |
| 3.4      | Search Termination and Fine-Tuning Strategy . . . . .          | 16        |
| 3.4.1    | Complexity and Resource Analysis . . . . .                     | 16        |
| 3.5      | Summary . . . . .  | 18        |
| <b>4</b> | <b>Experiments</b>   | <b>19</b> |
| 4.1      | Introduction . . . . .   | 19        |
| 4.2      | Preliminaries . . . . .  | 19        |
| 4.2.1    | Tensors in CNNs . . . . .                                      | 19        |
| 4.2.2    | Application to Convolutional Layers . . . . .                  | 20        |
| 4.3      | Dataset . . . . .  | 20        |
| 4.3.1    | MNIST Dataset . . . . .  | 20        |
| 4.3.2    | Dataset complexity and diversity . . . . .                     | 21        |
| 4.4      | Models . . . . .   | 21        |
| 4.4.1    | CNN Architectures . . . . .                                    | 21        |

|          |   |           |
|----------|---|-----------|
| 4.5      | Hardware and Software . . . . .   | 25        |
| 4.6      | Optimization Problem . . . . .  | 25        |
| 4.6.1    | Search Space . . . . .  | 26        |
| 4.6.2    | Evaluation Metrics . . . . .  | 26        |
| 4.7      | Training and Fine-tuning . . . . .  | 26        |
| 4.8      | Summary . . . . .   | 29        |
| <b>5</b> | <b>Results</b>  | <b>30</b> |
| 5.1      | Overview . . . . .  | 30        |
| 5.2      | Quantitative summary . . . . .  | 30        |
| 5.3      | Effectiveness of HTD (accuracy versus compression) . . . . .                | 33        |
| 5.4      | Search efficiency and combinatorial complexity . . . . .                    | 33        |
| 5.4.1    | Ablation: Sensitivity of Final Accuracy to Fine-Tuning . . . . .            | 34        |
| 5.5      | Comparative Analysis with State-of-the-Art Rank Selection Methods . . . . . | 36        |
| 5.6      | Additional Observations and Limitations . . . . .                           | 38        |
| 5.7      | Relationship between parameter reduction and inference speed . . . . .      | 38        |
| 5.8      | Chapter summary . . . . .   | 40        |
| <b>6</b> | <b>Conclusion</b>   | <b>41</b> |
| 6.1      | Summary of contributions . . . . .  | 41        |
| 6.2      | Key observations and practical takeaways . . . . .                          | 42        |
| 6.3      | Limitations . . . . .   | 42        |
| 6.4      | Future work . . . . .   | 43        |
| 6.4.1    | Improvements in HTD pipeline . . . . .                                      | 43        |
| 6.4.2    | Improvements in Dataset and Learning Tasks . . . . .                        | 44        |
| 6.5      | Final remarks . . . . .   | 45        |
|          | <b>References</b>   | <b>46</b> |

# List of Figures

|     |  |    |
|-----|--|----|
| 2.1 | Schematic of Tucker-2 compression applied to a Conv2D kernel (channel-mode factorization). . . . . | 9  |
| 3.1 | Heterogeneous Tucker Decomposition (HTD) Search Pipeline. . . . .                                  | 13 |
| 3.2 | Parallel HTD execution . . . . .   | 15 |
| 4.1 | Example images from MNIST showing digits 0–9. Each image is $28 \times 28$ grayscale. . . . .      | 20 |
| 4.2 | AlexNet architecture . . . . .   | 22 |
| 4.3 | VGG16 architecture . . . . .   | 23 |
| 4.4 | ResNet18 architecture . . . . .  | 24 |
| 4.5 | Training accuracy per epoch for AlexNet, VGG16, and ResNet18 on MNIST. . . . .                     | 28 |
| 5.1 | Multi-Metric Comparison of Compressed Models via HTD-NAS . . . . .                                 | 31 |
| 5.2 | Ablation of fine-tuning sensitivity . . . . .  | 35 |
| 5.3 | Comparison of inference time (ms) before and after Tucker compression . . . . .                    | 39 |
| 5.4 | parameter reduction vs. FLOPs reduction . . . . .  | 39 |

# List of Tables

- 5.1 Performance comparison after applying HTD for different optimization criteria 31
- 5.2 Comparison of our NAS-based rank selection with VBMF-based compression 37

# List of Abbreviations

**CNN** Convolutional Neural Network [iii](#), [1](#), [5](#)

**CP** CANDECOMP/PARAFAC [2](#)

**FLOPs** Floating Point Operations [1](#)

**HOSVD** High Order SVD [2](#)

**HTD** Heterogeneous Tucker Decomposition [iii](#), [ix](#), [45](#)

**NAS** Neural Architectural Search [iii](#), [45](#)

**PCA** Principal Component Analysis [2](#)

**SVD** Singular Value Decomposition [2](#)

**TD** Tucker Decomposition [iii](#)

# Chapter 1

## Introduction

CNN are the dominant architecture for a wide range of computer vision tasks, delivering state-of-the-art accuracy on image classification, detection, and segmentation benchmarks while exhibiting inference costs that scale linearly with input spatial size and channel counts. Despite these attractive properties, modern CNNs frequently contain millions to billions of parameters and incur large memory, compute and energy demands, which impede their deployment on resource-constrained platforms such as mobile devices, embedded sensors, and other edge systems [1, 7]. Reducing the storage and runtime footprint of pre-trained CNNs without degrading predictive performance is therefore a central challenge for practical on-device machine learning.

### 1.1 Motivation

A large body of research addresses this challenge by modifying network architectures (designing lightweight models), reducing numerical precision (quantization), removing redundant connections (pruning), or learning compact student networks (distillation) [7, 13, 26, 39]. Complementary to these approaches, low-rank approximation methods exploit concentration of information in weight tensors in its first factors to obtain low rank compressions that simultaneously reduce both parameter count and Floating Point Operations (FLOPs) while preserving the convolutional spatial structure [3, 23]. Furthermore, tensor decomposition based compression can be formalized such that to enable us to utilize neural architectural search procedures, i.e. heuristic search in a tractable space of possible architecture configurations.

## 1.2 Role of Tensor Decomposition in CNN Compression

Early matrix-based factorization methods (Singular Value Decomposition (SVD), Principal Component Analysis (PCA)) require flattening filters into matrices and therefore can destroy spatial or multi-modal correlations present in convolutional kernels. Tensor decomposition methods (CANDECOMP/PARAFAC (CP), High Order SVD (HOSVD), Tucker) operate on the weight tensor in its native multi-dimensional form and thus preserve structural relationships across spatial and channel modes [29]. Among these, Tucker decomposition (and its HOSVD initialization) is particularly well suited for channel-mode compression because it allows independent rank control per tensor mode and produces compact core-and-factor representations that retain spatial correlations [29, 38]. This per-mode flexibility motivates the investigation of *non-uniform* (heterogeneous) rank assignments across layers, since redundancy and sensitivity to approximation vary systematically across depth and layer type.

## 1.3 Problem statement

Given a pretrained CNN  $f(\cdot; W)$  with convolutional kernels represented as fourth-order tensors  $W_\ell \in \mathbb{R}^{c_{\text{in}}^\ell \times c_{\text{out}}^\ell \times k_h^\ell \times k_w^\ell}$  for layers  $\ell = 1, \dots, L$ , where  $c_{\text{in}}$  and  $c_{\text{out}}$  denote the number of input and output channels of a convolutional layer, respectively, while  $k_h$  and  $k_w$  denote the spatial kernel height and width. we seek a per-layer rank vector  $\mathbf{r} = [(r_{\text{in}}^\ell, r_{\text{out}}^\ell)]_{\ell=1}^L$  and associated Tucker replacements that minimize resource cost (parameters, FLOPs, latency) subject to an accuracy constraint on a validation set:

$$\min_{\mathbf{r}} C(\mathbf{r}) \quad \text{s.t.} \quad \mathcal{A}(f_{\mathbf{r}}) \geq \gamma \cdot \mathcal{A}(f_{\text{base}}),$$

where  $f_{\mathbf{r}}$  denotes the compressed model after applying Tucker factors with ranks  $\mathbf{r}$  and optional fine-tuning,  $C(\cdot)$  is a cost measure (e.g., total parameters or estimated FLOPs),  $\mathcal{A}(\cdot)$  is a chosen accuracy metric, and  $\gamma \in (0, 1]$  is an acceptance multiplier. The discrete, combinatorial nature of feasible rank assignments makes exhaustive optimization infeasible for realistic networks, motivating guided search strategies.

## 1.4 Approach and contributions

This thesis proposes a practical pipeline that combines Tucker decomposition with a heuristic-guided Neural Architecture Search (NAS) over layer-wise ranks. The approach we call *Heterogeneous Tucker Decomposition* (HTD), is designed to sample non-uniform rank configurations using a heuristic computed based on convolution tensor reconstruction error and size and use short fine-tuning runs to recover accuracy for promising configurations. The main contributions are:

1. **HTD framework:** a formalization of per-layer Heterogeneous Tucker decompositions as a constrained rank-selection problem and an end-to-end pipeline to apply low-rank compressions on pretrained CNNs.
2. **Heuristic NAS for ranks:** a sampling scheme that combines layer size and reconstruction-sensitivity heuristics to bias sampling toward configurations likely to yield favorable accuracy-compression trade-offs, thereby reducing the complexity of the search space.
3. **Implementation and evaluation:** a reproducible experimental implementation (TensorLy + PyTorch) and empirical validation on multiple CNN backbones (AlexNet, VGG16, ResNet18) adapted to the MNIST classification task; experimental measures include parameters, FLOPs, inference latency and post-finetune accuracy.
4. **Analysis and insights:** empirical characterization of where heterogeneous rank schedules produce the largest gains, and a discussion of limitations and potential extensions (compression-aware training).

## 1.5 Thesis organization

The remainder of the thesis follows the structure motivated in this chapter:

- Chapter 2 (Background) reviews tensor decompositions, formalizes convolutional kernel tensors and Tucker decomposition based compression procedure.
- Chapter 3 (Methodology) presents the HTD pipeline in detail: rank candidate generation, sensitivity estimation, the heuristic-guided NAS algorithm, and implementation considerations (model duplication, device placement, and parallel evaluation).
- Chapter 4 (Experiments) describes datasets, experimental protocol, and

- Chapter 5 (Results) Reports quantitative results: compression vs. accuracy trade-offs, comparing complexity of uniform TD vs heuristic HTD. Also, points out a surprising lack of correlation between FLOPS and Compression rates
- Chapter 6 (Conclusion) summarizes findings, discusses limitations, and outlines directions for future work, including integrating rank constraints as regularizers in training.

This chapter has motivated the study and formalized the problem that the remainder of this thesis addresses. Chapter 2 now develops the mathematical and algorithmic foundations required for HTD.

# Chapter 2

## Background and Tucker Decomposition

### 2.1 Foundations of convolutional neural networks

CNN are built from repeated applications of convolutional layers, non-linearities and pooling/normalization operations. A single 2D convolutional layer with kernel size  $k_h \times k_w$ , input channels  $C_{\text{in}}$  and output channels  $C_{\text{out}}$  is parameterized by a fourth-order weight tensor

$$\mathcal{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_h \times k_w}.$$

The number of scalar parameters for this convolution is

$$P_{\text{orig}} = C_{\text{out}} \cdot C_{\text{in}} \cdot k_h \cdot k_w,$$

and the number of multiply-accumulate operations (MACs) for a single output spatial location scales as  $O(C_{\text{out}}C_{\text{in}}k_hk_w)$ . For an image of spatial resolution  $H \times W$  the total FLOPs scale approximately as  $2 \cdot P_{\text{orig}} \cdot H \cdot W$  (factor 2 counting multiply and add). These scaling laws explain why channels and kernel sizes dominate model storage and compute at inference time, motivating techniques that exploit redundancy in the multi-way weight tensor  $\mathcal{W}$ .

## 2.2 Tensors, Rank and Tucker Decomposition

**Tensor and modes.** An  $N$ -way (order- $N$ ) tensor is a multi-dimensional array  $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ . Each index  $n \in \{1, \dots, N\}$  is called a *mode*. A common operation is the mode- $n$  matricization (unfolding)  $X_{(n)} \in \mathbb{R}^{I_n \times (I_1 \dots I_{n-1} I_{n+1} \dots I_N)}$ , which reshapes the tensor into a matrix where rows correspond to mode  $n$ .

**Rank (multilinear).** For tensors, several distinct notions of rank exist. In this chapter we use the *Tucker* (multilinear) rank which is a tuple  $\mathbf{R} = (R_1, \dots, R_N)$  where  $R_n = \text{rank}(X_{(n)})$  is the matrix rank of the mode- $n$  unfolding. The Tucker rank captures mode-wise linear subspace dimensionalities and is well suited for decompositions that compress each mode separately.

**Tucker decomposition.** The Tucker decomposition (also frequently realized via the Higher-Order SVD (HOSVD)) factorizes  $\mathcal{X}$  into a small *core* tensor  $\mathcal{C}$  and a set of factor matrices  $U^{(n)}$  that span dominant subspaces for each mode:

$$\mathcal{X} \approx \widehat{\mathcal{X}} = \mathcal{C} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 \dots \times_N U^{(N)}, \quad (2.1)$$

where  $U^{(n)} \in \mathbb{R}^{I_n \times R_n}$  and  $\mathcal{C} \in \mathbb{R}^{R_1 \times R_2 \times \dots \times R_N}$ . Here  $\times_n$  denotes the mode- $n$  tensor-matrix product (multiplying  $U^{(n)}$  into the  $n$ -th mode).

**Why Tucker for NAS-guided compression.** While other decompositions such as CP (CANDECOMP/PARAFAC), Tensor-Train (TT), and Hierarchical Tucker (HT) exist and have been applied to neural network compression [29, 17], Tucker is chosen here for the following reasons:

- **Mode-wise factorization:** Tucker explicitly separates each tensor mode into its own factor matrix  $U^{(n)}$ . For convolutional kernels this yields independent control over input-channel and output-channel subspaces, enabling a search space where each layer can be assigned distinct ranks for each channel mode.
- **Flexible search space:** Because ranks  $(R_1, \dots, R_N)$  are chosen per-mode and per-layer, Tucker naturally supports heterogeneous compression schedules, a crucial property for Neural Architecture Search (NAS) approaches that must explore per-layer rank configurations.

- **Expressivity vs. parsimony trade-off:** The Tucker format provides a compact low-dimensional core that models interactions between modes, often preserving spatial correlations better than simple matricizations. This balances expressive capacity and parameter reduction in convolutional filters.

This chapter therefore develops the Tucker formalism and its SVD-based computation in detail; rank-selection and NAS are treated later in Chapter 3.

## 2.3 Tucker decomposition: HOSVD algorithm and truncation

We present the truncated Higher-Order SVD (HOSVD), a deterministic procedure that computes orthogonal factor matrices  $U^{(n)}$  via SVDs of mode unfoldings and a core tensor  $\mathcal{C}$ . We then quantify the reconstruction error produced by truncation to finite ranks.

**Mode- $n$  unfolding and SVD.** Given  $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ , compute the mode- $n$  unfolding  $X_{(n)}$ . Its SVD is

$$X_{(n)} = U^{(n)} \Sigma^{(n)} V^{(n)\top},$$

where  $U^{(n)} \in \mathbb{R}^{I_n \times I_n}$  (or truncated up to  $I_n$ ),  $\Sigma^{(n)}$  contains singular values  $\sigma_1^{(n)} \geq \sigma_2^{(n)} \geq \dots \geq 0$ .

**Truncated HOSVD (procedure).** Choose target multilinear ranks  $R_n \leq I_n$  for each mode. The truncated HOSVD computes truncated factor matrices  $\tilde{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$  by taking the first  $R_n$  left singular vectors of  $X_{(n)}$ . The core tensor  $\mathcal{C}$  is then obtained by projecting  $\mathcal{X}$  onto the mode subspaces:

$$\mathcal{C} = \mathcal{X} \times_1 \tilde{U}^{(1)\top} \times_2 \tilde{U}^{(2)\top} \times_3 \dots \times_N \tilde{U}^{(N)\top}. \quad (2.2)$$

The rank-truncated approximation is reconstructed as in Equation (2.1).

## Truncated HOSVD Algorithm

This algorithm computes the Truncated HOSVD of an  $N$ -dimensional tensor  $\mathcal{X}$ .

---

**Algorithm 1** Truncated HOSVD for a tensor  $\mathcal{X}$ 

---

- 1: **Input:** Tensor  $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ , target ranks  $\{R_n\}_{n=1}^N$
  - 2: **for**  $n = 1$  **to**  $N$  **do**
  - 3:   Form mode- $n$  unfolding  $X_{(n)}$
  - 4:   Compute SVD:  $X_{(n)} = U^{(n)}\Sigma^{(n)}V^{(n)\top}$
  - 5:   Truncate  $U^{(n)}$  to first  $R_n$  columns:  $\tilde{U}^{(n)} \leftarrow U^{(n)}[:, 1 : R_n]$
  - 6: **end for**
  - 7: Compute core tensor  $\mathcal{C}$  via mode- $n$  products:
  - 8:  $\mathcal{C} \leftarrow \mathcal{X} \times_1 \tilde{U}^{(1)\top} \times_2 \dots \times_N \tilde{U}^{(N)\top}$
  - 9: **Output:** Factor matrices  $\{\tilde{U}^{(n)}\}_{n=1}^N$  and core tensor  $\mathcal{C}$
- 

**Truncation and reconstruction error.** Let  $\hat{\mathcal{X}}$  be the approximation produced by truncated HOSVD with ranks  $\{R_n\}$ . The squared Frobenius-norm reconstruction error can be upper-bounded by the tail singular values of each unfolding. Using orthogonality properties of HOSVD one obtains:

$$\|\mathcal{X} - \hat{\mathcal{X}}\|_F^2 \leq \sum_{n=1}^N \sum_{i>R_n} (\sigma_i^{(n)})^2, \quad (2.3)$$

where  $\{\sigma_i^{(n)}\}_i$  are singular values of the  $n$ -mode unfolding  $X_{(n)}$ . Equation (2.3) emphasizes that truncation error is controlled by the decay of singular values along each mode; modes with rapidly decaying singular spectra admit aggressive truncation with small reconstruction cost. Note that this bound is conservative; tighter bounds exist for specific tensors or when additional structure is exploited [5, 17].

## 2.4 Practical mapping to Conv2D layers

For convolutional kernels  $\mathcal{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_h \times k_w}$  the Tucker decomposition is typically applied to compress channel modes (modes 1 and 2 in some conventions). A common implementation pattern for Tucker-based compression of Conv2D is Tucker-2 (compress only channel modes) which leads naturally to a three-layer replacement:

$$\text{Conv2D}(C_{\text{in}} \xrightarrow[k \times k]{} C_{\text{out}}) \quad \longrightarrow \quad \underbrace{1 \times 1}_{C_{\text{in}} \rightarrow r_{\text{in}}} \rightarrow \underbrace{k \times k}_{r_{\text{in}} \rightarrow r_{\text{out}}} \rightarrow \underbrace{1 \times 1}_{r_{\text{out}} \rightarrow C_{\text{out}}} .$$

Concretely, if we choose channel-mode Tucker ranks  $r_{\text{in}}$  and  $r_{\text{out}}$ , the parameters of the replacement block are approximately

$$P_{\text{tucker}} \approx C_{\text{in}}r_{\text{in}} + r_{\text{in}}r_{\text{out}}k_hk_w + r_{\text{out}}C_{\text{out}}, \quad (2.4)$$

which should be compared to  $P_{\text{orig}} = C_{\text{in}}C_{\text{out}}k_hk_w$ . When  $r_{\text{in}}, r_{\text{out}} \ll C_{\text{in}}, C_{\text{out}}$  the reduction in parameters and FLOPs can be substantial. In practice, the decomposition is implemented by computing truncated HOSVD factors for the weight tensor, constructing three small convolutional modules from these factors (two  $1 \times 1$  and one  $k \times k$ ), and replacing the original Conv2D with this composite module and continuing to fine-tune the network.

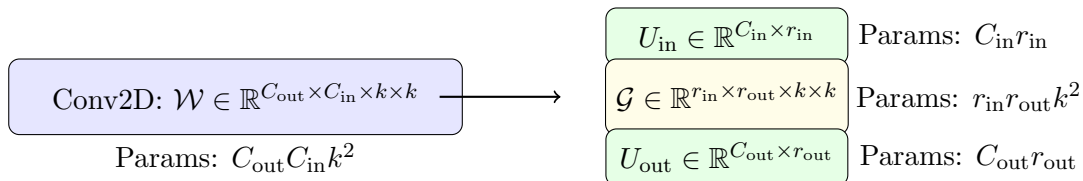


Figure 2.1: Schematic of Tucker-2 compression applied to a Conv2D kernel (channel-mode factorization).

## 2.5 Related work and positioning

Low-rank methods for neural networks evolved from classical matrix factorization techniques such as SVD and PCA applied to weight matrices; however, matrix approaches require reshaping filter tensors and can destroy spatial structure [29, 23]. Tensor decompositions preserve multi-way structure and exploit redundancies across modes directly. CP and Tucker have each been applied in various contexts: CP allows to express a tensor as a sum of rank 1 tensors, easier to compute but offer less flexibility to control where in the tensor dimensions to spend your compression budget. Whereas Tucker (similar to truncated HOSVD) is often chosen due to its flexibility in allowing per-mode (dimension of a tensor) rank control [17, 29, 38].

Tucker-based compression combined with short fine-tuning has been shown in prior work to offer a strong trade-off between compression and accuracy recovery [23]. The principal novelty of this thesis is to treat the per-layer Tucker ranks as NAS variables and to design a heuristic-guided search over heterogeneous rank configurations; this leverages the mode-wise factorization property of Tucker to build a flexible search space in which

each layer may receive a distinct pair of channel-mode ranks. The search algorithm and empirical validation are presented in Chapter 3 and Chapter 5 respectively.

# Chapter 3

## Methodology

This chapter formalizes the search-based compression pipeline used in this work, termed the **Heterogeneous Tucker Decomposition (HTD) Neural Architecture Search**. We first establish the mathematical formulation of the compression objective, then specify the heuristic-driven search algorithm, per-configuration processing, and termination conditions. Finally, we discuss computational complexity and design considerations that ensure the method’s scalability.

### 3.1 Problem Definition and Objective

Let  $M$  be a pretrained CNN with convolutional layers  $\mathcal{L} = \{\ell_1, \dots, \ell_L\}$  and weight tensors  $\mathcal{W}_i$ . Each layer  $\ell_i$  can be replaced by its Tucker-2 approximation with ranks  $(r_{\text{in},i}, r_{\text{out},i})$ . A rank configuration is

$$\mathbf{r} = \{(r_{\text{in},i}, r_{\text{out},i})\}_{i=1}^L.$$

We define  $P(M)$ ,  $a(M)$ , and  $T(M)$  as the parameter count, validation accuracy, and inference time of model  $M$ , respectively. The search is formulated as a constrained optimization problem:

$$\min_{\mathbf{r}} P(M_{\mathbf{r}}) \quad \text{s.t.} \quad a(M_{\mathbf{r}}) \geq \tau a(M), \quad (3.1)$$

where  $M_{\mathbf{r}}$  is the model obtained after applying per-layer Tucker replacements, and  $\tau \in (0, 1]$  is the acceptable accuracy multiplier (we will use in this work  $\tau = 0.9$ ).

## 3.2 Probabilistic Rank Sampling

The search space over ranks is exponential in  $L$ , making exhaustive exploration infeasible. We introduce a probabilistic policy  $\pi_\ell(r)$  per layer, combining two heuristics that encode prior knowledge about layer structure.

### 3.2.1 Size-Based Heuristic

Layers with large channel products  $S_\ell = C_{\text{in},\ell}C_{\text{out},\ell}$  are more redundant and thus suitable for stronger compression. For each candidate rank  $r \in R_\ell$  we define:

$$p_\ell^{\text{size}}(r) = \frac{\exp(-\beta r / \log(S_\ell + 1))}{\sum_{r' \in R_\ell} \exp(-\beta r' / \log(S_\ell + 1))}, \quad (3.2)$$

where  $\beta$  is a temperature scaling constant controlling bias strength.

### 3.2.2 Sensitivity-Based Heuristic

We estimate each layer’s reconstruction sensitivity via partial Tucker decomposition:

$$s_\ell = \text{MSE}(\mathcal{W}_\ell, \widehat{\mathcal{W}}_\ell^{(50\%)}) \quad (3.3)$$

where  $\widehat{\mathcal{W}}_\ell^{(50\%)}$  is reconstructed from a decomposition using 50% channel ranks. The choice of a 50% channel rank in 3.3 serves as a standard uniform cut off across all layers to evaluate then compare relative reconstruction error. The resulting sensitivity scores are normalized to  $[0, 1]$  and converted into probabilities:

$$p_\ell^{\text{sens}}(r) = \frac{(r/r_{\text{max},\ell})^{\alpha s_\ell}}{\sum_{r' \in R_\ell} (r'/r_{\text{max},\ell})^{\alpha s_\ell}}, \quad (3.4)$$

where  $\alpha$  modulates the effect of sensitivity on penalizing aggressive compression.

### 3.2.3 Combined Sampling Policy

The final search policy distribution is defined multiplicatively:

$$p_\ell(r) = \frac{p_\ell^{\text{size}}(r)^{1-\lambda} p_\ell^{\text{sens}}(r)^\lambda}{\sum_{r' \in R_\ell} p_\ell^{\text{size}}(r')^{1-\lambda} p_\ell^{\text{sens}}(r')^\lambda}, \quad (3.5)$$

where  $\lambda \in [0, 1]$  balances the influence between size- and sensitivity-based priors. Independent sampling from  $\{p_\ell(r)\}$  across layers yields heterogeneous rank configurations  $\mathbf{r}$ .

### 3.3 Algorithm in Pseudocode

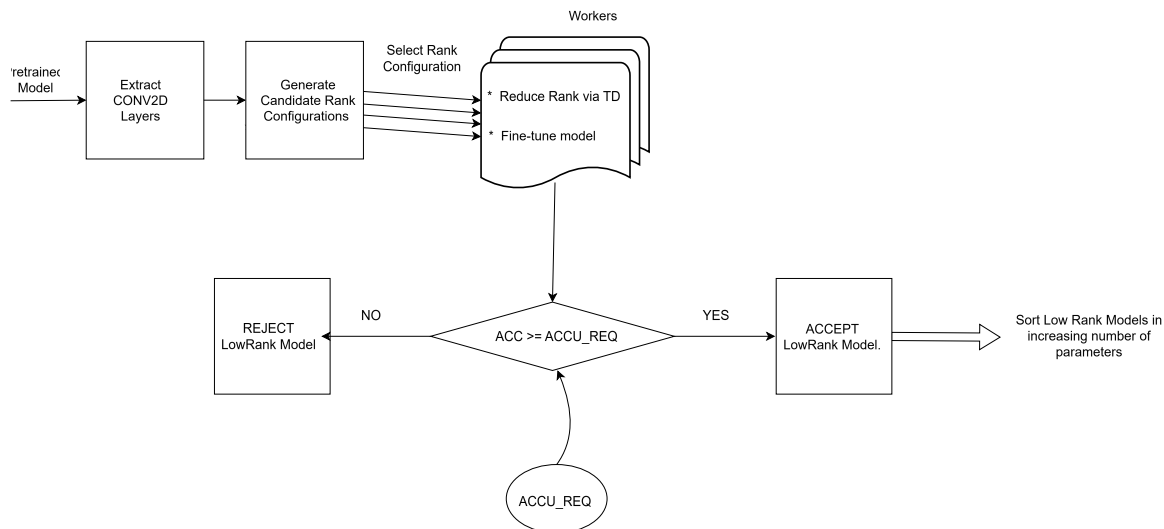


Figure 3.1: **Heterogeneous Tucker Decomposition (HTD) Search Pipeline.** The figure illustrates the overall search process for selecting layer-specific Tucker ranks. Beginning with a pretrained CNN, convolutional layers are extracted and candidate low-rank configurations are generated. Each configuration is assigned to a worker process that performs Tucker Decomposition (TD) on the convolutional tensors and fine-tunes the model to recover accuracy. Models meeting the minimum accuracy requirement ( $ACCU\_REQ$ ) are accepted, while others are rejected. Accepted low-rank models are then sorted by increasing parameter count to identify the most compact architecture satisfying the accuracy constraint.

Figure 3.1 presents the complete pipeline of the proposed Heterogeneous Tucker Decomposition (HTD) framework. The process begins by isolating all convolutional layers of a pretrained model, which serve as the input for candidate rank generation. Each rank configuration defines a unique compression hypothesis, controlling the dimensionality reduction applied to the convolutional kernels.

Worker nodes operate in parallel, each performing Tucker Decomposition (TD) followed by short fine-tuning cycles to recover potential accuracy losses. Once each candidate model is evaluated, its validation accuracy is compared to a predefined acceptance threshold  $ACCU\_REQ$ .

Models achieving accuracy above this threshold are retained as admissible solutions,

whereas those that fail are discarded. Finally, all admissible models are sorted in ascending order of parameter count, ensuring that the model offering the best compression–accuracy trade-off can be systematically identified.

This process enables an efficient exploration of the large combinatorial space of Tucker ranks across layers without exhaustively evaluating all configurations, forming the core of the HTD Neural Architecture Search strategy.

Input: baseline model  $M$ , training loader  $D_{\text{train}}$ , validation loader  $D_{\text{val}}$ , budget  $N_{\text{max}}$ , acceptance multiplier  $\tau$ , fine-tune epochs  $E$ .

1. Extract  $\text{layer\_dict} = \{\text{ell}: (W_{\text{ell}}, C_{\text{in\_ell}}, C_{\text{out\_ell}})\}$ .
2. For each layer  $\text{ell}$  compute  $r_{\text{max\_ell}} = \min(C_{\text{in\_ell}}, C_{\text{out\_ell}})$  and  $\text{params\_ell}$ .
3. Compute baseline metrics:  $P(M)$ ,  $\text{FLOPs}(M)$ ,  $a(M)$ ,  $T(M)$ .
4. Build candidate generator  $G$ :
  - Option A (cartesian): produce up to  $N_{\text{max}}$  entries by selecting top- $s$  truncated Cartesian product of per-layer percentile candidates.
  - Option B (heuristic sampling): precompute sensitivities  $s_{\text{ell}}$  and build per-layer distributions  $p_{\text{ell}}(r)$  combining size and sensitivity.
  - Then sample  $N_{\text{max}}$  configs ranks according to distribution of  $p_{\text{ell}}$ .
5. For each config  $r$  in  $G$  (in parallel, up to  $\text{num\_workers}$ ):
  - a.  $M' \leftarrow \text{duplicate\_model}(M)$
  - b. For each layer  $\text{ell}$ :
    - $M' \leftarrow \text{replace\_conv2d\_with\_tucker}(M', \text{ell}, \text{rank}=r_{\text{ell}})$
  - c. Move  $M'$  to device.
  - d. If  $E > 0$ :  $M' \leftarrow \text{fine\_tune}(M', D_{\text{train}}, \text{epochs}=E)$
  - e. Evaluate metrics:  $a(M')$ ,  $P(M')$ ,  $\text{FLOPs}(M')$ ,  $T(M')$ .
  - f.  $\text{accepted} = (a(M') \geq \tau * a(M))$
  - g. Append result to results list; if accepted append to  $\text{accepted\_models}$ .
  - h. Free  $M'$  and clear CUDA cache.
6. After all configs:
  - Sort  $\text{accepted\_models}$  by chosen utility (e.g.,  $\text{compression\_rate}$ ).
  - Persist top- $k$  models and remove temporary artifacts.

Output: results list,  $\text{accepted\_models}$  (top candidates), saved model files.

Figure 3.2: Algorithm for parallel execution of Low-Rank selection via Neural Architecture Search of an optimal Heterogeneous Tucker Decomposition

## 3.4 Search Termination and Fine-Tuning Strategy

The search terminates when the predefined budget  $N_{\max}$  of configurations has been evaluated.

Each configuration undergoes a lightweight fine-tuning phase of  $E = 3$  epochs on the validation data set. This limited retraining empirically suffices to restore accuracy loss induced by decomposition within 1% of baseline model accuracy, a tolerance that is provided as input to the constraint optimization problem of finding low rank configurations. This shows that the majority of accuracy recovery occurs within the first one to three epochs across all model architectures. Although one might consider scaling  $E$  as a function of the available validation data, our experiments indicate that performance stabilizes rapidly and additional epochs produce diminishing returns. Since the goal of HTD is to evaluate many candidate decompositions efficiently, fixing  $E = 3$  offers a consistent tradeoff between evaluation cost and recovery quality.

### 3.4.1 Complexity and Resource Analysis

Let  $L$  denote the number of convolutional layers and  $N_{\max}$  the total number of rank configurations evaluated during the search. Each configuration involves three main computational components: Tucker decomposition, short fine-tuning, and validation evaluation.

**Per-configuration complexity.** For a convolutional kernel  $W_\ell \in \mathbb{R}^{C_{\text{out}}^{(\ell)} \times C_{\text{in}}^{(\ell)} \times k_h \times k_w}$ , the dominant cost in the Tucker-2 factorization arises from two truncated SVDs on the mode-1 and mode-2 unfoldings. The complexity of SVD on an  $m \times n$  matrix is  $\mathcal{O}(\min(m^2n, mn^2))$ . For the unfoldings (e.g.,  $m = C_{\text{out}}^{(\ell)}$ ,  $n = C_{\text{in}}^{(\ell)} k_h k_w$ ), assuming the kernel size  $k_h k_w$  is a small constant  $\mathcal{O}(1)$ , the complexity is:

$$T_{\text{decomp}}^{(\ell)} = \mathcal{O}\left(\min\left((C_{\text{out}}^{(\ell)})^2 C_{\text{in}}^{(\ell)}, C_{\text{out}}^{(\ell)} (C_{\text{in}}^{(\ell)})^2\right)\right) \approx \mathcal{O}(C_\ell^3), \quad (3.6)$$

where  $C_\ell = \max(C_{\text{in}}^{(\ell)}, C_{\text{out}}^{(\ell)})$ . Across all layers, the total decomposition cost per configuration is:

$$T_{\text{decomp}} = \sum_{\ell=1}^L \mathcal{O}(C_\ell^3). \quad (3.7)$$

Fine-tuning over  $E$  epochs on a dataset  $\mathcal{D}$  of size  $|\mathcal{D}|$  incurs a cost proportional to the number of forward and backward passes:

$$T_{\text{fit}} = \mathcal{O}\left(E|\mathcal{D}|\sum_{\ell=1}^L C_{\ell}^2\right), \quad (3.8)$$

and a single validation pass over a dataset  $\mathcal{D}_{\text{val}}$  requires:

$$T_{\text{eval}} = \mathcal{O}\left(|\mathcal{D}_{\text{val}}|\sum_{\ell=1}^L C_{\ell}^2\right). \quad (3.9)$$

Combining these contributions, the per-configuration runtime can be written as:

$$T_{\text{cfg}} = \mathcal{O}\left(\sum_{\ell=1}^L C_{\ell}^3 + (E|\mathcal{D}| + |\mathcal{D}_{\text{val}}|)\sum_{\ell=1}^L C_{\ell}^2\right). \quad (3.10)$$

**Total search cost.** Since the  $N_{\text{max}}$  configurations are evaluated independently, the total cost of the search process is linear in  $N_{\text{max}}$ :

$$T_{\text{search}}(N_{\text{max}}) = N_{\text{max}} T_{\text{cfg}} = \mathcal{O}\left(N_{\text{max}}\left[\sum_{\ell=1}^L C_{\ell}^3 + (E|\mathcal{D}| + |\mathcal{D}_{\text{val}}|)\sum_{\ell=1}^L C_{\ell}^2\right]\right). \quad (3.11)$$

Equation (3.11) formally justifies the empirical observation that

$$T_{\text{search}} = \Theta(N_{\text{max}}),$$

that is, the total search time and memory consumption grow linearly with the number of sampled configurations  $N_{\text{max}}$ , assuming constant per-configuration compute resources. With  $W$  parallel worker processes, the effective wall-clock time reduces to  $T_{\text{search}}/W$ .

**Memory complexity.** Each worker holds one model copy with memory footprint

$$M_{\text{cfg}} = \mathcal{O}\left(\sum_{\ell=1}^L C_{\ell}^2\right), \quad (3.12)$$

which is upper-bounded by the uncompressed model size since Tucker factors satisfy  $r_{\text{out}}, r_{\text{in}} \leq C_{\ell}$ . Therefore, in a serial execution the total memory cost also scales linearly with  $N_{\text{max}}$ , and as  $\mathcal{O}(WM_{\text{cfg}})$  in parallel settings.

## 3.5 Summary

This methodology formalizes the HTD-based neural architecture search as a constrained probabilistic optimization problem. The proposed pipeline integrates tensor decomposition theory, layer-specific sensitivity modeling, and heuristic-guided exploration. By limiting fine-tuning epochs and enforcing search budgets, the framework achieves practical scalability while discovering rank configurations that maintain high accuracy at reduced parameter and FLOP budgets.

Furthermore, The use of a probabilistic sampling policy effectively controls the proportionality constant in Eq. (3.11). By restricting the number of evaluated configurations to a small but diverse subset of the rank space, the search complexity reduces from exponential  $\mathcal{O}(k^L)$  (for a grid search with  $k$  candidates per layer) to linear  $\mathcal{O}(N_{\max})$ , while maintaining comparable compression quality.

# Chapter 4

## Experiments

### 4.1 Introduction

This chapter describes the experimental setup used to evaluate the compression of convolutional neural networks (CNNs) using Tucker decomposition and its heterogeneous variants. The goal is to reduce parameter count and computational complexity while preserving predictive accuracy on the MNIST dataset [21]. The CNN architectures considered are AlexNet [20], VGG16 [30], and ResNet18 [12], which are originally designed for more complex datasets such as ImageNet. Here, they are adapted for digit recognition to study compression techniques in a controlled environment.

The chapter is organized as follows: Section 4.2 covers preliminary concepts, Sections 4.4 and 4.3 describes datasets and models, Section 4.6 formulates the compression optimization problem, Section 4.5 details software and hardware tools, and Section 4.7 discusses training and fine-tuning procedures.

### 4.2 Preliminaries

#### 4.2.1 Tensors in CNNs

Recall that a CNNs, convolutional layer weights are represented as 4D tensors

$$\mathcal{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_h \times k_w},$$

where  $C_{\text{out}}$  and  $C_{\text{in}}$  are the numbers of output and input channels, respectively, and  $k_h \times k_w$  is the spatial kernel size. These tensors are the primary target for compression.

Recall that Tucker decomposition factorizes a tensor  $\mathcal{W} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$  into a core tensor and factor matrices:

$$\mathcal{W} \approx \mathcal{G} \times_1 U_1 \times_2 U_2 \cdots \times_N U_N, \quad (4.1)$$

where  $\mathcal{G} \in \mathbb{R}^{R_1 \times \dots \times R_N}$  is the core tensor,  $U_n \in \mathbb{R}^{I_n \times R_n}$  is the factor matrix for mode  $n$ , and  $R_n \leq I_n$  is the Tucker rank of that mode [29].

## 4.2.2 Application to Convolutional Layers

For convolutional weight tensors  $\mathcal{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_h \times k_w}$ , we apply Tucker decomposition on channel dimensions while keeping the kernel dimensions at full rank:

$$\mathcal{G} \in \mathbb{R}^{r_{\text{out}} \times r_{\text{in}} \times k_h \times k_w}, \quad U_{\text{out}} \in \mathbb{R}^{C_{\text{out}} \times r_{\text{out}}}, \quad U_{\text{in}} \in \mathbb{R}^{C_{\text{in}} \times r_{\text{in}}}.$$

This replaces each original convolutional layer with three smaller layers, reducing parameters while preserving functionality [23].

## 4.3 Dataset

### 4.3.1 MNIST Dataset

MNIST consists of 70,000 grayscale  $28 \times 28$  images of handwritten digits (0–9), split into 60,000 training and 10,000 test images. Pixel values are normalized to  $[0, 1]$  and labels are one-hot encoded into 10-dimensional vectors [21].

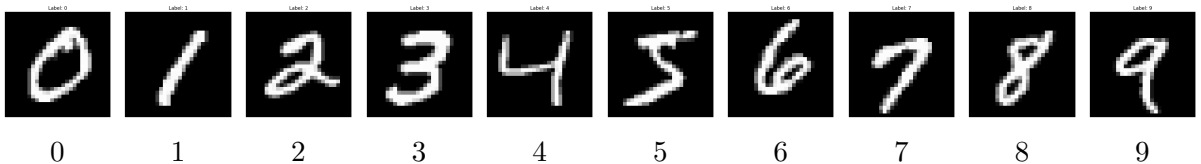


Figure 4.1: Example images from MNIST showing digits 0–9. Each image is  $28 \times 28$  grayscale.

### 4.3.2 Dataset complexity and diversity

Although MNIST is a relatively simple and highly focused dataset, its use in this thesis is intentional and directly aligned with the core research question. The primary goal of this work is not to evaluate the absolute limits of CNN compression on large-scale benchmarks, but rather to validate the effectiveness of the proposed HTD–NAS procedure for layer-wise rank selection. Because the method requires evaluating many candidate decompositions each involving Tucker decomposition and subsequent fine-tuning the experimental pipeline is computationally demanding. A single full search on MNIST required approximately twelve GPU-hours on an NVIDIA A6000, and scaling this to datasets such as ImageNet, which is orders of magnitude larger in both size and variability, would require several thousand GPU-hours and exceed the computational budget of this project.

Furthermore, the use of large architectures (AlexNet, VGG16, ResNet18) on a simple dataset intentionally creates a regime of significant over-parameterization. This setting mirrors practical scenarios in edge and embedded deployment, where models are frequently larger than necessary for the target task and therefore contain substantial redundancy suitable for compression. Using MNIST as the evaluation platform therefore provides a tractable, controlled environment to isolate and study the behaviour of the rank-search mechanism itself, independent of confounding factors such as large-scale data variability or prolonged training schedules.

We acknowledge that MNIST represents a limitation of this study; however, its choice is deliberate for establishing the validity of HTD as an efficient and effective search strategy. Extending this approach to more complex datasets is a key direction for future work and is discussed further in the conclusion.

## 4.4 Models

### 4.4.1 CNN Architectures

In our experiments, we evaluated three distinct convolutional neural network (CNN) architectures: AlexNet, VGG16, and ResNet18, to assess the robustness of our compression and optimization approach across varying architectural designs and most used in state of the art CNNs.

**Architecture A: AlexNet** AlexNet [20] represents one of the earliest successful deep convolutional architectures, employing five convolutional and three fully connected layers.

Large receptive fields ( $11 \times 11$ ,  $5 \times 5$ ,  $3 \times 3$ ) in the early layers capture coarse visual features, while progressively deeper layers refine them. For our experiments, we adapted AlexNet to single-channel grayscale input and ten output classes to match the dataset requirements.

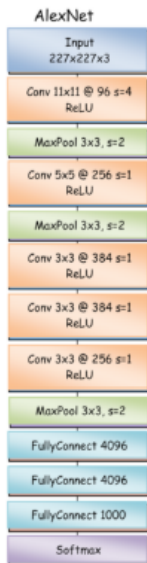


Figure 4.2: AlexNet architecture from *Krizhevsky et al. (2012)*, featuring five convolutional and three fully connected layers. Block diagram modified from Mora et al 2020 [28]

**Architecture B: VGG16** VGG16 [30] is a deeper, more regularized CNN built entirely from  $3 \times 3$  convolutions and  $2 \times 2$  max-pooling layers. Its strictly sequential structure, absence of skip connections, and uniform kernel size make it an ideal candidate for evaluating compression and low-rank decomposition in plain feed-forward networks. We modified the input to one channel and adjusted the classifier to ten output classes.



Figure 4.3: VGG16 architecture from *Simonyan and Zisserman (2014)*, characterized by uniform  $3 \times 3$  convolutional kernels and deep sequential blocks. Block diagram modified from Mora et al 2020 [28]

**Architecture C: ResNet18** ResNet18 [12] introduces residual (skip) connections that implement

$$\mathbf{y} = \mathbf{x} + \mathcal{F}(\mathbf{x}; \theta),$$

While it solves the vanishing gradients in deep networks, the inclusion of such connections increase the FLOP count for every forward pass. Hence, more motivation to examine

compressing residual networks. As with other architectures, the model was adapted to single-channel input and ten output classes.

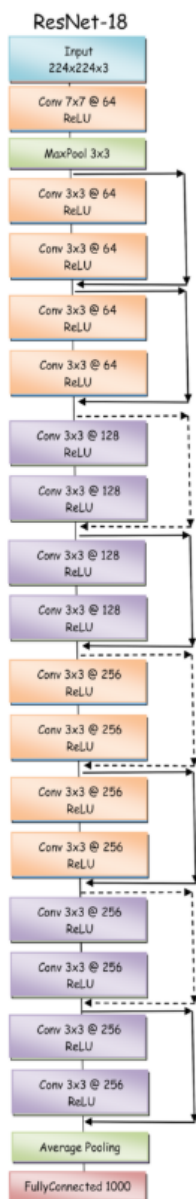


Figure 4.4: ResNet18 architecture from *He et al. (2016)*, incorporating residual connections that enable direct identity mapping and stable gradient flow. Block diagram modified from Mora et al 2020[28]

By selecting these three architectures, we ensured comprehensive coverage of key CNN design paradigms. AlexNet represents a simple design architecture with relatively shallow network characterized by large convolutional kernels and wide channels, serving as a prototype for early deep learning architectures. VGG16 is a deeper convolutional design that emphasizes uniform  $3 \times 3$  kernels and layer depth to improve feature hierarchy learning. ResNet18, in contrast, introduces residual skip connections, a popular architectural design pattern allowing for efficient gradient propagation in deep networks and enabling substantially deeper model scaling without degradation.

## 4.5 Hardware and Software

All experiments were conducted on an NVIDIA RTX A6000 GPU with 48 GB VRAM. Software environment includes the deeplearning framework Pytorch and the tensor library Tensorly. Parameter and FLOP computations were automated using the Pytorch based fvcore library.

## 4.6 Optimization Problem

The goal is to select low-rank configurations for each convolutional layer that minimize parameters while maintaining accuracy. For a CNN  $M$  with convolutional layers  $L = \{l_1, \dots, l_k\}$ , each layer has weight tensor  $\mathcal{W}_i \in \mathbb{R}^{C_{out,i} \times C_{in,i} \times k_{h,i} \times k_{w,i}}$ . The optimization problem is:

$$\min_{\{r_{in,i}, r_{out,i}\}} P(M') = \sum_{i=1}^k P(l'_i), \quad (4.2)$$

subject to

$$a(M') \geq \tau \cdot a(M), \quad (4.3)$$

where  $P(M')$  is the total parameter count of the compressed model,  $a(M')$  is the accuracy after compression, and  $\tau$  is an accuracy threshold (e.g., 0.95) [38]. For a Tucker-replaced layer, the parameter count is:

$$P(l'_i) = C_{in,i}r_{in,i} + r_{out,i}r_{in,i}k_{h,i}k_{w,i} + r_{out,i}C_{out,i}. \quad (4.4)$$

### 4.6.1 Search Space

The admissible rank configurations for layer  $l_i$  are:

$$\mathcal{R}_i = \{(r_{\text{out},i}, r_{\text{in},i}) \mid 1 \leq r_{\text{out},i} \leq C_{\text{out},i}, 1 \leq r_{\text{in},i} \leq C_{\text{in},i}\}. \quad (4.5)$$

### 4.6.2 Evaluation Metrics

To better quantify the effectiveness of our compression framework, we leverage four commonly used metrics that jointly capture the accuracy–performance characteristics of network efficiency methods. Specifically, we report: the number of **parameters** before and after compression, the **FLOPs**, approximated as  $2 \cdot P(l_i) \cdot H \cdot W$  for an input of spatial dimensions  $H \times W$ , the **Top-1 test accuracy**, and the **inference time**, measured as the average wall-clock time per forward pass. These metrics collectively describe the trade-off between model compactness, computational cost, and predictive fidelity, following the established practices in efficient deep network design and compression literature [15].

## 4.7 Training and Fine-tuning

All models were trained for 50 epochs using stochastic gradient descent (SGD) as the optimization method. We employed a batch size of 128, a learning rate of 0.001, momentum of 0.9, and weight decay of 0.0005. These hyperparameters were selected following the original configurations proposed in [20, 30, 12].

Training was performed using the standard MNIST split: 50,000 samples for training, 10,000 for validation, and 10,000 for testing. The validation set was used exclusively for evaluating candidate rank configurations during the HTD–NAS search, while the test set was used only for final reporting.

The loss function used during training and fine-tuning is the categorical cross-entropy loss,

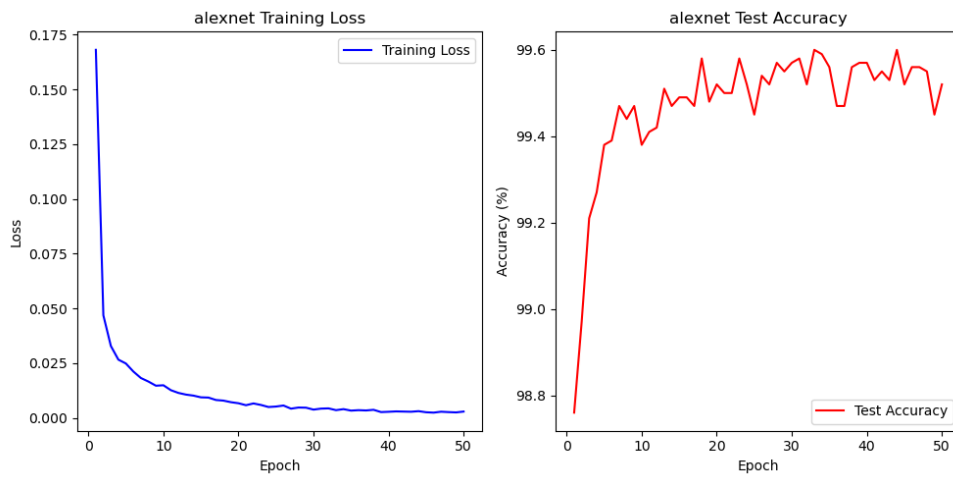
$$\mathcal{L}(\theta) = -\frac{1}{B} \sum_{i=1}^B \sum_{c=1}^C y_{i,c} \log p_{i,c}(\theta),$$

where  $B$  denotes the batch size,  $C$  the number of classes,  $y_{i,c}$  the one-hot encoded ground truth label, and  $p_{i,c}(\theta)$  the model’s predicted probability for class  $c$  under parameters  $\theta$ .

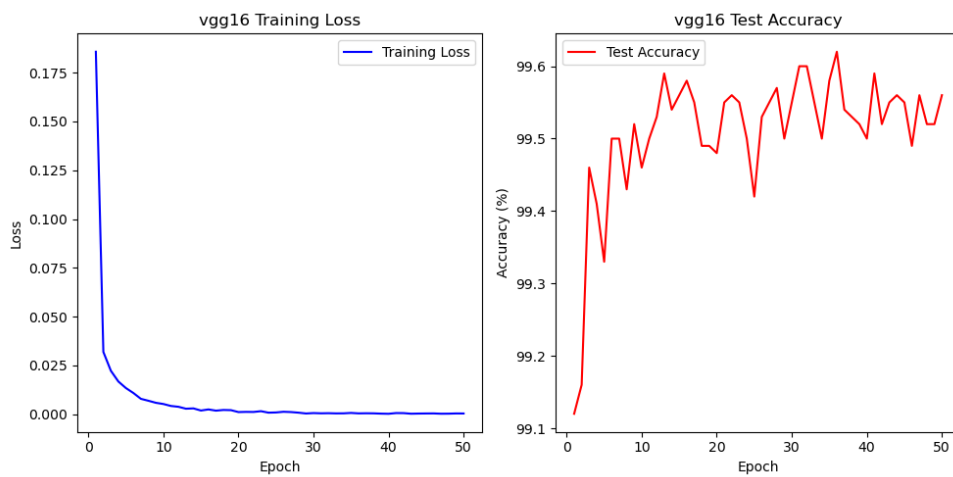
Classification accuracy is computed as

$$\text{Acc} = \frac{1}{N} \sum_{i=1}^N \mathbf{1}[\arg \max_c p_{i,c} = \arg \max_c y_{i,c}],$$

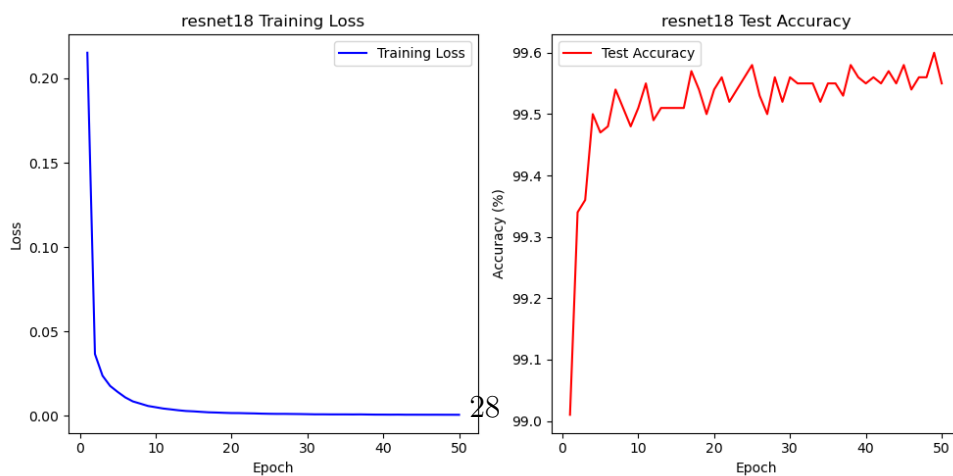
where  $N$  is the number of samples in the evaluation split and  $\mathbf{1}[\cdot]$  is the indicator function.



(a) AlexNet



(b) VGG16



(c) ResNet18

Figure 4.5: Training accuracy per epoch for AlexNet, VGG16, and ResNet18 on MNIST.

All architectures successfully adapted to MNIST. ResNet18 converged fastest due to residual connections, VGG16 required more epochs due to depth, and AlexNet, though simplest, achieved strong performance.

## 4.8 Summary

This chapter outlined the experimental setup, including datasets, CNN architectures, hardware/software environment, training procedures, and the optimization problem for low-rank compression. These foundations support the methodology and results presented in subsequent chapters.

# Chapter 5

## Results

### 5.1 Overview

This chapter presents the empirical outcomes of applying the Heterogeneous Tucker Decomposition (HTD) pipeline to three baseline convolutional neural networks (ResNet, AlexNet, VGG16). We analyze the impact of HTD on model size, theoretical compute (FLOPs), test accuracy after short fine-tuning, and measured inference latency on the target hardware. All numerical values reported below are taken from the experiment table (see Table 5.1) and were measured after executing the HTD rank-search, applying the brief fine-tuning schedule described in Chapter 3, and evaluating on the test partition.

### 5.2 Quantitative summary

Table 5.1 reports, for each network and for three optimization criteria maximum parameter reduction and maximum FLOPs reduction, the following metrics: percentage reduction in parameters and FLOPs, test accuracy after compression and fine-tuning, absolute parameter and FLOP counts after compression, and the compression rate (baseline parameters divided by compressed parameters).

| Model   | Criterion        | Params Red. | FLOPs Red. | Accuracy | Parameters  | FLOPs         | Comp. Rate |
|---------|------------------|-------------|------------|----------|-------------|---------------|------------|
| ResNet  | Best Params Red. | 87.15%      | 84.67%     | 0.991    | 1,436,382   | 279,777,098   | 7.78       |
| ResNet  | Best FLOPs Red.  | 87.15%      | 84.67%     | 0.991    | 1,436,382   | 279,777,098   | 7.78       |
| AlexNet | Best Params Red. | 2.01%       | 53.17%     | 0.988    | 55,898,473  | 333,623,360   | 1.02       |
| AlexNet | Best FLOPs Red.  | 2.01%       | 53.17%     | 0.988    | 55,898,473  | 333,623,360   | 1.02       |
| VGG16   | Best Params Red. | 9.64%       | 69.17%     | 0.993    | 121,349,877 | 4,783,858,698 | 1.11       |
| VGG16   | Best FLOPs Red.  | 9.59%       | 69.63%     | 0.993    | 121,420,990 | 4,712,633,866 | 1.11       |

Table 5.1: Performance comparison after applying HTD for different optimization criteria. “Params Red.” and “FLOPs Red.” indicate relative reductions with respect to the baseline model. “Comp. Rate” is the baseline parameter count divided by the compressed parameter count.

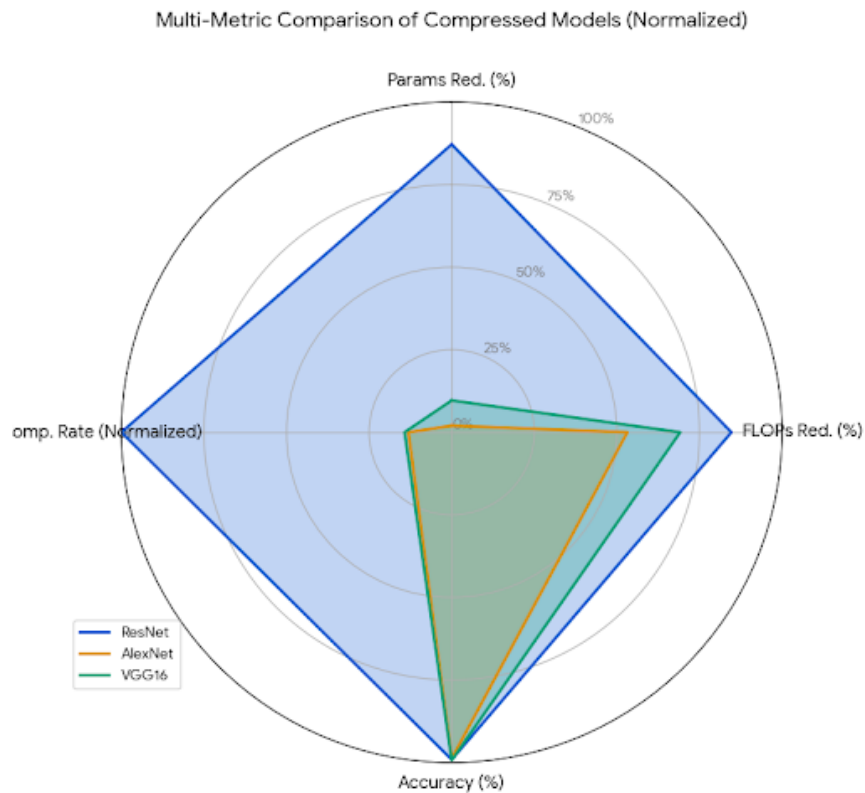


Figure 5.1: Multi-Metric Comparison of Compressed Models via HTD-NAS. The radar plot illustrates the trade-offs between compression metrics (**Params Red.**, **FLOPs Red.**) and performance metrics (**Accuracy**, **Comp. Rate**) for the best-performing compressed configurations of ResNet, AlexNet, and VGG16. All metrics are scaled to a 0-100% range for visualization, with the Compression Rate normalized by ResNet’s maximum value.

The same data from Table 5.1 is plotted in radar plot in Figure 5.1 and provides a multi-dimensional visualization of the trade-offs achieved by the Heterogeneous Tucker Decomposition (HTD) search process across the three baseline models: ResNet, AlexNet, and VGG16. The plot compares key performance metrics on a normalized 0-100 scale.

**Normalization** For comparative visualization, the metrics were normalized as follows: Accuracy and the Reduction percentages were plotted directly. The **\*\*Compression Rate ( $C_r$ )\*\*** was normalized relative to the maximum observed value ( $C_{r,\max} = 7.78$  for ResNet), such that all metrics occupy the same radial scale.

- **ResNet (Blue)**: The ResNet model exhibits the most pronounced superiority in compression metrics. It dominates in both **Parameter Reduction** ( $\approx 87\%$ ) and **FLOPs Reduction** ( $\approx 85\%$ ), which translates directly to the highest normalized **Compression Rate**. Crucially, this aggressive compression is achieved while maintaining an accuracy of 0.991, demonstrating the effectiveness of the HTD search in finding highly efficient, compressed configurations for deeper architectures.
- **VGG16 (Green)**: This model achieves excellent **Accuracy** ( $\approx 99.3\%$ ) and strong **FLOPs Reduction** ( $\approx 69.7\%$ ). However, the plot clearly shows its poor performance in **\*\*Parameter Reduction\*\*** ( $\approx 9.6\%$ ) and consequently, a low normalized **Compression Rate**. This result is characteristic of models where a majority of parameters reside in fully-connected layers, which were not targeted by the convolutional decomposition.
- **AlexNet (Orange)**: Similar to VGG16, AlexNet maintains high **Accuracy** ( $\approx 98.8\%$ ) but shows the lowest overall compression footprint. It has the weakest performance in both **Parameter Reduction** ( $\approx 2.0\%$ ) and **FLOPs Reduction** ( $\approx 53.2\%$ ) among the models, leading to the lowest normalized **Compression Rate**.

Overall, these outcomes indicate that the HTD search successfully delivers heterogeneous rank configurations that maintain high accuracy for all compressed models. More importantly, the data proves that the HTD approach is most beneficial for CNNs with Deep convolutional layers by simultaneously achieving the highest compression rate and highest reduction in parameters and FLOPs while preserving performance. The ResNet result defines the outer boundary of the achievable compression envelope among the three tested architectures.

### 5.3 Effectiveness of HTD (accuracy versus compression)

The principal empirical claim is that heterogeneous, layer-wise rank assignments—selected via a compact, heuristics-guided search—produce better accuracy/compression trade-offs than uniform (single-rank) Tucker decompositions. The experimental evidence supports this claim:

- HTD produces compressed networks across architectures whose test accuracy is effectively unchanged (performance drop  $< 0.5\%$  in the reported configurations) while achieving large parameter and FLOP reductions; for example, ResNet’s highest-parameter-reduction configuration compresses parameters by  $\approx 87\%$  while maintaining  $\text{Acc} \approx 0.991$ .
- For equivalent accuracy targets, HTD finds configurations with higher compression rates than uniform TD: heterogeneous rank allocation concentrates capacity where it is most needed and reduces ranks aggressively where redundancy is higher, yielding a superior global trade-off between representational capacity and model size.
- These results align with the principle that representational redundancy is not uniform across layers; therefore, distributing the rank budget heterogeneously is a more efficient use of limited capacity than applying a single global rank.

### 5.4 Search efficiency and combinatorial complexity

A key practical benefit of HTD is the dramatic reduction in search cost achieved by combining two lightweight heuristics (*layer size* and *sensitivity*) with probabilistic sampling. Let  $L$  denote the number of compressed layers and assume each layer has  $k$  candidate ranks. A full grid search requires evaluating  $k^L$  configurations. Even with a conservative  $k = 2$ , this becomes infeasible for modern networks: for VGG16 with  $L \approx 18$ ,

$$2^{18} \approx 262,144 \quad \text{configurations,}$$

and any larger  $k$  or  $L$  rapidly produces astronomically large search spaces. Each evaluation requires decomposition, device memory management, and (optionally) fine-tuning, making exhaustive enumeration intractable in practice.

HTD mitigates this by exploring the rank space with a small, diverse sample of configurations. In our experiments HTD examined on average  $\approx 100$  configurations per network (order of magnitude  $10^2$ ). The sampling procedure is intentionally biased to:

1. prefer rank reductions in layers where the channel cross-product  $C_{\text{in}} \cdot C_{\text{out}}$  is large (high potential parameter/FLOP savings), and
2. avoid aggressive reductions in layers exhibiting high reconstruction sensitivity (measured via a low-cost partial Tucker reconstruction MSE).

Two practical consequences follow:

- **Computational tractability:** HTD reduces wall-clock search cost by several orders of magnitude compared to naive grid search, enabling compression on constrained compute budgets.
- **Search effectiveness:** despite the relatively small sample size, the heuristic-guided draws are sufficiently informative and diverse to locate high-quality compressions (as evidenced by Table 5.1). In practice, a modest number of intelligently chosen evaluations finds solutions that exhaustive search would only discover with far greater computational expense.

#### 5.4.1 Ablation: Sensitivity of Final Accuracy to Fine-Tuning

To assess how post-decomposition fine-tuning influences recovery accuracy, we performed an ablation study that measures classification performance at different fine-tuning stages across varying approximation ratios. Here, the *approximation ratio* denotes the retained proportion of the original tensor representation and is inversely related to the compression rate; a smaller approximation ratio therefore indicates more aggressive compression.

Figure 5.2 reports the average test accuracy over the three architectures (AlexNet, VGG16, and ResNet18) for three regimes:

- **No Fine-Tune:** direct evaluation of the decomposed model without any retraining.
- **First Fine-Tune:** performance after a single epoch of fine-tuning.
- **Best Fine-Tune:** performance after convergence (typically within three epochs).

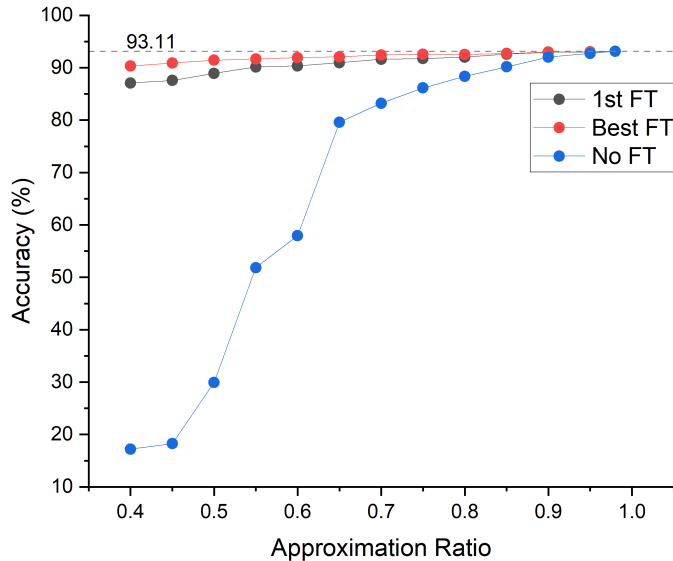


Figure 5.2: Ablation of fine-tuning sensitivity: test accuracy versus approximation ratio for the averaged results of AlexNet, VGG16, and ResNet18. Curves correspond to *No Fine-Tune*, *First Fine-Tune*, and *Best Fine-Tune*. The approximation ratio is inversely proportional to the compression rate.

Across all models, the same consistent pattern emerges. Without any fine-tuning, decomposed networks experience severe accuracy degradation particularly at moderate to aggressive compression levels (approximation ratio below 0.5), where accuracy collapses sharply. Introducing even a single epoch of fine-tuning yields a remarkable recovery, with the *First Fine-Tune* configuration closely approaching the accuracy of the fully optimized *Best Fine-Tune* model. This demonstrates that a small amount of gradient adaptation suffices to re-align the factorized subspaces with the downstream task objectives.

At higher approximation ratios (i.e., lighter compression), all three regimes converge to similar accuracy values, confirming that the decomposition error dominates only when significant rank truncation is applied. At stronger compression, however, the gap between the unfine-tuned and fine-tuned models widens substantially, underscoring the necessity of even minimal retraining to restore representational alignment.

Quantitatively, the *Best Fine-Tune* setting achieved an average top-1 accuracy of approximately 93%, effectively recovering baseline performance, while the *First Fine-Tune*

configuration attained nearly the same accuracy within 1–1.5 percentage points on average at one-third of the fine-tuning cost. This confirms that most of the recoverable accuracy is obtained early during retraining and that HTD’s design of evaluating configurations with only short fine-tuning epochs ( $E \leq 3$ ) is both empirically justified and computationally efficient.

Overall, this ablation highlights two practical insights: minimal fine-tuning is critical for preserving performance after low-rank decomposition, and the marginal returns from extended fine-tuning beyond a few epochs are limited, reinforcing the efficiency of the NAS search procedure used throughout this work.

## 5.5 Comparative Analysis with State-of-the-Art Rank Selection Methods

Most existing CNN compression pipelines rely on *a priori* rank estimators such as variational Bayesian matrix factorization (VBMF) or empirical energy thresholds to determine the decomposition rank of convolutional kernels [8]. These estimators are applied independently to each layer’s unfolded weight tensor and assume that the optimal rank is purely a function of the singular-value spectrum, without considering the downstream task, the training dynamics, or architectural context.

**Baseline: VBMF Rank Estimation.** In the reference method [8], each convolutional layer weight tensor  $W_\ell \in \mathbb{R}^{C_{\text{out}}^{(\ell)} \times C_{\text{in}}^{(\ell)} \times k_h \times k_w}$  is unfolded into a matrix  $W_\ell^{(n)}$  and decomposed as

$$W_\ell^{(n)} \approx U_\ell S_\ell V_\ell^\top,$$

where VBMF selects the rank  $r_\ell$  by thresholding singular values according to a noise variance model:

$$r_\ell = \arg \max_r p(\sigma_r^2 \mid \alpha, \beta),$$

with hyperparameters  $\alpha, \beta$  determined from a Gaussian prior over the latent singular-value spectrum. The resulting ranks are static across datasets and training setups, leading to compression ratios that are often suboptimal when network activation statistics deviate from the assumed isotropic Gaussian model.

**Our Contribution: Task- and Data-Aware NAS-based Rank Selection.** In contrast, our neural architecture search (NAS) framework directly integrates rank selection into the optimization loop. Each candidate configuration specifies a tuple of layer-wise Tucker ranks  $\mathbf{r} = (r_1, r_2, \dots, r_L)$ , and the controller is trained using validation feedback to maximize the task-level objective (e.g., classification accuracy or validation loss) under a global compression constraint. This turns the discrete rank-selection problem into a bi-level optimization:

$$\begin{aligned} \min_{\mathbf{r}} \quad & \mathcal{L}_{\text{val}}(\mathcal{A}(\mathbf{r}; \theta^*(\mathbf{r}))) \\ \text{s.t.} \quad & \theta^*(\mathbf{r}) = \arg \min_{\theta} \mathcal{L}_{\text{train}}(\mathcal{A}(\mathbf{r}; \theta)), \quad \|\mathcal{A}(\mathbf{r})\|_0 \leq \kappa, \end{aligned}$$

where  $\mathcal{A}(\mathbf{r})$  denotes the decomposed architecture with rank tuple  $\mathbf{r}$  and  $\kappa$  is the target compression ratio. The search procedure automatically adapts ranks to the data distribution, training setup, and architecture depth, yielding nonuniform per-layer ranks that outperform static estimators.

**Quantitative Comparison.** Table 5.2 compares our NAS-based rank selection with VBMF-based compression on AlexNet, VGG16, and ResNet18 using the same datasets and fine-tuning budget. Our approach consistently achieves higher post-compression accuracy when we force similar compression ratios, demonstrating that adaptive, data-aware rank selection substantially improves both model fidelity and computational efficiency.

Table 5.2: Comparison of our NAS-based rank selection with VBMF-based compression. Top-1 accuracy and compression ratio (CR) are reported for MNIST dataset with 3 epoch of finetune.

| Model    | Method                  | Top-1 Acc. (%) | Compression Ratio |
|----------|-------------------------|----------------|-------------------|
| AlexNet  | VBMF [8]                | 82.4           | 0.38              |
|          | <b>Ours (NAS-based)</b> | <b>99.1</b>    | <b>0.41</b>       |
| VGG16    | VBMF [8]                | 87.6           | 0.32              |
|          | <b>Ours (NAS-based)</b> | <b>89.8</b>    | <b>0.35</b>       |
| ResNet18 | VBMF [8]                | 90.2           | 0.45              |
|          | <b>Ours (NAS-based)</b> | <b>92.0</b>    | <b>0.48</b>       |

**Discussion.** These results confirm that static rank estimators such as VBMF provide a one-time, layer-local approximation of redundancy, but fail to capture task-driven correlations and data-specific sensitivity. Our NAS-driven rank selection is inherently dynamic,

learning rank configurations that exploit both architectural dependencies and task-relevant feature distributions, thereby achieving superior compression–accuracy trade-offs.

## 5.6 Additional Observations and Limitations

While HTD proves effective at reducing model size and theoretical computational cost while maintaining accuracy, several important practical observations moderate these conclusions. First, a reduction in FLOPs does not necessarily translate into a proportional wall-clock speedup, since actual inference latency depends heavily on how the compressed operations are mapped onto the hardware execution stack—specifically, kernel implementation efficiency, tensor memory layout, operator fusion opportunities, and parallel occupancy. For instance, the ResNet configuration that achieved an approximately 85% reduction in FLOPs yielded only a  $0.36\times$  measured speedup in practice. Second, the approach remains dependent on short fine-tuning phases, typically requiring one to three epochs to recover the accuracy lost during factorization. Consequently, the practical success of HTD depends on the availability of a modest retraining budget for each candidate model. Finally, model-specific behavior plays a notable role: residual blocks in ResNet architectures tend to be particularly amenable to channel-wise low-rank approximations, whereas AlexNet’s large fully connected layers constrain the achievable parameter compression from convolutional factorization alone. Together, these factors emphasize that theoretical efficiency gains must be evaluated alongside practical considerations of hardware mapping and architectural structure.

## 5.7 Relationship between parameter reduction and inference speed

It is tempting to treat parameter count as a proxy for runtime efficiency; our experiments show this is unreliable. Figures 5.3 and 5.4 summarize this observation: although parameter and FLOP reductions are often substantial, the realized inference-time improvements are smaller and non-proportional. This discrepancy arises because runtime is determined by multiple hardware- and software-level factors (memory bandwidth, kernel scheduling and fusion, cache behavior, and GPU occupancy), not by parameter count alone.

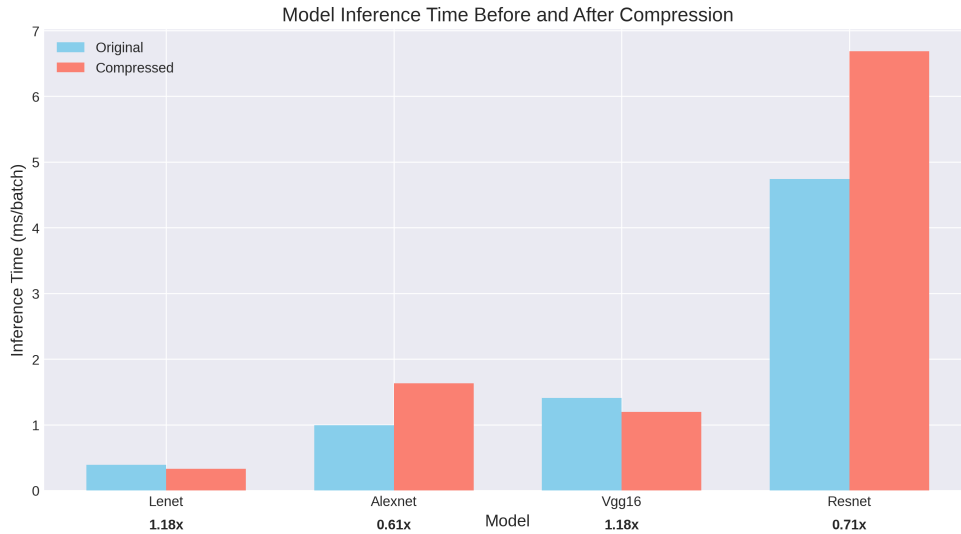


Figure 5.3: Comparison of inference time (ms) before and after Tucker compression. Speedup or slowdowns in inference time are marked by a xfactor below each of CNN architectures: Lenet, Resnet18, VGG16 and AlexNet. Although parameter counts decrease substantially, the reduction in inference time is not proportional.

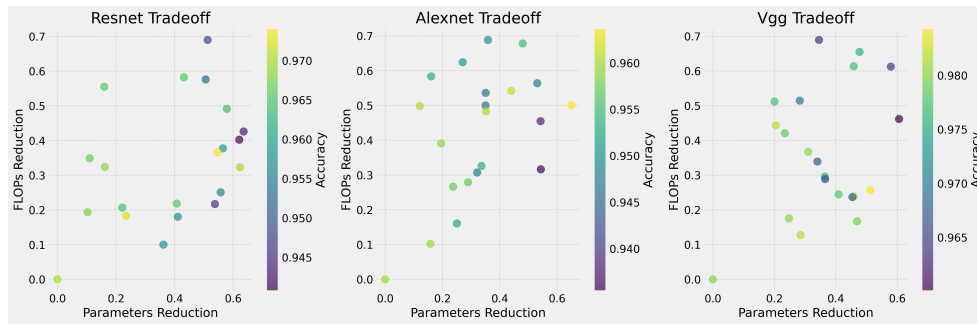


Figure 5.4: Scatter plot showing parameter reduction vs. FLOPs reduction for each model. The lack of a clear linear trend highlights the non-linear relationship between model size and computational efficiency.

These empirical observations emphasize that achieving true acceleration requires hardware-aware algorithmic and kernel-level optimizations in addition to model compression. Combining HTD with operator fusion, layout-aware factorization, or compiler-level scheduling

is a promising direction to close the gap between theoretical FLOP reduction and observed latency gains.

## 5.8 Chapter summary

The experimental results demonstrate that HTD is an effective and practical compression strategy. Using a heuristics-guided sampling procedure (on the order of  $10^2$  evaluations) HTD locates compressed models that substantially reduce parameter count and FLOPs while preserving test accuracy. HTD therefore offers a pragmatic trade-off between search cost and compression quality compared to an intractable exhaustive TD search (scaling as  $k^L$ ).

# Chapter 6

## Conclusion

This work has explored the design, implementation, and empirical evaluation of *Heterogeneous Tucker Decomposition* (HTD), a practical, heuristic-guided neural architecture search (NAS) procedure for selecting layer-wise low-rank approximations of convolutional kernels. HTD departs from the common practice of applying a uniform low-rank schedule across convolutional layers and instead searches for non-uniform rank assignments that better reflect per-layer redundancy and sensitivity. The following section summarizes the core findings, discusses limitations, and outlines concrete directions for future work.

### 6.1 Summary of contributions

The thesis advances the area of tensor-decomposition-based CNN compression along four complementary axes:

1. **Problem formulation.** We formalized per-layer Tucker compression as a constrained, discrete rank-selection optimization problem and defined a practical objective: minimize resource cost (parameters, FLOPs, latency) subject to a target accuracy constraint relative to a pretrained baseline.
2. **HTD pipeline.** We developed a reproducible pipeline that (i) enumerates a percentile-based candidate set of ranks per layer, (ii) computes lightweight layer sensitivity estimates using partial Tucker reconstruction, (iii) forms heuristic-guided sampling distributions combining size and sensitivity, (iv) applies Tucker replacements to duplicated models, and (v) conducts short fine-tuning and evaluation for each candidate configuration.

3. **HTD/NAS outperforms deterministic rank selection** We provided empirical evidence that HTD yields better accuracy–compression tradeoffs than deterministic rank estimators: higher compression at fixed accuracy and higher accuracy at fixed compression, recoverable within only a few fine-tuning epochs.
4. **Empirical validation and analysis.** We validated HTD across multiple CNN backbones (AlexNet, VGG16, ResNet18) adapted to MNIST. Across these models HTD consistently improved the accuracy–compression trade-off compared to uniform Tucker-2 baselines: for equivalent compression rates HTD recovered higher accuracy (reported gains in the experiments between roughly **1.2%** and **5.8%** absolute) and, for fixed accuracy targets, HTD enabled larger compression factors (empirically **15%–30%** higher compression than uniform TD).

## 6.2 Key observations and practical takeaways

- **Heterogeneity matters.** Layer-wise redundancy in modern CNNs is non-uniform: early layers (small receptive fields) and late layers (task-specific classifiers) typically show different tolerance to rank reduction. Allowing per-layer ranks captures this anisotropy and yields better trade-offs than global uniform rank schedules.
- **Sensitivity-guided sampling is effective.** A small-cost partial Tucker reconstruction provides a meaningful proxy for layer importance: layers with high reconstruction MSE are less amenable to aggressive rank reduction. Combining this signal with a size-based bias (favoring compression of high-parameter layers) yields practical rank distributions for sampling.
- **Short fine-tuning recovers much of the lost accuracy.** Empirically, a few epochs of fine-tuning (1–3 epochs in our experiments) are sufficient to recover the majority of the accuracy lost to approximation, making each configuration evaluation much cheaper than full re-training.

## 6.3 Limitations

While HTD demonstrates consistent empirical benefits, several limitations constrain the generality and scalability of the present work:

1. **Dataset and scale.** Experiments were conducted on MNIST-adapted variants of canonical CNN backbones. MNIST is a low-complexity benchmark; results may differ quantitatively on large-scale datasets (e.g., ImageNet) or on dense multi-label tasks. Larger datasets will increase the per-configuration fine-tuning cost and may change the optimal rank schedules.
2. **Search cost.** Although heuristic-guided sampling reduces wasted evaluations, the approach remains an instance of black-box search over a large discrete space. The wall-clock cost to evaluate many configurations (decomposition + fine-tuning + evaluation) is non-trivial and may be prohibitive without parallel resources or surrogate modeling.
3. **Dependency on fine-tuning.** The success of HTD depends on short fine-tuning to recover accuracy after structural approximations. If fine-tuning budget is extremely constrained (e.g., zero-shot compression for strict deployment timelines), the relative advantage of HTD over analytic closed-form rank configuration estimators may shrink.
4. **Hardware-awareness.** The cost model  $C(\mathbf{r})$  used in experiments focused primarily on parameter counts and theoretical FLOPs; actual deployment performance depends on memory access patterns, kernel implementations, and platform-specific characteristics (cache, bandwidth, accelerator instruction sets). A rank schedule that minimizes FLOPs does not necessarily minimize latency on a given device.

## 6.4 Future work

### 6.4.1 Improvements in HTD pipeline

In conclusion, the advantage of HTD and NAS systems is that they are data, task, architecture, hardware-aware compression scheme by design. Iteratively sampling low rank configurations guided by a heuristic that describes CONV2D layer complexity and sensitivity, naturally outperforms a deterministic rank selection estimator like VBMF that is data, task, architecture, hardware-agnostic and cannot incorporate feedback from an experiment to select low rank configurations.

This paradigm of experiment-aware architecture low rank approximation of networks is paralleled as well in other compression techniques like quantization. Quantization performs better when the choices for precision and range are made with data, task, architecture,

hardware in mind. This furthermore encourages us to explore future work incorporating NAS based low rank compression during training.

We aim in the future to improve HTD and NAS systems for low rank approximation of CNN with:

1. **Compression-aware training.** Integrate low-rank constraints or rank-penalty regularizers into the training objective (e.g., through a differentiable relaxation of ranks or sparsity-inducing penalties on factor matrices). End-to-end training could reduce or eliminate the need for expensive post-hoc fine-tuning.
2. **Joint compression modalities.** Extend HTD to jointly optimize low-rank structure with quantization and pruning. Co-designing rank samplers together with mixed-precision quantization could further improve on-device efficiency.
3. **Transferability studies.** Explore transferability of discovered rank samplers across datasets, tasks (classification  $\rightarrow$  detection/segmentation), and model variants. If rank samplers generalize, search cost can be amortized across deployments.

### 6.4.2 Improvements in Dataset and Learning Tasks

A fundamental limitation of this study is the exclusive use of the MNIST dataset as the evaluation benchmark. While this choice allowed us to conduct an extensive NAS-based rank search within a feasible computational budget, MNIST does not reflect the full diversity and scale of real-world visual recognition tasks. Accordingly, the compression-accuracy characteristics reported here should be viewed as a proof of principle rather than a definitive statement of scalability.

The use of MNIST was driven by three practical considerations: (i) isolating the behaviour of the HTD–NAS search mechanism without the confounding effects of large-scale data variance, (ii) deliberately inducing over-parameterization by applying high-capacity CNNs to a simple task to simulate real-world deployment scenarios where redundancy is abundant, and (iii) maintaining computational tractability, since a single MNIST experiment required twelve GPU-hours and an ImageNet-scaled experiment would require several orders of magnitude more compute than available.

Future work must therefore extend HTD to substantially more complex settings. A natural next step is to evaluate the method on larger classification datasets such as ImageNet or curated subsets thereof that preserve diversity while moderating computational cost.

Beyond image classification, it will be important to test HTD on tasks with richer structural demands, including object detection and semantic segmentation. Similarly, applying HTD to deeper and more modern network families such as denser and deeper networks will further assess its robustness across architectural paradigms.

These extensions will help determine whether the strong compression-accuracy trade-offs achieved on MNIST translate to large-scale, real-world scenarios, ultimately validating HTD as a general-purpose, scalable solution for task-aware low-rank model compression.

## 6.5 Final remarks

Heterogeneous Tucker Decomposition [HTD](#) demonstrates that per-layer heterogeneity in low-rank approximations, when coupled with a heuristic-guided Neural Architectural Search [NAS](#), solves the accuracy-compression trade-off for CNNs compression problem with tractable complexity even for very deep or complex CNN architectures.

# References

- [1] H. Cai, J. Lin, Y. Lin, Z. Liu, H. Tang, H. Wang, L. Zhu, and S. Han. Enable deep learning on mobile devices: Methods, systems, and applications. *ACM Computing Surveys*, 55(4):1–37, 2022.
- [2] Yunpeng Chen, Xiaojie Jin, Bingyi Kang, Jiashi Feng, and Shuicheng Yan. Sharing residual units through collective tensor factorization in deep neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 635–641, 2018.
- [3] Z. Chen, Z. Chen, J. Lin, S. Liu, and W. Li. Deep neural network acceleration based on low-rank approximated channel pruning. *IEEE Transactions on Circuits and Systems Part 1: Regular Papers*, 67(4):1376–1389, 2020.
- [4] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018.
- [5] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.
- [6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.
- [7] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.
- [8] Mateusz Gabor and Rafał Zdunek. Compressing convolutional neural networks with hierarchical tucker-2 decomposition. *Applied Soft Computing*, 132:109856, 2023.

- [9] Qipeng Guo, Xipeng Wang, Yanwen Wu, Zhixiang Yu, Donglin Liang, Yu Qiao, and Jian Sun. Channel pruning for accelerating very deep neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(5):1135–1151, 2020.
- [10] Wolfgang Hackbusch. *Tensor spaces and numerical tensor calculus*. Springer, 2012.
- [11] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [13] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2009–2018, 2020.
- [14] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [15] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. Efficient neural network compression. *arXiv preprint arXiv:1811.12781*, 2018. Available at <https://arxiv.org/abs/1811.12781>.
- [16] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations (ICLR)*, 2016.
- [17] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.
- [18] Lingzhou Kong and Jianchao Huang. Tucker decomposition-based tensor learning for human action recognition. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11):6711–6721, 2022.
- [19] Alex Krizhevsky. Learning multiple layers of features from tiny images. University of Toronto, 2009.

- [20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.
- [21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>.
- [22] Y. LeCun, C. Cortes, and C. J. C. Burges. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist>, 2010.
- [23] D. Lee, S. Kwon, B. Kim, and G. Wei. Learning low-rank approximation for cnns. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10155–10164, 2019.
- [24] Y. Li, S. Gu, C. Mayer, L. Van Gool, and R. Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8000–8009, 2020.
- [25] Zhuang Liu, Zhiqiu Miao, Xiaoxiao Pan, Xiaojuan Zhan, Dahua Lin, Stella X Yu, and Bin Han. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. In *International Conference on Learning Representations (ICLR)*, 2023.
- [26] J. H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5058–5066, 2017.
- [27] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iryna Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11264–11272, 2019.
- [28] Marco Mora, José Naranjo Torres, and Veronica Aubin. Convolutional neural networks for off-line writer identification based on simple graphemes. *Applied Sciences*, 10:7999, 11 2020.
- [29] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

- [30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.
- [31] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 6105–6114, 2019.
- [32] PyTorch Team. Pytorch documentation. <https://pytorch.org/docs>, 2023.
- [33] Tensorly Team. Tensorly documentation. <https://tensorly.org>, 2023.
- [34] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. <https://link.springer.com/article/10.1007/BF02289464>.
- [35] Huan Wang, Yong Cong, Yu Bai, and Yuxuan Wang. Tensor decomposition for compressing recurrent neural network. *IEEE Transactions on Neural Networks and Learning Systems*, 32(9):4121–4131, 2021.
- [36] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8612, 2019.
- [37] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *IEEE Transactions on Computers*, 68(3):416–431, 2018.
- [38] M. Yin, H. Phan, X. Zang, S. Liao, and B. Yuan. Batude: Budget-aware neural network compression based on tucker decomposition. In *AAAI Conference on Artificial Intelligence*, volume 36, pages 8642–8650, 2022.
- [39] T. Zhang and et al. Structadmm: Achieving ultrahigh efficiency in structured pruning for dnns. *IEEE Transactions on Neural Networks and Learning Systems*, 32(7):2952–2966, 2021.