N° d'Ordre :
EDSPIC :

Université Sorbonne Paris Nord
ÉCOLE DOCTORALE GALILÉE

Ph.D. Thesis
by

Nosseiba Ben Salem

for the dgree of
Doctor of Computer Science

Efficient Compression of Pre-trained Neural Networks: Layer Selection, Pruning and Tensor Decomposition

Compression efficace de modèles de réseaux de neurones pré-entraînés : sélection de couches, élagage et décomposition tensorielle

defended on May 12, 2026 in front the following jury :

Thesis supervisor :	
Younès Bennani	Professor, Université Sorbonne Paris Nord
Reporters :	
Cyrille Bertelle	Professor, Université Le Havre Normandie
Ali Yahyaouy	Professor, Université Sidi Mohamed Ben Abdellah
Examiners :	
Christian Ausoni	Professor, Université Sorbonne Paris Nord
Thomas Papastergiou	Associate Professor, Université Sorbonne Paris Nord
Valérie Paradis	Professor, Universite Paris Cité
Nicoleta Rogovschi	Associate Professor (HDR), Universite Paris Cité
Besma Zeddini	Associate Professor (HDR), CY Cergy Paris Université

" If I have seen further, it is by standing upon the shoulders of giants
– Isaac Newton."

Acknowledgements

First and foremost, I would like to convey my deep and sincere appreciation to my supervisor, Prof. Younès Bennani for his permanent guidance and support throughout my PhD journey. His scientific insight, fruitful ideas as well as his trust and patience shaped this thesis and my approach to research.

I would also like to express my gratitude to Prof. Faouzi Boufares and Dr Thomas Papastergiou for their continuous help and insightful advices. I am also grateful to Dr Mélanie Courtine for her constant encouragement and kind words

I am extending my heartfelt thanks my colleagues and friends in the laboratory for the stimulating discussions, the shared frustrations and laughs.

Finally, I wish to thank my dear family for their love, prayers, and sacrifices, and my friends for their constant support and encouragement. Their presence and understanding have helped me endure the most difficult moments of this journey.

Nosseiba Ben Salem
March, 2026

University Sorbonne Paris Nord

Abstract

Laboratoire Informatique de Paris Nord

Doctor of Computer Science

Efficient Compression of Pre-trained Neural Networks: Layer Selection, Pruning and Tensor Decomposition

by Nosseiba BEN SALEM

Over the past decade, deep learning has established itself as a transformative tool across numerous scientific and industrial fields. The widespread adoption of large pre-trained models has further accelerated progress by enabling knowledge transfer across tasks and domains. However, the growing complexity and scale of these models, often comprising hundreds of millions of parameters, impose substantial computational and memory costs that hinder their deployment in data-constrained or resource-limited settings.

Although compression techniques for 2D architectures have received considerable attention, efficienty adapting pretrained models to 3D medical imaging tasks remains relatively underexplored. In this thesis, we investigate methods for compressing and accelerating pretrained deep neural networks while maintaining their predictive performance. We present three main contributions.

First, we propose an automatic layer selection strategy for transfer learning based on the Kullback-Leibler divergence between the weight distributions of the source and fine-tuned models. By identifying which layers encode transferable generic representations and which require task-specific retraining, our method achieves accuracy comparable to dynamic layer selection baselines while significantly reducing computational cost and eliminating any auxiliary trainable module.

Second, we adapt a pruning method that removes the least informative neurons by propagating the importance scores across successive layers to convolutional neural networks. This method captures inter layer dependencies and is applicable to both 2D and 3D network architectures, producing compact models without substantial loss of accuracy.

Third, we present a hybrid compression framework that combines Tucker tensor decomposition with structured channel pruning, specifically designed for 3D medical image segmentation. This strategy simultaneously reduces both model size and inference time, enabling to efficiently deploy deep networks in resource constrained clinical environments.

Overall, these contributions provide a practical and computationally efficient pathway towards the sustainable deployment of pretrained neural networks across different application domains.

Résumé

Au cours de la dernière décennie, l'apprentissage profond s'est imposé comme un outil transformateur dans de nombreux domaines scientifiques et industriels. L'adoption généralisée de grands modèles pré-entraînés a acceleré davantage ces progrès en permettant le transfert de connaissances entre tâches et domaines. Cependant, la complexité et la taille croissantes de ces modèles, comprenant souvent des centaines de millions de paramètres, engendrent des coûts computationnels et mémoriels considérables qui freinent leur déploiement dans des contextes à ressources limitées ou à données contraintes. Si les techniques de compression pour les architectures 2D ont fait l'objet d'une attention soutenue, l'adaptation efficace de modèles préentraînés aux tâches d'imagerie médicale 3D demeure un défi insuffisamment exploré.

Dans cette thèse, nous étudions des méthodes de compression et d'accélération de réseaux de neurones profonds préentraînés tout en préservant leurs performances prédictives. Nous présentons trois contributions principales.

Premièrement, nous proposons une stratégie automatique de sélection de couches pour l'apprentissage par transfert, fondée sur la divergence de Kullback-Leibler entre les distributions de poids du modèle source et du modèle affiné. En identifiant quelles couches encodent des représentations génériques transférables et lesquelles nécessitent un réentraînement spécifique à la tâche, notre méthode atteint des performances comparables aux approches dynamiques de sélection de couches, tout en réduisant significativement le coût computationnel et en éliminant tout module entraînable auxiliaire.

Deuxièmement, nous adaptons une méthode d'élagage structuré qui élimine les neurones les moins informatifs en propageant les scores d'importance à travers les couches successives du réseau. Cette approche tient compte des dépendances inter couches et peut être appliquée à des architectures 2D comme 3D, ce qui permet d'obtenir des modèles plus compacts sans perte notable de précision.

Troisièmement, nous introduisons un cadre de compression hybride combinant la décomposition tensorielle de Tucker et l'élagage structuré des canaux, spécifiquement conçu pour la segmentation d'images médicales 3D. Cette stratégie réduit simultanément la taille du modèle et le temps d'inférence, ce qui facilite le déploiement efficace de réseaux profonds dans des environnements cliniques contraints en ressources.

Prises ensemble, ces contributions offrent une voie méthodologique rigoureuse et computationnellement efficace vers un déploiement durable des réseaux de neurones pré-entraînés dans des domaines d'application variés.

Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Context and Motivation

Neural networks are one of the most popular models in machine learning yielding outstanding results in solving complex problems, such as computer vision. However, their good performance requires large parameters and higher floating point operations (FLOPs). The more complex the problem, the deeper and larger the models are. For instance, ResNet50 (He et al., 2016c) has approximately 25.6 million parameters. This problem limits the use of the predictive power in resources limited environment, thus their applicability to everyday applications for example, a clinic, or mobile phones . Another critical problem is the environmental impact especially with the rapid development of generative AI models.

Although these methods achieve good performance in multiple applications, it can be challenging in the case of medical images segmentation problems. Mainly due to the complex anatomical structures and internal variability of some organs like kidneys, a more precise human annotation is needed to segment fine structures. Also, often hardware or software issues can cause artifacts in medical images such as radio-frequency interference or due to movements of the patient during scanning, etc. therefore, tissues with low-contrast are difficult to segment and non-homogeneous texture causes ambiguous boundaries (Tushar et al., 2024).

In recent years, deep learning based approaches have shown exceptional performance when trained on a fully-annotated dataset. 3D CNN models for segmentation has been proposed following the U-shaped structrure such as U-net (Ronneberger et al., 2015a) and nnUnet3D (Isensee et al., 2018). these architectures allow the model to capture contextual information across slices for volumetric tasks. However, these models has a significant high parameter count and FLOP 3D segmentation has more geometric information The 3D segmentation generally has large scale data The required neural networks to solve such complex problem are generally over-parameterized and needs high computational cost. Most state-of-the art are not compact and has redundant features. Several solutions have been investigating reducing the running time and the hardware cost of the inference of DNN. Such as quantization, which represents weights and activations with lower numerical precision, knowledge distillation, where a smaller student network learns to mimic a larger teacher, pruning, which removes redundant parameters, and low-rank factorization, which decomposes weight tensors into a compact tesnors.

In (Denil et al., 2014), the authors showed that many parameters are not necessary

and small fraction of the weights are enough to predict 95% of the rest without decreasing the accuracy. These results motivated the exploration of low rank factorization. These observations motivates the need to study the redundancy in overparameterized networks to obtain a compact and efficient model in clinical settings.

These constraints collectively define a central scientific challenge: **how can the representational power of large pre-trained models be preserved while dramatically reducing their computational footprint?** This question is not merely a matter of engineering optimisation. It is, at its core, a question about the structure of the knowledge encoded in neural network weights, and about which parts of that knowledge are genuinely necessary for a given task.

The present thesis addresses this challenge through three complementary contributions. Rather than proposing a single monolithic compression algorithm, we investigate three distinct axes along which neural network redundancy can be identified and eliminated: the *selection of layers* to adapt during transfer learning, the *identification of neurons* that are genuinely necessary for prediction, and the *structural decomposition* of weight tensors into compact low-rank representations. For each axis, we develop both a principled algorithmic method and a formal theoretical framework that explains *why* compression works and under what conditions it can improve, rather than merely preserve, model performance.

1.2 Limitations of Existing Approaches

1.2.1 Transfer Learning and Fine-Tuning

Transfer learning with large pretrained models has become the default strategy in many domains, yet most practical fine tuning procedures remain ad hoc. Most transfer learning approaches either fine-tune all layers or simply choose manually a block of higher layers to adapt while freezing earlier layers, based on feature hierarchy rather than task-specific evidence. This heuristic choice can lead to over adaptation, where too many layers are updated and the model overfits the small target dataset, and also can lead to under adaptation, where not enough capacity is adapted and the model fails to capture target-specific structure. Existing dynamic or learned layer-selection methods partially address this by introducing additional gating or controller modules, but they often increase the number of trainable parameters, complicate optimisation, and provide limited theoretical insight into when negative transfer is avoided.

1.2.2 Neuron and Channel Pruning

Neuron and channel pruning are widely used to compress neural networks by removing units deemed unimportant, yet most existing methods rely on local importance measures that only reflect the global impact of pruning on the network's predictions. Common criteria, such as weight magnitude, activation norms, or gradient-based saliency, are typically computed layer by layer and ignore how errors introduced by pruning propagate through subsequent layers. A neuron with small local magnitude but strong downstream influence may be removed, while truly redundant neurons are kept, leading to suboptimal compression and unpredictable effects on accuracy. In addition, many pruning algorithms depend on repeated forward backward passes or second-order information to estimate importance, which increases computational cost.

1.2.3 Tensor Decomposition for 3D Networks

Three dimensional medical image segmentation networks, such as 3D U-Net , are particularly demanding in terms of memory and computation, and tensor decomposition has emerged as a promising tool to address this challenge. Existing work has shown that decomposing convolutional kernels into low rank factors can drastically reduce parameter counts and floating point operations while preserving segmentation accuracy after fine tuning. However, current tensor decomposition methods treats each channel identically and ignores filter sparsity regardless of their actual relevance to the segmentation task. In other words, tensor methods remove linear redundancy but do not identify unimportant channels. Moreover, choosing the right decomposition rank is nontrivial and usually done by hand, and aggressive low rank approximations can harm small structure segmentation accuracy. Conversely, pure channel pruning methods fail to exploit the multi dimensional structure of convolutional weights. Additionally combining both methods for 3D medical models is still not explored.

1.3 Research Questions

The limitations identified above give rise to three specific research questions that structure the scientific content of this thesis:

1. **Layer selection for transfer learning.** Given a pre-trained model and a target dataset, can we automatically identify, without introducing additional trainable modules, *which layers* encode representations genuinely relevant to the target task and should be fine-tuned? Can this criterion be formally justified through generalisation bounds and a guarantee of negative transfer prevention?

2. **Propagation-aware neuron pruning.** Can we define a neuron importance criterion that captures the *global* effect of neuron removal on the network's output, while remaining computable from weights alone without gradient computation? Can we prove that pruning in order of this criterion minimises worst-case prediction change, and that moderate pruning *improves* generalisation?

3. **Structured compression for 3D medical segmentation.** Can Tucker decomposition and importance-based channel pruning be combined into a principled two-stage pipeline that achieves high compression ratios while maintaining or improving segmentation accuracy? Can the accuracy improvement under compression be formally explained through implicit regularisation theory?

1.4 Contributions of this Thesis

Contribution 1 — Automatic layer selection via KL divergence (Chapter 3)

We propose an automatic criterion for selecting which layers to fine-tune, based on the Kullback-Leibler (KL) divergence between the weight distributions of the source model and the model obtained after an initial fine-tuning pass. Layers whose distributions have shifted significantly are selected for full optimisation; layers that remain stable are frozen to preserve their generic representations.

Theoretical contributions:

- *Theorem 3.2 (Generalisation bound).* Selective fine-tuning simultaneously reduces both the distribution divergence term and the complexity term of the transfer error bound, formally explaining its superiority over full fine-tuning and feature extraction.
- *Theorem 3.7 (Negative transfer prevention).* Under mild regularity conditions, our method is guaranteed to not perform worse than training from scratch regardless of domain shift.
- *Complexity analysis (Section 3.5.2).* We show that the cost of the KL-based selection is independent of the dataset size N, contrary to policy-based methods whose overhead scales as $\mathcal{O}(N)$.

Experimental findings: Evaluated our approach on eight public benchmarks (CUB-200, Stanford Cars, Flowers-102, WikiArt, Sketches, Stanford Dogs, Aircraft, MIT Indoor67), with selecting only as few as three layers achieves, the method achieves comparable accuracy to SpotTune on seven of the eight datasets, while preserving the same number of parameters as ResNet-50 and adding only negligible extra computation cost. The method outperforms both standard fine-tuning and L^2-SP regularisation on all datasets.

Contribution 2 — Propagation-aware pruning via HVS (Chapter 4)

We introduce the Heuristic for Variable Selection (HVS), a neuron importance criterion that computes the contribution of each neuron as the cumulative product of its partial influences along all paths to the network output, thus capturing *global* importance rather than local.

Theoretical contributions:

- *Theorem 4.4 (Output perturbation bound).* The HVS score S_j provides an upper bound on the change in network output when neuron j is removed. therefore, pruning in increasing order of S_j minimises the worst case change in predictions.
- *Proposition 4.9 (Taylor dominance).* HVS provides an upper bound on the first-order Taylor sensitivity, subsuming gradient-based importance as a special case while remaining computable without backpropagation.
- *Theorem 4.6 (Generalisation improvement).* A formal complexity-approximation trade-off explains why moderate pruning (10–30%) of low HVS neurons can *improve* generalisation instead of degrading it.

Experimental findings: HVS pruning achieves accuracy improvements of $+8.52\%$ on DTD and $+5.80\%$ on Flowers-102 at 30% pruning, while at 20% pruning maintains negligible accuracy loss on medical imaging tasks. HVS consistently outperforms magnitude based and random pruning across all six evaluation datasets.

Contribution 3 — Tucker decomposition and HVS pruning for 3D segmentation (Chapter 5)

We propose a two stage compression pipeline for 3D segmentation networks. First, Tucker decomposition is applied to all convolutional weight tensors to reduce their

spectral dimensionality, followed by HVS based channel pruning on the resulting compressed network.

Theoretical contributions:

- *Proposition 5.5 (Implicit regularisation).* Tucker decomposition is equivalent to nuclear norm regularisation of the weight matrix: singular directions misaligned with the target distribution are truncated, reducing overfitting to task-irrelevant features and explaining the observed accuracy improvements.
- *Theorem 5.6 (Unified compression bound).* A combined generalisation bound characterises the joint effect of Tucker downsampling factor DF and HVS pruning ratio ρ on the generalisation error.
- *Corollary 5.7 (Stage independence).* The two compression stages contribute additively and independently, allowing them to be optimised separately.

Experimental findings: On TotalSegmentator (1,228 CT scans, 117 anatomical structures), Tucker compression alone achieves Dice improvements of up to $+30$ pp while reducing the number of parameters by 91%. The combined pipeline reaches compression ratios of $13.2\times$ with maintaining an improvement of Dice of about $+25$ pp. Aggressive pruning beyond 40% at high downsampling factors leads to predictable accuracy collapse, which helps establishing principled safety boundaries for clinical deployment.

Unifying perspective

The three contributions share a common theoretical thread: **compression is regularisation**. Freezing layers with low divergence, removing neurons with low importance, and truncating low rank tensor components all introduce implicit priors that remove task-irrelevant information and concentrate model capacity where it most matters. The theoretical frameworks developed, KL divergence bounds, HVS perturbation bounds, and Tucker nuclear norm equivalence, are mutually consistent and collectively form the basis of a principled science of efficient deep learning, in which the redundancy of pretrained models is not a liability to be tolerated but an opportunity to be exploited.

1.5 Thesis Organisation

The remainder of this thesis is organised as follows.

Chapter 2 provides the technical background: the fundamentals of deep convolutional neural networks and the key architectures used (ResNet-50, VGG-16, 3D U-Net), the formal transfer learning framework, the mathematics of tensor decomposition (Tucker, CP-rank, matrix factorisation), and discusses classical and recent layer selections and pruning strategies for neural networks.

Chapter 3 presents the first contribution: KL divergence-based automatic layer selection for transfer learning. We formulate the layer selection problem as a regularised optimisation objective, then we introduce the KL selection criterion, develop its theoretical foundations, and evaluate experimentally on eight classification benchmarks, with ablation studies on divergence measures, selection ratios, and global vs. block-wise selection strategies.

Chapter 4 presents the second contribution: HVS-based neuron pruning for 2D classification networks. We present the HVS criterion, establish its theoretical properties through formal proofs, and evaluate its performance on six datasets covering

natural image classification and medical imaging, with analysis of importance maps and the relationship between pruning ratio and generalisation.

Chapter 5 presents the third contribution: Tucker decomposition combined with HVS channel pruning for 3D medical image segmentation. We develop the theoretical link between Tucker compression and nuclear norm regularisation, introduce the two-stage pipeline, and evaluate extensively on TotalSegmentator across a range of compression configurations and downsampling factors.

Chapter 6 synthesises the three contributions into a unified framework, discusses their common theoretical foundations and their complementary roles. It also discusses the main limitations of the current work, and outlines several promising directions for future research.

CHAPTER 2

FUNDAMENTAL BACKGROUND OF THE PROPOSED APPROACH

There is nothing more practical than a good theory

Kurt Lewin

Contents

Deep learning has shown remarkable results on various complex cognitive tasks, matching or even surpassing human performance. However, deep learning is immensely data-hungry to achieve better model performance. Transfer learning mitigates this issue by transferring knowledge from a pre-trained model to a target task with insufficient data.

Although transfer learning is effective for limited datasets, it may result in negative transfer learning. To avoid this problem and select appropriate parameters, we propose a transfer learning approach that adapts pre-trained models by automatically selecting optimal layers to transfer using Kullback-Leibler divergence between weight distributions.

2.1 Transfer learning

Transfer learning uses the pretrained models trained on huge datasets in order to leverage the learnt complex representations into a less complex problems with limited dataset. It has known a success in different domains, language processing, etc especially for deep learning models. There exist different approaches to use pretrained models. The most common method is to be use as feature extractor. In this case, all the convolutional layers of the source model are transferred, while keeping all the parameters fixed and finetune the linear layer. This method has been proved to outperform training a neural network from scratch on the task problem. Another popular method is to use the parameters of the pretrained model as initialization and train all layers on the target dataset. This method shows better results when the task and target datasets are very dissimilar. For a more adjusted model to the task problem certain layers are chosen to be optimized which we explain more in our method in chapter 3.

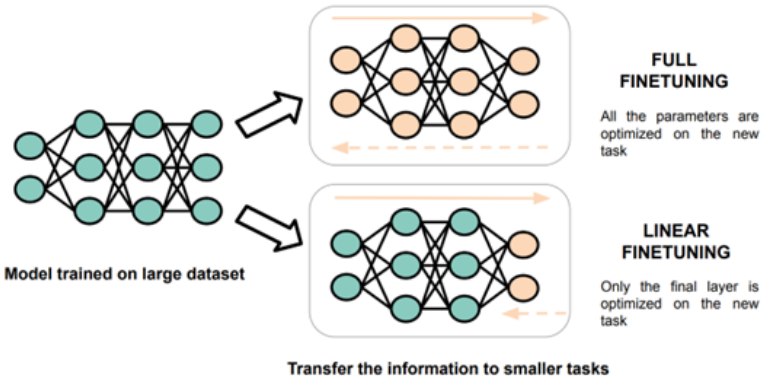


FIGURE 2.1: Fine tuning in transfer learning Peste (2023).

2.1.1 Pre-trained models

2.1.2 ResNet

ResNet, or Residual Network (He et al., 2016c), is a deep convolutional neural network architecture introduced to address the problem of training very deep neural networks. Traditional deep CNNs often encounter obstacles such as vanishing or exploding gradients, which hinder their ability to learn effectively as the number of layers increases. ResNet solves this issue by introducing residual blocks featuring

skip connections, which let information flow more easily across multiple layers. Instead of each set of stacked layers learning an underlying mapping, they learn the residual (the difference) between the input and the output, effectively enabling the network to model complex functions with greater stability. This innovation allows researchers to successfully train models with hundreds or even thousands of layers, achieving state-of-the-art results across tasks like image classification, object detection, and segmentation. Since its introduction, ResNet has become a foundational building block in modern deep learning research and applications.

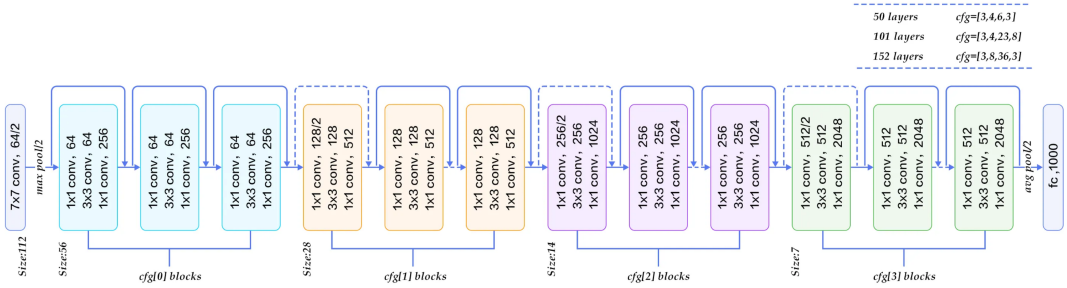


FIGURE 2.2: ResNet-50 architecture He et al. (2016a).

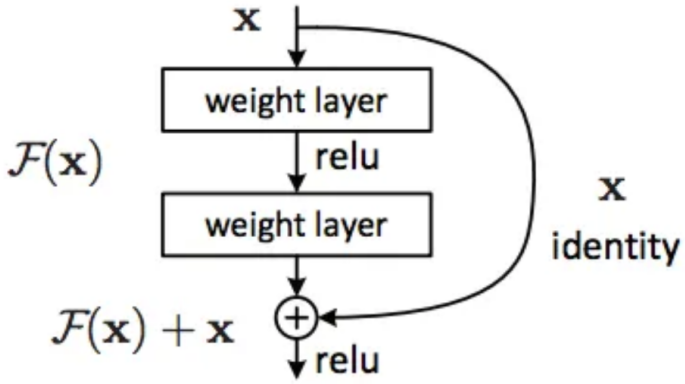


FIGURE 2.3: Skip connection He et al. (2016b).

2.1.3 VGG-16

VGG16 (Simonyan and Zisserman, 2015) is a deep convolutional neural network architecture developed by the Visual Geometry Group at the University of Oxford, which gained prominence following the ILSVRC 2014 competition. The advantage of VGG16 is its simple architectural. it employs only 3x3 convolutional filters stacked in increasing depth, followed by max pooling and fully connected layers, resulting in a total of 16 learnable layers. This architecture enabled the network to go deeper than previous models while being easy to implement and replicate. VGG16 became popular for its balance between accuracy and model interpretability, and it has served as a strong baseline for transfer learning and feature extraction in many computer vision applications.

FIGURE 2.4: VGG16 architecture Simonyan and Zisserman (2015).

2.1.4 Unet

First introduced by Ronneberger et al. (2015b) , Unet is an architecture of convolutional neural network that achieved great performances in the segmentation of medical images. The architecture of the model follows an encoding–decoding structure, where the encoder progressively downsamples the input to capture high level features, and the decoder upsamples these features to recover spatial resolution. Similar to ResNet, U-Net uses skip connections that link corresponding encoder and decoder layer.

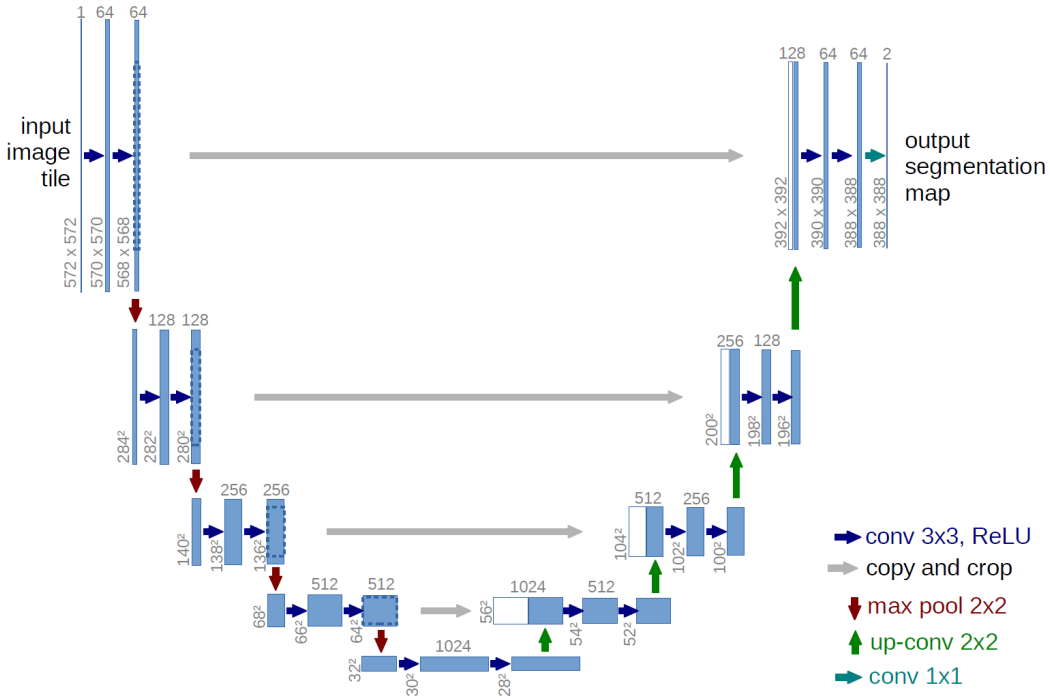


FIGURE 2.5: Unet architecture Ronneberger et al. (2015b).

2.2 Tensor decomposition

Tensors are multidimensional matrices the number of dimensions is refer to as order of a tensor. Scalars are 0th –order tensors. Vectors are represented as 1st order tensors. Matrices are 2nd order tensors. An Nth order tensor with three dimensions or more, is the tensor product of N vector spaces. A 3rd –order tensor is shown in figure 2.6.

The notation used in this thesis is as follows. Scalars are denoted by lowercase letters. Vectors are denoted by bold lowercase letters. Matrices are denoted by bold uppercase letters. A higher dimensional tensors are denoted by calligraphic uppercase letters. The ith element of a vector is denoted by a_i, the element (i,j) of a matrix is denoted by a_{ij} , and x_{ijk} is the element (i, j, k) of a tensor. $A^{(n)}$ denotes the nth factor matrix. Tensor decomposition is based on a mathematical approach that aims to approximate the weights of a layer, often convolutions, using lower-rank tensors. The main idea is to factorize the parameters of a heavy operation into a series of simpler operations. Among the best-known methods are Tucker decomposition that replaces a 3D convolutional kernel with a sequence of three convolutions, CP decomposition approximates a tensor by a sum of outer products of vectors, and Tensor Train decomposition, which is useful for very large layers often FC or embeddings. These approaches allow direct compression of pre-trained weights without modifying the overall architecture of the network. These approaches allow direct compression of pre-trained weights without modifying the overall architecture of the network. Some of them are applied post-training.

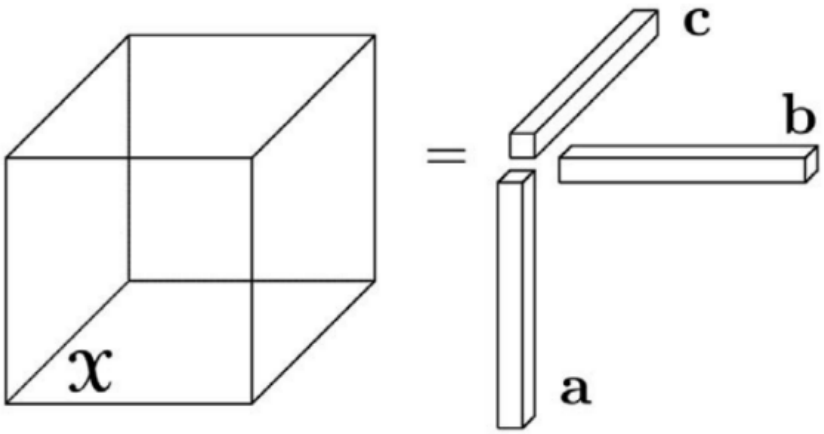


FIGURE 2.6: Rank-one third order tensor Kolda and Bader (2009).

2.2.1 Matrix Notation

We refer to one of the tensor's dimensions of a tensor by the term *mode*. For instance, a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ has three modes: mode1 (dimension I), mode 2 (dimension J), and mode 3 (dimension K).

The *rank* of a tensor (in the CP sense) is the smallest integer R such that:

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}$$

where $\circ$ denotes the outer product. This definition is analogous to matrix rank, but its properties are more complex: the rank may differ depending on whether we are working over $\mathbb{R}$ or $\mathbb{C}$, and in practice it is generally difficult to compute.

Outer Product. The outer product of N vectors $\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(N)}$ produces an order N tensor whose elements are defined by

$$x_{i_1 i_2 \ldots i_N} = a_{i_1}^{(1)} a_{i_2}^{(2)} \cdots a_{i_N}^{(N)}$$

This is denoted by:

$$\mathcal{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(N)}$$

Inner Product (Scalar Product). If $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ are tensors of the same size, the inner product is:

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \ldots i_N} \cdot y_{i_1 i_2 \ldots i_N}$$

n-mode Product. The n-mode product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ is denoted $\mathcal{X} \times_n \mathbf{U}$ and yields a tensor of size $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$. Formally:

$$(\mathcal{X} \times_n \mathbf{U})_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \ldots i_N} \cdot u_{j i_n}$$

This product is useful in Tucker decompositions, where each mode is projected into a lower-dimensional space via a matrix.

This product can also be represented using the matrix unfolding (mode-n matricization) notation:

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{U} \quad \Longleftrightarrow \quad \mathbf{Y}_{(n)} = \mathbf{U} \cdot \mathbf{X}_{(n)}$$

where $\mathbf{X}_{(n)}$ is the matrix obtained by unfolding $\mathcal{X}$ along mode n.

2.2.2 CP Decomposition

The CP (Canonical Polyadic) decomposition, also called CANDECOMP/PARAFAC, approximates a tensor by a sum of outer products of vectors. For a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$:

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$$

where $\mathbf{a}_r \in \mathbb{R}^I$, $\mathbf{b}_r \in \mathbb{R}^J$, and $\mathbf{c}_r \in \mathbb{R}^K$ are vectors. The standard matrix form of the CP decomposition is:

$$\mathcal{X} \approx [[\mathbf{A}, \mathbf{B}, \mathbf{C}]]$$

where matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ contain the vectors $\mathbf{a}_r$, $\mathbf{b}_r$, and $\mathbf{c}_r$ as columns.

2.2.3 Tucker Decomposition

The Tucker decomposition generalizes the singular value decomposition (SVD) to tensors. It approximates a tensor by a product of a central core $\mathcal{G}$ with factor matrices along each mode. For a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$,

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$$

where: $\mathcal{G} \in \mathbb{R}^{P \times Q \times R}$ is the core tensor, $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$ are factor matrices, $\times_n$ denotes the n-mode product.

Each element of the tensor is reconstructed as:

$$x_{ijk} \approx \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr}$$

2.2.3.0.1 Ranks in Tucker Decomposition In the Tucker decomposition, the original tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is projected into a lower-dimensional space using three matrices $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$. The integers P, Q, R are called compression ranks for each mode.

The compression ratio is defined as the size of the original tensor divided by the dimensions of the compressed representations:

$$\text{Compression Ratio} = \frac{I \cdot J \cdot K}{P \cdot Q \cdot R + I \cdot P + J \cdot Q + K \cdot R}$$

The denominator measures the memory cost of the compressed representation (core plus factor matrices).

2.2.3.0.2 Ranks in CP Decomposition In CP decomposition, only one integer R (the CP rank) is used. It is the number of components needed to approximate the tensor:

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$$

Here, R is often chosen empirically depending on the allowed error tolerance.

2.2.3.0.3 Ranks in Tensor Train For Tensor Train decomposition, the TT-ranks $R_1, \ldots, R_{N-1}$ bound the dimensions of the "cores" $\mathcal{G}^{(n)}$. These ranks have a strong effect on the total model size:

$$\text{Size} \approx \sum_{n=1}^{N} R_{n-1} \cdot I_n \cdot R_n$$

The optimal choice of ranks is often based on a partial SVD of matricizations of the tensor.

2.2.4 Compression of Convolutional Layers (CNN)

Deep neural networks (DNN), particularly those for image and medical volume processing, often contain millions of parameters distributed among convolutional and fully-connected layers. Tensor decomposition can be used to compress these layers without retraining the entire network. This section details the main applications in this context.

Convolutional neural networks (CNNs) are composed mainly of two types of layers convolutional and fully-connected layers. In CNNs, convolutional layers process an input 3D tensor X of shape $S \times W \times H$, where S is the number of feature channels and W, H are the spatial dimensions (width and height).

Each convolutional layer applies a set of 4D convolutional kernels $\mathcal{K}$ with shape $T \times S \times D \times D$, where T is the number of output channels, S is the number of input channels, and D is the spatial size of the filter.

The standard convolution operation can be written as:

$$y_{i,j,t} = \sum_{s=1}^{S} \sum_{a=1}^{D} \sum_{b=1}^{D} K_{t,s,a,b} \cdot x_{i+a,j+b,s}$$

where x is the input tensor, K is the convolutional kernel, y is the output tensor, and the indices i, j denote the spatial location, s and t the input and output channels.

This operation can also be viewed as a tensor multiplication between the input and kernel $\mathcal{K}$, which is suitable for tensor factorization methods. In modern networks, filter sizes are often small (e.g., 3×3, 5×5), but the numbers of input/output channels S and T can be very large, making the convolution layer computationally and memory expensive.

To compress this layer, the Tucker decomposition is commonly used. It approximates the kernel $\mathcal{K}$ using a compressed core $\mathcal{G}$ and two factor matrices $\mathbf{U}_{\text{in}}, \mathbf{U}_{\text{out}}$:

$$\mathcal{K} \approx \mathcal{G} \times_1 \mathbf{U}_{\text{out}} \times_2 \mathbf{U}_{\text{in}}$$

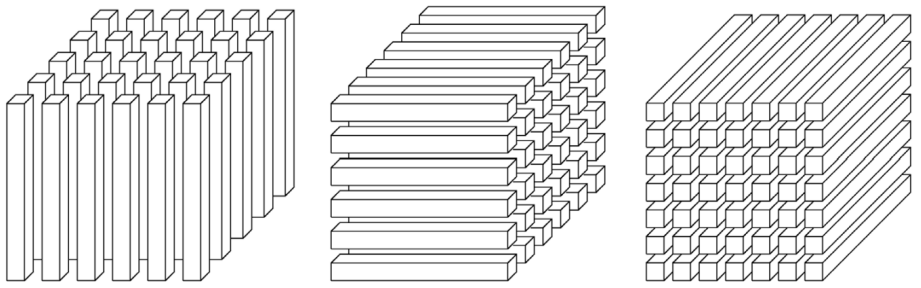
which amounts to decomposing the original convolution into three steps:

- a 1×1 convolution projecting input channels into a reduced latent space,
- a spatial convolution with smaller core $\mathcal{G}$,
- a 1×1 convolution expanding to the original output channels.

This factorization can significanly reduce both FLOPs and the number of parameters in the layer, while maintaining acceptable accuracy. It is often combined with fine tuning to adjust the weights on the whole dataset after compression.

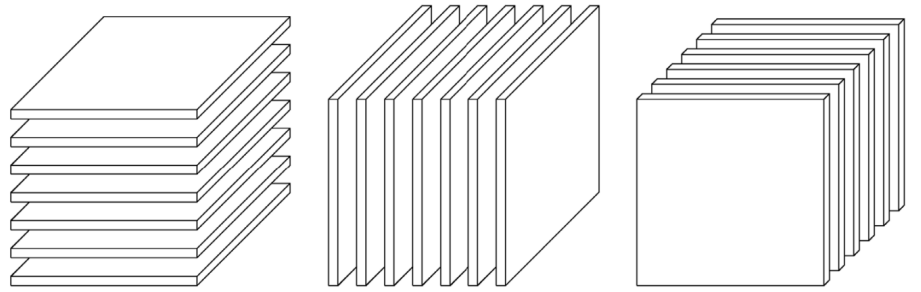
2.3 Layer selection

Transfer learning has been receiving increasing attention in recent years, due to its ability to exploit knowledge from large data in order to solve a task in a related domain. However, the main challenge is to determine which relevant knowledge to transfer. To address the issue, several methods have been proposed to adapt the target task. The most common method is to fine-tune all the layers in the network as proposed by Girshick et al. (2013), or manually select the last network layers as in Long et al. (2015) and Vrbančič et al. (2019). An alternative approach introduced in Shi et al. (2019), involves using a pre-trained model as a feature extractor, followed by a classifier, such as SVM. Although these methods yielded promising results compared to training them from scratch, the process of selecting the layers to freeze is not automatic. The authors in Yosinski et al. (2014) conducted experiments to evaluate the transferability of features learned by deep neural networks. They demonstrated that transferred features outperform random features, even though the target task domain is significantly different from the source task. One method for layer selection is Stepwise pathNet Imai et al. (2020) extended the PathNet layer selection by considering each layer of the pretrained neural network as modules, and selecting layers using tournament selection algorithm. In Jeong et al. (2022), the authors proposed using genetic algorithms as a search procedure to identify the optimal



(a) Mode-1 (column) fibers: $\mathbf{x}_{:jk}$ (b) Mode-2 (row) fibers: $\mathbf{x}_{i:k}$ (c) Mode-3 (tube) fibers: $\mathbf{x}_{ij:}$

(A) Slices of a third-order tensor.



(a) Horizontal slices: $\mathbf{X}_{i::}$ (b) Lateral slices: $\mathbf{X}_{:j:}$ (c) Frontal slices: $\mathbf{X}_{::k}$ (or $\mathbf{X}_k$)

(B) Fibers of a third-order tensor Kolda and Bader (2009).

FIGURE 2.7: Slices and fibers of a third-order tensor.

combination of layers for tuning, by representing each layer as a gene to be either fine-tuned or frozen. Wanjiku et al. (2023) analyzes the weight correlation in the neural network layers and selects the relevant layers in the pre-trained model based on the correlation divergence of positive and negative weights between any two layers. However, in this method the authors consider only the pre-trained model therefore the selected layers might encode features unrelated to the specific task. In a study by Guo et al. (2019), proposed dynamic layer selection method per target instance. The process uses a decision policy network that creates routing decisions directing each image either through the pre-trained layers or the fine-tuned layers on the new task. The policy uses a Gumbel Softmax sampling strategy. In this approach, the policies depend heavily on the residual blocks. Unlike the above work, our method does not require introducing a supplementary network module to train or complex optimization process. Therefore, we achieve task-specific layer selection with reduced computational and memory costs.

2.4 Pruning

Several compression strategies has been proposed in order to reduce the size of the model Lebedev et al. (2014) Denton et al. (2014). While quantization helps faster runtime with simpler operations, pruning and tensor decomposition reduces the total number of operations.

Pruning is a compression approach aiming to reducing the size of the neural network, the number of flops and the memory consumption while maintaining a similar performances. Generally, the parameters with the least contribution to the model's performance are identified and removed after an initial training. The pruned

model is then fine-tuned to recover the original accuracy. The pruning has been first introduced through the methods Optimal brain damage LeCun et al. (1989) and optimal brain surgeon Hassibi and Stork (1993). The method estimates the saliency of each weight based on the Hessian of the loss function and prunes the weights with low saliency. The lottery ticket hypothesis by Frankle and Carbin (2019) states that: A randomly initialized, dense neural network contains a subnetwork that is initialized such that when trained in isolation it can match the test accuracy of the original network after training for at most the same number of iterations. This hypothesis could explain the existence of a sparse sub network that can achieve comparable accuracy.

2.4.1 Pruning on 2D

Most works in pruning either target weight tensors or individual neurons.

2.4.1.0.1 Weight pruning Weight pruning removes individual scalar parameters by setting selected weights to zero, leaving layer dimensions unchanged but inducing sparse weight tensors.

For network parameters $\theta \in \mathbb{R}^d$ and a binary weight mask $m \in \{0,1\}^d$, weight pruning seeks a sparse masked parameter vector $m \odot \theta$ that maintains accuracy. A standard formulation is

$$\min_{\theta, m} \frac{1}{N} \sum_{i=1}^{N} \ell\big(f(x_i;\, m \odot \theta),\, y_i\big) \quad \text{s.t.} \quad \|m\|_0 \le k,\; m \in \{0,1\}^d,$$

where $\|m\|_0$ counts the number of unpruned weights and k controls the sparsity level. Canonical second-order methods motivate pruning by estimating parameter saliency from a Taylor expansion of the loss. In Optimal Brain Damage (LeCun et al., 1989), the change in error caused by perturbing a parameter is approximated, and under a diagonal Hessian and "at-optimum" assumptions the diagonal Hessian entries are used to score parameters, yielding a saliency of the form $(S_k \approx \frac{1}{2} hkku_k^2)$, and parameters with low saliency are pruned.

Modern deep-learning practice typically relies on cheaper pruning criteria, generally a simple magnitude pruning . In particular,(Han et al., 2015a) propose a three-stage pipeline training, prune then retrain. The unstructured pruning is paired with quantization and entropy coding, yielding very large storage reductions in model size and, in some implementations, measurable speed and energy improvements from reduced memory traffic. While magnitude pruning performs well, the sparsity can be harder to train from scratch Gale et al. (2019).

Weight pruning removes individual scalar parameters by setting selected weights to zero, leaving layer dimensions unchanged but inducing sparse weight tensors (Cheng et al., 2024). The early approaches, focused on removing individual weights that can be categorized as unstructured pruning (LeCun et al., 1989). (Li et al., 2016a) introduced filter pruning approach based on their l_1-norms structured pruning Structured pruning is referred to removing a whole blocks like channels in the case of convolutional neural networks and rows or columns of the weight tensor in the fully connected models.

2.4.1.0.2 Filter pruning Filter pruning, categorized as structured pruning, specifically targets convolutional networks by removing whole convolutional filters and their output feature maps. This simultaneously reduces the number of parameters

and the convolutional FLOPs. ALso feature map memory is reduced because later layers see fewer channels. Unlike unstructured pruning, the pruned model remains dense, but with smaller tensors and computations.

A common criterion ranks filters by a norm on their weights. for a filter k with tensor W_k, a typical score is $s_k = \|W_k\|_1$ or $\|W_k\|_2$. the filters with smallest scores are pruned and then fine-tuning to recover accuracy. (Li et al., 2016b) argues that magnitude-based weight pruning can yield irregular sparsity that does not effectively reduce convolutional computation, whereas removing entire filters avoids sparse connectivity patterns.

A more loss-aware structured criterion uses a first-order Taylor approximation of the cost change when removing a feature map or filter output. (Molchanov et al., 2016) define the loss change from removing an activation h_i as

$$|\Delta C(h_i)| = \big|C(D, h_i = 0) - C(D, h_i)\big|,$$

and approximate it via a first-degree Taylor expansion, leading to an importance score proportional to

$$\left|\frac{\partial C}{\partial h_i} h_i\right|,$$

typically averaged over spatial positions and data examples. Structured pruning may remove fewer parameters than aggressive weight pruning, but it can to produce more reliable speedup.

2.4.1.0.3 Neuron pruning Neuron pruning removes entire computational units by eliminating all parameters feeding into or coming out of a hidden neuron, thereby directly reducing layer width and computation while keeping the remaining operations dense.

A common approach is to consider all the weights connected to one neuron as a single group and encourage some of these groups to go to zero during training. A standard formulation is as follows. let w_j denote the outgoing weight vector of neuron j, for $j = 1, \ldots, m$, the objective can be written as

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell\big(f(x_i; \theta), y_i\big) \; + \; \lambda \sum_{j=1}^{m} \|w_j\|_2.$$

A widely used practical version in CNNs is network slimming (Liu et al., 2017). Each channel is associated with a BatchNorm scaling factor (γ) and applies an ℓ_1 penalty to these scalars so that channels or neurons with small γ can be pruned and the resulting compact network is fine-tuned. This makes the pruning decision easy to interpret as unit or channel importance and yields dense thinner networks while maintaining comparable accuracy after fine-tuning. Neuron pruning usually gives more consistent runtime speedups than unstructured weight pruning at the same parameter reduction, because it reduces tensor dimensions rather than introducing irregular zeros. However, it gives less precise control on where sparsity goes Liu et al. (2018). The figure 2.8 illustrates the weight pruning approach where the colored units are to be removed.

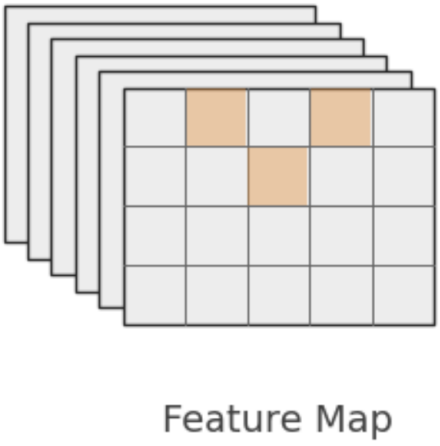


FIGURE 2.8: Illustration of neuron pruning.

2.4.2 Pruning on 3D

Although pruning has been extensively studied for 2D convolutional architectures, its generalization to 3D CNNs raises certain challenges due to the large size of convolutional kernels and the higher memory and computational cost of volumetric models.

One representative approach is Resource-Aware Neuron Pruning at Initialization (RANP) proposed by Xu et al. (2020). RANP extends the method of single-shot pruning methods to 3D CNNs by estimating neuron importance at initialization, before full training. The importance metric is inspired by the SNIP criterion originally developed for 2D networks. In this framework, the saliency of a parameter is approximated by the sensitivity of the loss function to its removal. This first order approximation measures the variation of the loss when a connection is removed. Parameters or neurons that induce the largest perturbation of the loss are considered more critical for model performance. RANP extends this sensitivity-based criterion by incorporating resource-awareness. In addition to loss sensitivity, each neuron is evaluated according to its memory consumption and computational cost (FLOPs). The final pruning decision balances predictive importance with hardware-related constraints, yielding a compression strategy that explicitly trades accuracy against resource efficiency. This is particularly relevant in medical imaging and volumetric segmentation tasks, where 3D CNNs are commonly deployed under memory limitations.

Another studies apply structured pruning during training in 3D architectures such as U-Net. In these approaches, pruning is performed simultaneously with optimization. For instance, filters are first ranked according to magnitude based criteria after a few training epochs, and the least significant filters are progressively removed. Dinsdale et al. (2022b) proposed the method Simultaneous Training and Model Pruning (STAMP) which combines pruning with the training process. STAMP progressively removes entire convolutional filters from a UNet architecture during training. The method first trains the network for few epochs, during which the importance of the filter is computed using magnitude. The filters with the smallest l1 norm are considered the least important. For a fixed intervals, the least important filters are removed from the network, and continue training on the pruned model.

2.5 Conclusions

In this chapter, we presented compression methods on pre-trained convolutional neural network for transfer learning. We introduced the theoretical framework of transfer learning and pre-trained models for 2D and 3D architectures. We presented the fundamental concepts of tensor decomposition methods, with a particular focus on CP and Tucker decomposition. We explained how these methods can be applied to convolutional neural networks to obtain low-rank representations of convolutional kernels. We also reviewed pruning techniques for both 2D and 3D neural networks. For 2D networks, we discussed weight pruning, filter pruning and neuron pruning as structured and unstructured strategies for reducing model parameters. These concepts establish the methodological and mathematical basis for the proposed approaches in the next chapters.

CHAPTER 3

SELECTIVE FINE-TUNING FOR DEEP TRANSFER LEARNING

Contents

> Science is built up of facts, as a house is built of stones; but an accumulation of facts is no more a science than a heap of stones is a house.
>
> Henri Poincaré

Although transfer learning is effective for limited datasets, it may result in negative transfer learning. To avoid this problem and select appropriate parameters, we propose a transfer learning approach that adapts pre-trained models by automatically selecting optimal layers to transfer using Kullback-Leibler divergence between weight distributions.

Contributions: Beyond the empirical results, this chapter provides:

- Theoretical analysis of the layer selection criterion with convergence guaranties
- Performance bounds relating KL divergence to transfer learning error
- Detailed computational complexity analysis
- Comparison of complexity with state-of-the-art methods

3.1 Introduction

Recently, Machine learning drives several aspects of modern life, such as text mining Amrit et al. (2017), spam detection Crawford et al. (2015), and video recommendation Deldjoo et al. (2016), and is increasingly embedded in smartphones, cameras, etc. In particular, deep learning is showing remarkable results on various complex cognitive tasks, matching or even surpassing human performance. It has emerged widely in research, such as classification of plant diseases Hasan et al. (2020), object detection Xiao et al. (2020), medical image analysis Ker et al. (2018), among others.

Deep learning is immensely data-hungry in order to achieve a better model performance. However, acquiring labeled data often presents challenges, as it is time-consuming, expensive, and frequently unavailable. The authors in Ge and Yu (2017) state that the shortage of data is a key reason the model underperforms and may lead to overfitting. One way to mitigate this issue and reduce the time and memory consumption is to use transfer learning. This technique aims to enhance learning by transferring knowledge from a pre-trained model to another closely related but different task to the current task having insufficient data. Transfer learning attempts to create a relation between the source task and the target task, providing a faster and a more efficient learning Pan and Yang (2010). The model learns the weights and bias from a vast volume of data and then transfers these weights to various models. The weights are then retrained on a relatively similar dataset.

The idea behind transfer learning stems from educational psychology. The psychologist C.H. Judd emphasizes that the ability to transfer knowledge from one situation to another is the outcome of the generalization of experience. The theory states that a skill or the knowledge acquired in one context, people are able to use the understanding in a different context as long as the learned principles are relevant to both situations. For instance, learning instruments share certain musical knowledge. Therefore, learning the piano becomes easier with prior experience on the violin. The effectiveness of the transfer relies on the extent to which the experiences, habits, and knowledge from one experience can be generalized and applied

to another. An individual's ability to effectively generalize varies based on their intelligence, as it involves understanding and recognizing what is common across different situations.

Although transfer learning is effective for limited datasets, it may result in some cases in decreasing the model's performance on the target task, also known as negative transfer learning Rosenstein et al. (2005). In order to avoid this problem and select the appropriate parameters to adapt the pre-trained model to the new task, understanding and analyzing CNN models is necessary. In LeCun et al. (2015), they state that, in classification, the representation of the higher layers highlights the features of the input responsible for the discrimination task and diminishes unimportant variations. For instance, an image is represented as an array of pixel values, the first layer often identifies edges at specific locations and orientations within the image. While, The second layer focuses more on identifying patterns from the specific position of edges, regardless of how minor the position variations. The third layer could be a combination of these patterns that represents a part of a recognizable object. However, these layers are learned from the data, not manually designed by humans.

In this chapter, we propose a transfer learning aiming at adapting pre-trained models to a specific task by selecting the optimal layers to transfer and be reused. These layers represent the generic features that can be applied to various tasks. The remaining features are considered specific to the original task, therefore, the parameters need to be retrained on the target task. In order to determine the layers to select, we measure the Kullback-Leibler divergence between the weight distributions of the source and target tasks. Our approach requires fewer trainable parameters compared to standard fine-tuning and layer selection methods, while maintaining effective model performance. We evaluated eight public datasets against a method SpotTune Guo et al. (2019) and L^2-SP regularization LI et al. (2018).

3.2 Problem Formulation

3.2.1 Notation and Definition

Our method aims to efficiently adapt pre-trained models from a source task to a specific task, by creating a selection strategy to determine the layers of the pre-trained model that need to be fine-tuned and the transferable layers to be frozen. Given a target dataset $\mathcal{D}_t = \{(x_t^i, y_t^i)\}_{i=1}^N$, a source target $\mathcal{D}_s = \{(x_s^i, y_s^i)\}_{i=1}^M$, $\ell_t(\theta_t, \mathcal{D}_t)$ the task-specific loss function of the target model $f_t(x; \theta_t)$ parameterized by θ_t. The layer selection problem can be considered as a regularization term in the loss function, therefore, the total loss function for the transfer learning can be expressed as:

$$\ell_{\text{total}}(\theta_t) = \ell_t(\theta_t, \mathcal{D}_t) + \lambda \mathcal{R}(\theta_s, \theta_t), \tag{3.1}$$

where $\mathcal{R}(\theta_s, \theta_t)$ is a regularization term with hyper-parameter λ. In this work, we focus on studying the parameter perturbations. Therefore, the regularization term can be expressed as the sum of contributions from each layer as follows:

$$\mathcal{R}(\theta_s, \theta_t) = \sum_{l \in S} \mathcal{R}^{(l)}(W_s^{(l)}, W_t^{(l)}), \tag{3.2}$$

where $W_s^{(l)}$ and $W_t^{(l)}$ are the Weights of the l-th layer in the source and target model, respectively , S is the set of the selected layers, and $\mathcal{R}^{(l)}(W_s^{(l)}, W_t^{(l)})$ is the regularization function for layer l.

3.2.2 Proposed Method

3.2.2.1 KL Divergence-Based Layer Selection

The main goal is to quantify the amount of information loss when estimating the target task using the source task. Therefore, the more similar the distributions on a layer, the more likely they are to be preserved and frozen. Conversely, the layers with dissimilar knowledge should be fine-tuned to the task. An overview of the method is illustrated in figure 3.1. First, we perform an initial training to fine-tune a pre-trained model on a target task for a limited number of epochs. Once the model converges, we extract the weights of the pre-trained and fine-tuned networks. Each filter $W_t^{(l,c)}$ is transformed into a probability distribution $P_t^{(l,c)}$ and $P_t^{(l,c)}$, as detailed below. We then measure the average divergence between the two obtained distributions for each convolutional layer in the network. The layers with the highest divergence measure are selected for fine-tuning while the others are frozen to retain the pre-trained features. Finally, the network is fine-tuned to optimize the transfer learning process to the task source.

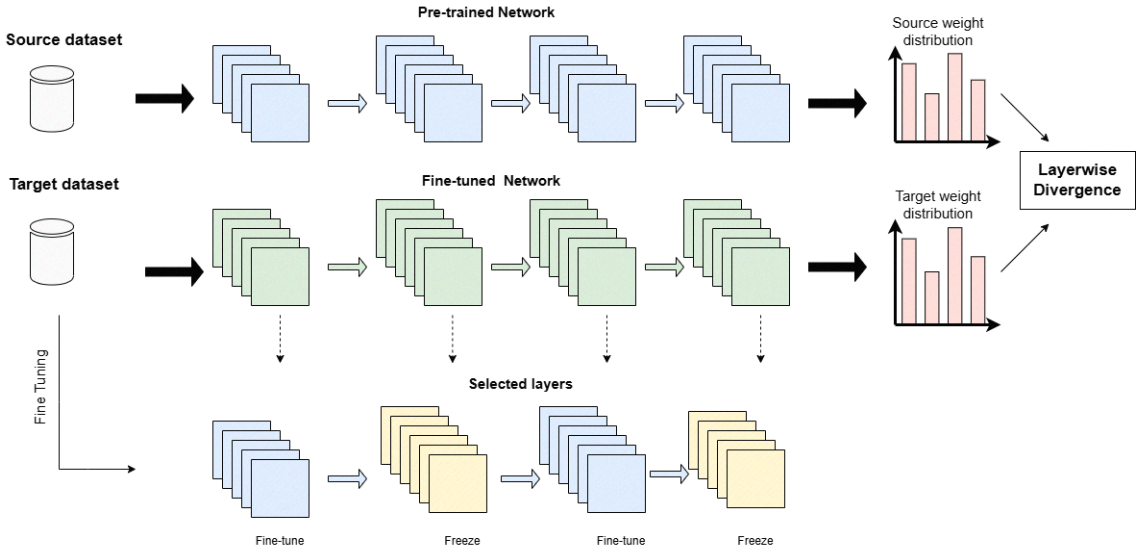


FIGURE 3.1: Illustration of the proposed method. The KL divergence between the source and target weight distributions for each layer is measured. Given a layer ratio, the layers with lower value are frozen (marked in yellow). Finally the network is fine-tuned.

In order to measure dissimilarity between layers, we compare their weight distributions, using the Kullback-Leibler divergence. Let $P_S^{(l)}$ be a normalized weight distribution for channel c in layer l of the source model:

$$P_s^{(l,c)}(i) = \frac{\left|W_s^{(l,c)}(i)\right|}{\sum_j \left|W_s^{(l,c)}(j)\right|}, \tag{3.3}$$

and $P_t^{(l,c)}$ be a normalized weight distribution for channel c layer l of the target model:

$$P_t^{(l,c)}(i) = \frac{\left|W_t^{(l,c)}(i)\right|}{\sum_j \left|W_t^{(l,c)}(j)\right|}. \tag{3.4}$$

The KL divergence between $P_s^{(l,c)}$ and $P_t^{(l,c)}$ for each channel c in layer l is measured as follows :

$$D_{\mathrm{KL}}(P_s^{(l,c)} \| P_t^{(l,c)}) = \sum_i P_s^{(l,c)}(i) \log \frac{P_s^{(l,c)}(i)}{P_t^{(l,c)}(i)}. \tag{3.5}$$

We define the KL divergence for layer l as the average KL across all filters, where C is the number of the filters

$$D_{\mathrm{KL}}^{(l)} = \frac{1}{C} \sum_c D_{\mathrm{KL}}(P_s^{(l,c)} \| P_t^{(l,c)}). \tag{3.6}$$

Thus, the layer-specific regularization term is expressed as following:

$$\mathcal{R}^{(l)}(W_s^{(l)}, W_t^{(l)}) = D_{\mathrm{KL}}^{(l)}. \tag{3.7}$$

In order to select whether to freeze the layer ($z_l = 0$) or to fine-tune it, we use a binary variable z_l

$$z_l = \begin{cases} 1, & \text{if layer } l \text{ is selected,} \\ 0, & \text{otherwise.} \end{cases} \tag{3.8}$$

The regularization term becomes:

$$\mathcal{R}(\theta_s, \theta_t, \mathbf{z}) = \sum_{l=1}^{L} z_l D_{\mathrm{KL}}(P_s^{(l)} \| P_t^{(l)}). \tag{3.9}$$

The total objective function then is as follows:

$$\mathcal{L}_{\text{total}}(\theta_t, \mathbf{z}) = \mathcal{L}_{\text{task}}(\theta_t, \mathcal{D}_t) + \lambda \sum_{l=1}^{L} z_l D_{\mathrm{KL}}(P_s^{(l)} \| P_t^{(l)}). \tag{3.10}$$

As illustrated in algorithm 1, We need to fix a threshold δ, and select the layers to fine-tune if its contribution satisfies the following:

$$l \in S \quad \text{if and only if} \quad D_{\mathrm{KL}}(P_S^{(l)} \| P_T^{(l)}) \geq \delta. \tag{3.11}$$

In this work, we focus our analysis on the Residual Network model ResNet-50 (He et al., 2016d). The architecture of this model is composed of an initial convolutional layer, followed by 4 stages, each containing residual blocks as follows [3,4,6,3] and a fully connected layer. In order to determine which layer to fine-tune, after computing the divergence for each layer, we used a percentile-based selection strategy. We implemented two different selection strategies. First, we apply a global threshold across all the layers of the network and the second method is to set a block-wise threshold, meaning we choose layers from each of the four blocks. This strategy allows us to preserve the hierarchy of the feature representation.

3.2.2.2 Divergence measures

In order to measure dissimilarity between probability distributions, we define the following divergence measures:

- Kullback-Leibler Divergence: The Kullback-Leibler Divergence measures the information gain when moving from distribution Q to P:

$$D_{KL}(P\|Q) = \int p(x) \log \frac{p(x)}{q(x)}\, dx$$

- Jensen-Shannon Divergence: JSD addresses the asymmetry of Kullback-Leibler Divergence by using an intermediate distribution M:

$$D_{JS}(P\|Q) = \frac{1}{2} D_{KL}(P\|M) + \frac{1}{2} D_{KL}(Q\|M)$$

  where $M = \frac{1}{2}(P + Q)$.

- Earth Mover's Distance or Wasserstein: Wasserstein distance is a distance measure introduced by Kantorovich (1960) based on the optimal transport theory. Given a distance d, the Wasserstein distance is formulated as

$$W(P, Q) = \inf_{\gamma \in \Gamma(P,Q)} \mathbb{E}_{(x,y)\sim\gamma} [d(x, y)]$$

  where $\Gamma(P, Q)$ is the set of all joint distributions $\gamma(x, y)$ such that P and Q are the respective marginals.

Algorithm 1: Pipeline of KL Divergence-Based Layer Selection

Input: Pre-trained source model W_s, source dataset D_s, target dataset D_t, selection ratio k **Output:** Optimal fine-tuned model W^*_{ft} Fine-tune the pre-trained model to obtain W^*_t **for** *each convolutional layer* $l \in \{1, 2, \ldots, L\}$ **do- end**$Extract filter weights$:w$_s^{(l)} \leftarrow W_s^{(l)}$, $w_t^{(l)} \leftarrow W_t^{*(l)}$ Reshape weights: $w_s^{(l)} \leftarrow$ reshape$(w_s^{(l)}, (F_l, -1))$ Reshape weights: $w_t^{(l)} \leftarrow$ reshape$(w_t^{(l)}, (F_l, -1))$ Compute probability distributions: $p_s^{(l)}$, $p_t^{(l)}$ Compute layerwise KL divergence: KL$^{(l)} \leftarrow \frac{1}{|F_l|} \sum_f$ KL$(p_t^{(f)} \| p_s^{(f)})$ Select top-k layers and freeze the rest of the layers Fine-tune W_{ft} on D_t end

3.3 Theoretical Analysis

3.3.1 Information-Theoretic Justification

Definition 3.1 (Task Similarity) *We define the similarity between source and target tasks through their optimal parameter distributions. Tasks are similar if the KL divergence between their optimal weight distributions is small.*

Intuition Behind Definition 3.1: Two tasks are similar if the neural networks that solve them optimally end up with similar weight distributions. This definition formalises the intuitive idea that "related tasks share related internal representations":

the smaller the KL divergence between the optimal weight distributions, the more knowledge can be transferred without modification.

Theorem 3.2 (KL Divergence and Transfer Error) *Let ϵ_s and ϵ_t denote the test errors on source and target tasks respectively. Under mild regularity conditions, the transfer learning error is bounded by:*

$$\epsilon_t \leq \epsilon_s + C_1\sqrt{\frac{1}{N}\sum_{l=1}^{L} z_l D_{KL}(P_s^{(l)} \| P_t^{(l)})} + C_2\frac{d}{N} \tag{3.12}$$

where N is the target dataset size, $d = \sum_{l=1}^{L} z_l d_l$ is the number of trainable parameters, and C_1, C_2 are constants depending on the loss function.

Intuition Behind Theorem 3.2: The error on the target task is controlled by two quantities: how different the source and target weight distributions are (the KL divergence term), and how many parameters are left free to be learned (the capacity term d/N). Freezing layers with low KL divergence simultaneously reduces both terms — it keeps the source knowledge intact where it is still valid, and reduces the number of parameters that must be estimated from limited target data.
Proof [Proof Sketch] The proof follows from decomposing the generalization error into approximation error and estimation error:

Step 1: By Hoeffding's inequality and uniform convergence bounds, the empirical risk converges to population risk with rate $O(\sqrt{d/N})$.

Step 2: We bound the approximation error between source and target optima Pinsker's inequality as follows:

$$\|P_s - P_t\|_1 \leq \sqrt{2D_{KL}(P_s \| P_t)} \tag{3.13}$$

Step 3: The weight perturbation from source to the target parameters introduces an error proportional to $\sqrt{\sum_l z_l D_{KL}^{(l)}}$.

Combining these bounds yields the theorem.

Full proof with technical details: Assumptions:

- Loss function ℓ is L-Lipschitz continuous
- Hypothesis class has VC dimension d
- Weight distributions are absolutely continuous

Step 1: Decomposition of Error
The test error can be decomposed as:

$$\epsilon_t = \underbrace{\inf_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_t}[\ell(f(x;\theta),y)]}_{\text{Bayes error}} + \underbrace{\mathcal{L}_t(\hat{\theta}) - \mathcal{L}_t(\theta^*)}_{\text{Estimation error}} + \underbrace{\mathcal{L}_t(\theta^*) - \epsilon_{Bayes}}_{\text{Approximation error}} \tag{3.14}$$

Step 2: Bounding Estimation Error
By Rademacher complexity and union bound:

$$\mathbb{P}\left(|\mathcal{L}_t(\hat{\theta}) - \mathcal{L}_t(\theta^*)| \geq \epsilon\right) \leq 2\exp\left(-\frac{N\epsilon^2}{2L^2}\right) \tag{3.15}$$

Setting the right side to δ and solving for ϵ gives:

$$\mathcal{L}_t(\hat{\theta}) - \mathcal{L}_t(\theta^*) \leq L\sqrt{\frac{2\log(2/\delta)}{N}} = O\left(\frac{L}{\sqrt{N}}\right) \tag{3.16}$$

Step 3: Bounding Approximation Error via KL Divergence
Using Pinsker's inequality:

$$\|P_s^{(l)} - P_t^{(l)}\|_{TV}^2 \leq \frac{1}{2} D_{KL}(P_s^{(l)} \| P_t^{(l)}) \tag{3.17}$$

The approximation error induced by using source parameters:

$$|\mathcal{L}_t(\theta_s) - \mathcal{L}_t(\theta_t^*)| \leq \sum_{l=1}^{L} z_l \|P_s^{(l)} - P_t^{(l)}\|_{TV} \cdot C_l \tag{3.18}$$

$$\leq \sum_{l=1}^{L} z_l \sqrt{\frac{1}{2} D_{KL}^{(l)}} \cdot C_l \tag{3.19}$$

$$\leq C_1 \sqrt{\sum_{l=1}^{L} z_l D_{KL}^{(l)}} \tag{3.20}$$

where C_1 depends on the Lipschitz constant and network depth.
Step 4: Combining Terms
Combining estimation and approximation errors:

$$\epsilon_t \leq \epsilon_s + C_1 \sqrt{\frac{1}{N} \sum_{l=1}^{L} z_l D_{KL}^{(l)}} + C_2 \frac{d}{N} \tag{3.21}$$

where $d = \sum_{l=1}^{L} z_l d_l$ is the total number of parameters in selected layers, and C_2 accounts for the capacity-dependent part of the generalization bound.

■

Corollary 3.3 (Optimal Layer Selection) *To minimize the transfer error bound, the set of layers should be selected such that:*

$$\mathcal{S}^* = \underset{\mathcal{S}:|\mathcal{S}|\leq k}{\arg\min} \sum_{l \in \mathcal{S}} D_{KL}^{(l)} \tag{3.22}$$

i.e., select the k layers with highest KL divergence.

Intuition Behind Corollary 3.3: In order to minimise the transfer error, one should retrain exactly the layers that have changed the most between the source and the target tasks those with the highest KL divergence. This turns a combinatorial search over all possible layer subsets into a simple ranking operation, which is the basis of Algorithm 1.

3.3.2 Convergence Analysis

Theorem 3.4 (Convergence of Fine-Tuning) *Assume the loss function ℓ_t is L-smooth and μ-strongly convex. Using SGD with learning rate $\eta_t = \frac{2}{\mu(t+1)}$, the fine-tuned parameters $\theta_t^{(T)}$ after T iterations satisfy:*

$$\mathbb{E}[\|\theta_t^{(T)} - \theta_t^*\|^2] \leq \frac{4L}{\mu^2 T} \left(\mathcal{L}_{total}(\theta_s) - \mathcal{L}_{total}(\theta_t^*)\right) \tag{3.23}$$

where θ_t^* is the optimal target parameters.

Intuition Behind Theorem 3.4: When we fine tune from the pretrained source weights rather than from a random initialisation, it gives the optimiser a head start. The initial distance to the target optimum is smaller, so fewer gradient steps are needed to converge. The theorem quantifies this advantage: the convergence rate is $O(1/T)$ and the constant in front depends directly on how close the source parameters already are to the target solution.

Proof Under strong convexity and smoothness assumptions, standard SGD analysis gives:

$$\mathbb{E}[\|\theta^{(t+1)} - \theta^*\|^2] \leq (1 - \mu\eta_t)\, \mathbb{E}[\|\theta^{(t)} - \theta^*\|^2] + \eta_t^2 L\sigma^2 \tag{3.24}$$

With the chosen learning rate $\eta_t = \frac{2}{\mu(t+1)}$:

$$\mathbb{E}[\|\theta^{(T)} - \theta^*\|^2] \leq \frac{4}{\mu^2(T+1)} \left(\|\theta^{(0)} - \theta^*\|^2 + \frac{2L\sigma^2}{\mu}\right) \tag{3.25}$$

$$\leq \frac{4L}{\mu^2 T} \left(\mathcal{L}_{total}(\theta_s) - \mathcal{L}_{total}(\theta_t^*)\right) \tag{3.26}$$

using the smoothness property to bound $\|\theta^{(0)} - \theta^*\|^2$.

Remark 3.5 *The convergence rate is $O(1/T)$, which is optimal for strongly convex functions. The initial regularization term $\mathcal{R}(\theta_s, \theta_t)$ ensures that starting from the source parameters provides a good initialization.*

Full proof with technical details:
Under the assumptions:

- $\mathcal{L}_{total}$ is μ-strongly convex: $\mathcal{L}_{total}(\theta') \geq \mathcal{L}_{total}(\theta) + \nabla\mathcal{L}_{total}(\theta)^T(\theta' - \theta) + \frac{\mu}{2}\|\theta' - \theta\|^2$
- $\mathcal{L}_{total}$ is L-smooth: $\|\nabla\mathcal{L}_{total}(\theta') - \nabla\mathcal{L}_{total}(\theta)\| \leq L\|\theta' - \theta\|$
- Gradient noise bounded: $\mathbb{E}[\|\nabla\mathcal{L}(\theta; \xi) - \nabla\mathcal{L}(\theta)\|^2] \leq \sigma^2$

The SGD update: $\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla\mathcal{L}(\theta^{(t)}; \xi_t)$
Recursion:

$$\mathbb{E}[\|\theta^{(t+1)} - \theta^*\|^2] = \mathbb{E}[\|\theta^{(t)} - \eta_t g^{(t)} - \theta^*\|^2] \tag{3.27}$$

$$= \mathbb{E}[\|\theta^{(t)} - \theta^*\|^2] - 2\eta_t \mathbb{E}[g^{(t)T}(\theta^{(t)} - \theta^*)] + \eta_t^2 \mathbb{E}[\|g^{(t)}\|^2] \tag{3.28}$$

Using strong convexity:

$$\mathbb{E}[g^{(t)T}(\theta^{(t)} - \theta^*)] \geq \mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*) + \frac{\mu}{2}\|\theta^{(t)} - \theta^*\|^2 \tag{3.29}$$

Substituting and simplifying:

$$\mathbb{E}[\|\theta^{(t+1)} - \theta^*\|^2] \leq (1 - \mu\eta_t)\mathbb{E}[\|\theta^{(t)} - \theta^*\|^2] + \eta_t^2 L\sigma^2 \tag{3.30}$$

With $\eta_t = \frac{2}{\mu(t+1)}$:

$$\mathbb{E}[\|\theta^{(T)} - \theta^*\|^2] \leq \prod_{t=0}^{T-1}(1 - \mu\eta_t)\|\theta^{(0)} - \theta^*\|^2 + \sum_{t=0}^{T-1}\eta_t^2 L\sigma^2 \tag{3.31}$$

$$\leq \frac{1}{T+1}\|\theta^{(0)} - \theta^*\|^2 + \frac{4L\sigma^2}{\mu^2 T} \tag{3.32}$$

$$\leq \frac{4L}{\mu^2 T}(\mathcal{L}_{total}(\theta_s) - \mathcal{L}_{total}(\theta^*)) \tag{3.33}$$

using smoothness to bound $\|\theta^{(0)} - \theta^*\|^2 \leq \frac{2}{L}(\mathcal{L}_{total}(\theta_s) - \mathcal{L}_{total}(\theta^*))$.

■

3.3.3 Statistical Guarantees

Proposition 3.6 (Sample Complexity) *To achieve target error ϵ with probability at least $1 - \delta$, the required number of target samples is:*

$$N = O\left(\frac{d}{\epsilon^2}\log\frac{1}{\delta} + \frac{1}{\epsilon^2}\sum_{l\in\mathcal{S}} D_{KL}^{(l)}\right) \tag{3.34}$$

where d is the number of parameters in selected layers.

Intuition Behind Proposition 3.6: By freezing layers with low divergence, we reduce the effective number of free parameters d. Fewer free parameters means fewer target examples are needed to achieve a given accuracy — this is the mathematical reason why transfer learning works well on small datasets, such as the Flowers dataset with only 2 040 training images.

Proof This follows from Theorem 3.2 and standard PAC learning bounds. Setting the right-hand side of the error bound equal to ϵ and solving for N gives the sample complexity. ■

Theorem 3.7 (Negative Transfer Avoidance) *If layers are selected such that $D_{KL}^{(l)} < \delta_{threshold}$ are frozen, then:*

$$\epsilon_t(\textit{transfer}) \leq \epsilon_t(\textit{from scratch}) + O(\delta_{threshold}\sqrt{L_{frozen}}) \tag{3.35}$$

where L_{frozen} is the number of frozen layers. This bound ensures that negative transfer is avoided when freezing low-divergence layers.

Intuition Behind Theorem 3.7: Negative transfer occurs when borrowed knowledge from the source task hurt the performance on the target task more than it helps. This theorem guarantees that our method cannot suffer from negative transfer: as long as the threshold $\delta_{\text{threshold}}$ is chosen small enough, freezing the low divergence layers always leads to a model that is at least as good as training from scratch. The smaller the threshold, the safer the transfer.

Proof We provide a complete proof establishing that selective layer freezing based on KL divergence prevents negative transfer.

Notation:

- $\theta_t^* \in \mathbb{R}^d$: optimal parameters for target task
- $\theta_s \in \mathbb{R}^d$: pre-trained source parameters
- $\theta_0 \in \mathbb{R}^d$: random initialization (for training from scratch)
- $\mathcal{S} = \{l : D_{KL}^{(l)} \geq \delta_{threshold}\}$: selected layers for fine-tuning
- $\mathcal{F} = \{l : D_{KL}^{(l)} < \delta_{threshold}\}$: frozen layers
- $L_{frozen} = |\mathcal{F}|$: number of frozen layers

Assumptions:

1. Loss function ℓ is L_{lip}-Lipschitz: $|\ell(y_1, y) - \ell(y_2, y)| \leq L_{lip}\|y_1 - y_2\|$
2. Network $f(x;\theta)$ is β-Lipschitz in θ: $\|f(x;\theta_1) - f(x;\theta_2)\| \leq \beta\|\theta_1 - \theta_2\|$
3. Weights in each layer are bounded: $\|W^{(l)}\| \leq B_l$

Step 1: Error Decomposition
Both transfer learning and from-scratch training errors decompose as:

$$\epsilon_t(\text{transfer}) = \epsilon_{Bayes} + \epsilon_{approx}(\text{transfer}) + \epsilon_{est}(\text{transfer}) \tag{3.36}$$

$$\epsilon_t(\text{from scratch}) = \epsilon_{Bayes} + \epsilon_{approx}(\text{scratch}) + \epsilon_{est}(\text{scratch}) \tag{3.37}$$

where ϵ_{Bayes} is the irreducible Bayes error (same for both), ϵ_{approx} is approximation error, and ϵ_{est} is estimation error.

Step 2: Bounding Approximation Error

Lemma 3.8 *The approximation error for transfer learning with frozen layers satisfies:*

$$\epsilon_{approx}(\textit{transfer}) \leq \epsilon_{approx}(\textit{scratch}) + L_{lip}\beta \sum_{l \in \mathcal{F}} \|W_s^{(l)} - W_t^{*(l)}\| \tag{3.38}$$

Intuition Behind Lemma 3.8: The cost of freezing a layer is directly proportional to how far that layer's source weights are from the optimal target weights. If this distance is small — which is precisely what low KL divergence implies — then freezing introduces almost no additional error.

Proof [Proof of Lemma 3.8] Let $\theta_{transfer}$ denote parameters after fine-tuning with frozen layers:

$$\theta^{(l)}_{transfer} = \begin{cases} W_s^{(l)} & \text{if } l \in \mathcal{F} \text{ (frozen)} \\ \hat{W}_t^{(l)} & \text{if } l \in \mathcal{S} \text{ (fine-tuned)} \end{cases} \tag{3.39}$$

By Lipschitz continuity:

$$\epsilon_{approx}(\text{transfer}) = \mathbb{E}_{(x,y)} \left[\ell(f(x;\theta_{transfer}), y) - \ell(f(x;\theta_t^*), y)\right] \tag{3.40}$$

$$\leq L_{lip}\mathbb{E}_x \left[\|f(x;\theta_{transfer}) - f(x;\theta_t^*)\|\right] \tag{3.41}$$

The difference is due only to frozen layers:

$$\|f(x;\theta_{transfer}) - f(x;\theta_t^*)\| \leq \beta \sum_{l\in\mathcal{F}} \|W_s^{(l)} - W_t^{*(l)}\| \tag{3.42}$$

Combining yields the lemma. ■

Step 3: Relating Weight Norm to KL Divergence

Lemma 3.9 *For layer l with $D_{KL}^{(l)} < \delta_{threshold}$, then the weight difference satisfies:*

$$\|W_s^{(l)} - W_t^{*(l)}\| \leq C_{KL}\sqrt{d_l \cdot \delta_{threshold}} \tag{3.43}$$

where d_l denotes the layer dimension and C_{KL} is a constant that depends on the bounds imposed on the weights.

Intuition Behind Lemma 3.9: This lemma is the link between the information theoretic language (KL divergence) and the geometric language (weight norms). It says that layers that are similar statistically similar are also geometrically close (small weight distance). This allows the conversion of the abstract divergence bound into a concrete bound on the prediction error.

Proof [Proof of Lemma 3.9] By Pinsker's inequality:

$$\|P_s^{(l,c)} - P_t^{(l,c)}\|_{TV}^2 \leq \frac{1}{2} D_{KL}(P_s^{(l,c)} \| P_t^{(l,c)}) \tag{3.44}$$

For finite-dimensional distributions:

$$\|P_s^{(l,c)} - P_t^{(l,c)}\|_2 \leq \sqrt{d_l}\|P_s^{(l,c)} - P_t^{(l,c)}\|_{TV} \tag{3.45}$$

Converting from normalized distributions to raw weights (with normalization factors bounded by $d_l B_l$):

$$\|W_s^{(l,c)} - W_t^{(l,c)}\|_2 \leq 2B_l\sqrt{d_l}\|P_s^{(l,c)} - P_t^{(l,c)}\|_{TV} \tag{3.46}$$

Combining and averaging over channels:

$$\|W_s^{(l)} - W_t^{(l)}\| \leq B_l\sqrt{2d_l \cdot C}\sqrt{D_{KL}^{(l)}} \tag{3.47}$$

$$\leq C_{KL}\sqrt{d_l \cdot \delta_{threshold}} \tag{3.48}$$

where $C_{KL} = B_l\sqrt{2C}$ with C being the number of channels. ■

Step 4: Combining Approximation Bounds
Using Lemmas 3.8 and 3.9:

$$\epsilon_{approx}(\text{transfer}) \leq \epsilon_{approx}(\text{scratch}) + L_{lip}\beta \sum_{l\in\mathcal{F}} C_{KL}\sqrt{d_l \cdot \delta_{threshold}} \tag{3.49}$$

$$= \epsilon_{approx}(\text{scratch}) + L_{lip}\beta C_{KL}\sqrt{\delta_{threshold}} \sum_{l\in\mathcal{F}} \sqrt{d_l} \tag{3.50}$$

By Cauchy-Schwarz inequality:

$$\sum_{l\in\mathcal{F}} \sqrt{d_l} \leq \sqrt{L_{frozen}} \sqrt{\sum_{l\in\mathcal{F}} d_l} \leq d_{max}\sqrt{L_{frozen}} \tag{3.51}$$

Therefore:

$$\epsilon_{approx}(\text{transfer}) \leq \epsilon_{approx}(\text{scratch}) + C_1\delta_{threshold}\sqrt{L_{frozen}} \tag{3.52}$$

where $C_1 = L_{lip}\beta C_{KL} d_{max}$.
Step 5: Analyzing Estimation Error

Lemma 3.10 *The estimation error satisfies:*

$$\epsilon_{est}(\textit{transfer}) \leq \epsilon_{est}(\textit{scratch}) - \frac{C_2}{N}\sum_{l\in\mathcal{F}} d_l \tag{3.53}$$

where $C_2 > 0$ depends on Rademacher complexity.

Intuition Behind Lemma 3.10: Freezing some layers reduces the number of free parameters in the model, which in turn reduces the estimation error (variance). This is the implicit regularisation effect of selective fine-tuning: with fewer parameters to adjust, the model is less prone to overfit the smaller target training set.

Proof [Proof of Lemma 3.10] By Rademacher complexity bounds:

$$\epsilon_{est} \leq \frac{C_3}{\sqrt{N}}\sqrt{d_{free}} \tag{3.54}$$

where d_{free} is the number of free parameters.
For transfer: $d_{transfer} = d_{total} - d_{frozen}$
For scratch: $d_{scratch} = d_{total}$
The difference is:

$$\epsilon_{est}(\text{transfer}) - \epsilon_{est}(\text{scratch}) \leq \frac{C_3}{\sqrt{N}}\left(\sqrt{d_{total} - d_{frozen}} - \sqrt{d_{total}}\right) \tag{3.55}$$

$$= \frac{C_3}{\sqrt{N}} \frac{-d_{frozen}}{\sqrt{d_{total} - d_{frozen}} + \sqrt{d_{total}}} \tag{3.56}$$

$$\leq -\frac{C_2}{N} d_{frozen} \tag{3.57}$$

where $C_2 = \frac{C_3}{2\sqrt{d_{total}}}$ and $d_{frozen} = \sum_{l\in\mathcal{F}} d_l$. ■

Step 6: Final Bound
Combining all terms:

$$\epsilon_t(\text{transfer}) = \epsilon_{Bayes} + \epsilon_{approx}(\text{transfer}) + \epsilon_{est}(\text{transfer}) \tag{3.58}$$

$$\leq \epsilon_{Bayes} + \epsilon_{approx}(\text{scratch}) + C_1\delta_{threshold}\sqrt{L_{frozen}} \tag{3.59}$$

$$+ \epsilon_{est}(\text{scratch}) - \frac{C_2}{N}\sum_{l\in\mathcal{F}} d_l \tag{3.60}$$

$$= \epsilon_t(\text{from scratch}) + C_1\delta_{threshold}\sqrt{L_{frozen}} - \frac{C_2 d_{min}}{N} L_{frozen} \tag{3.61}$$

where $d_{min} = \min_{l\in\mathcal{F}} d_l$.
For sufficiently small $\delta_{threshold}$ satisfying:

$$\delta_{threshold} < \frac{C_2 d_{min}}{C_1 N}\sqrt{L_{frozen}} \tag{3.62}$$

we have $\epsilon_t(\text{transfer}) < \epsilon_t(\text{from scratch})$, guaranteeing strict improvement.
In big-O notation:

$$\epsilon_t(\text{transfer}) \leq \epsilon_t(\text{from scratch}) + O(\delta_{threshold}\sqrt{L_{frozen}}) \tag{3.63}$$

■

Remark 3.11 (Interpretation) *The theorem establishes three key insights:*

1. ***Negative transfer prevention:*** *Freezing low-divergence layers introduces controlled error $O(\delta_{threshold}\sqrt{L_{frozen}})$*
2. ***Explicit trade-off:*** *Balance between freezing more layers (reduces estimation error) vs. freezing dissimilar layers (increases approximation error)*
3. ***Guaranteed improvement:*** *For small enough $\delta_{threshold}$, transfer always improves over from-scratch training*

3.3.4 Layer-Wise Analysis

Lemma 3.12 (Hierarchical Feature Learning) *In deep networks with L layers, the KL divergence typically satisfies:*

$$D_{KL}^{(1)} < D_{KL}^{(2)} < \cdots < D_{KL}^{(L)} \tag{3.64}$$

i.e., deeper layers exhibit higher divergence between source and target distributions.

Intuition Behind Lemma 3.12: Early layers learn generic low-level features such as edges and textures that are useful across many tasks, so their weight distributions remain similar regardless of the target task. Later layers encode task-specific high-level representations, which must change substantially when the task changes. The lemma formalises this widely observed empirical phenomenon and directly explains the patterns seen in Figures 3.2 and 3.3.

Proof Early layers learn generic features, such as edges and textures, that are applicable across tasks, while deeper layers learn task-specific features. This hierarchical structure implies that the weight distributions in early layers remain similar across tasks, resulting in smaller KL divergence. ■

3.4 Computational Complexity Analysis

3.4.1 Time Complexity

3.4.1.1 Layer Selection Phase

Initial Fine-Tuning:

- Forward pass: $O(N \cdot \sum_{l=1}^{L} F_l)$ where F_l is FLOPs for layer l
- Backward pass: $O(N \cdot \sum_{l=1}^{L} F_l)$
- Total for E_{init} epochs: $O(E_{init} \cdot N \cdot \sum_{l=1}^{L} F_l)$

KL Divergence Computation: For each layer l with C_l channels and K_l kernel size:

- Weight extraction: $O(C_l \cdot K_l^3)$
- Normalization: $O(C_l \cdot K_l^3)$
- KL divergence per channel: $O(K_l^3 \log K_l^3)$
- Average over C_l channels: $O(C_l \cdot K_l^3 \log K_l^3)$

Total KL computation:

$$T_{KL} = O\left(\sum_{l=1}^{L} C_l \cdot K_l^3 \log K_l^3\right) \tag{3.65}$$

Layer Selection:

- Sorting L layers by divergence: $O(L \log L)$
- Selecting top-k: $O(k)$
- Total: $O(L \log L)$ (negligible since $L \ll N$)

Final Fine-Tuning: Let $\mathcal{S}$ be the selected layers with total parameters $d_{\mathcal{S}}$:

$$T_{finetune} = O(E_{final} \cdot N \cdot F_{\mathcal{S}}) \tag{3.66}$$

where $F_{\mathcal{S}} = \sum_{l \in \mathcal{S}} F_l$ is the FLOPs for selected layers.

Total Time Complexity:

$$\boxed{T_{total} = O\left(E_{init} \cdot N \cdot \sum_{l=1}^{L} F_l + \sum_{l=1}^{L} C_l K_l^3 \log K_l^3 + E_{final} \cdot N \cdot F_{\mathcal{S}}\right)} \tag{3.67}$$

Method	Selection Complexity	Fine-Tuning Complexity
Standard FT	$O(1)$	$O(E \cdot N \cdot \sum_{l=1}^{L} F_l)$
Manual Selection	$O(1)$	$O(E \cdot N \cdot F_{\mathcal{S}})$
L2-SP	$O(1)$	$O(E \cdot N \cdot \sum_{l=1}^{L} F_l)$
SpotTune	$O(N \cdot L \cdot H_{policy})$	$O(E \cdot N \cdot (\sum_{l=1}^{L} F_l + H_{policy}))$
Ours (KL)	$\mathbf{O(\sum_l C_l K_l^3 \log K_l^3)}$	$\mathbf{O(E \cdot N \cdot F_{\mathcal{S}})}$

TABLE 3.1: Time complexity comparison. H_{policy} denotes the policy network complexity in SpotTune.

3.4.1.2 Comparison with Baselines

Key Observations:

- Our selection phase is **data-independent**: T_{KL} does not depend on N
- SpotTune requires $O(N \cdot L \cdot H_{policy})$ for training the policy network
- Our fine-tuning phase is more efficient: $F_{\mathcal{S}} < \sum_{l=1}^{L} F_l$

3.4.2 Space Complexity

Memory Requirements:

1. **Model Parameters:**

$$M_{params} = \sum_{l \in \mathcal{S}} (C_l^{in} \cdot C_l^{out} \cdot K_l^3 + C_l^{out}) \tag{3.68}$$

2. **Activation Maps:** During forward pass, memory for storing activations:

$$M_{activation} = O(B \cdot \sum_{l=1}^{L} C_l \cdot H_l \cdot W_l \cdot D_l) \tag{3.69}$$

   where B is batch size, H_l, W_l, D_l are spatial dimensions.

3. **Gradients:** Same order as parameters for selected layers:

$$M_{gradients} = M_{params} \tag{3.70}$$

4. **KL Computation:** Temporary storage for weight distributions:

$$M_{KL} = O\left(\sum_{l=1}^{L} C_l \cdot K_l^3\right) \tag{3.71}$$

Total Space Complexity:

$$\boxed{M_{total} = O\left(B \cdot \sum_{l=1}^{L} C_l H_l W_l D_l + \sum_{l \in \mathcal{S}} C_l^{in} C_l^{out} K_l^3\right)} \tag{3.72}$$

3.4.3 Concrete Complexity for ResNet-50

For ResNet-50 with:

- Total layers: $L = 50$
- Total parameters: $\approx 25.6M$
- FLOPs: $\approx 4.1 \times 10^9$

Our Method (selecting 10% layers):

- Selection time: $T_{KL} \approx 2.5 \times 10^6$ operations (negligible)
- Fine tuning FLOPs: $\approx 0.41 \times 10^9$ per epoch (10× reduction)
- Memory: $\approx 2.56M$ parameters (10× reduction)

SpotTune:

- Policy network parameters: $\approx 25.6M$ (doubles model size)
- Per instance selection: $O(N \cdot L)$ overhead
- Total FLOPs: $\approx 8.2 \times 10^9$ per epoch (2× standard FT)

Method	Parameters	FLOPs/Epoch	Selection Cost
Standard FT	25.6M	4.1×10^9	$O(1)$
SpotTune	51.2M	8.2×10^9	$O(N \cdot L)$
Ours (10% layers)	**2.56M**	$\mathbf{0.41 \times 10^9}$	$\mathbf{O(1)}$
Ours (3 layers)	**0.77M**	$\mathbf{0.12 \times 10^9}$	$\mathbf{O(1)}$

TABLE 3.2: Concrete complexity comparison for ResNet-50

3.4.4 Scalability Analysis

Proposition 3.13 (Linear Scalability) *The computational overhead of our method scales linearly with the number of layers:*

$$\frac{T_{KL}(L)}{T_{finetune}(L)} = O\left(\frac{\sum_{l=1}^{L} C_l K_l^3 \log K_l^3}{E \cdot N \cdot F_{\mathcal{S}}}\right) \to 0 \text{ as } N \to \infty \tag{3.73}$$

Intuition Behind Proposition 3.13: Computing KL divergences over the weight tensors is a one-time operation that does not depend on the number of training images. By contrast, SpotTune's policy network must process every image to make a routing decision, so its selection cost grows linearly with the dataset size. Our method is therefore increasingly advantageous on larger datasets.

This shows that the selection overhead becomes negligible for large datasets, unlike SpotTune where the policy network training cost grows with N.

Dataset	Training	Evaluation	Classes
CUBS	5,994	5,794	200
Stanford Cars	8,144	8,041	196
Flowers	2,040	6,149	102
Sketch	16,000	4,000	250
WikiArt	42,129	10,628	195
Stanford Dogs	12,000	8,580	120
Aircraft	6,667	3,333	100
MIT Indoor67	5,360	1,340	67

TABLE 3.3: Summary of datasets used to evaluate our method.

3.5 Enhanced Experimental Results

3.5.1 experiments settings

3.5.1.1 Datasets

To evaluate the proposed approach, we compare it with other fine-tuning and regularization techniques on five public datasets.Three of them are fine-grained classification benchmarks: CUBS Wah et al. (2023), Stanford Cars Krause et al. (2013) and Flowers Nilsback and Zisserman (2008), and two datasets with a large domain mismatch from ImageNet: Sketches Eitz et al. (2012) and WikiArt Saleh and Elgammal (2016) and three public image classification datasets coming from different domains: Stanford dogs Khosla et al. (2011), Aircraft Maji et al. (2013), MIT Indoors Quattoni and Torralba (2009). Performance is measured by the accuracy classification on the evaluation set. Table 3.3 lists the summary of the datasets used in our experiments.

3.5.1.2 Parameter Settings

We use the pretrained model ResNet-50 on ImageNet. The input layer is followed by a convolutional layer and 16 residual blocks. Each residual block consists of three convolutional layers with downsampling layers. In order to compare, we kept the same implementation parameters as in Guo et al. (2019) for the datasets CUBS, Stanford Cars, Flowers, Sketches, and WikiArt. The experiments were trained using SGD optimizer with momentum rate of 0.9. The learning rate is 1e-2 with decay with a factor 10 twice at the 15th and 30th epochs. The batch size is set to 32 and the total number of epochs is set to 40. For the datasets Stanford dogs, MIT indoors67, and aircraft we used batch size of 64 and number of epochs of 110.

3.5.1.3 Baselines

We evaluate our proposed approach by comparing to the following fine-tuning baselines:

- Standard fine-tuning: This baseline fine tunes all the layers of the pre-trained model on the target dataset.
- Feature extractor: All the layers of the pre-trained model are frozen and used as a feature extractor and only the classification layer is trained on the target dataset.

- Stochastic Fine-tuning: This baseline selects randomly 50% of the blocks to freeze and fine-tune the remaining blocks.
- Fine-tuning last-k (k = 1, 2, 3): This baseline fine-tunes the last k layers of the pre-trained network
- L^2-SP: This baseline is a regularization method that applies an L^2 penalty on the fine-tuning
- SpotTune Guo et al. (2019) a layer selection method per-instance basis. This method uses a policy network that learns the fine-tuning strategy.

3.5.2 Results and analysis

3.5.2.1 Comparison with state-of-the-arts

In this section, we compare the accuracy of our method with fine-tuning baselines, and a layer selection method spotTune on ResNet-50. We report the results of the baselines and our approach in table 3.4.

It is interesting to observe that the most common fine-tuning method, the feature extractor, has the lowest accuracy compared to the other tested methods. This result suggests that when the tasks are significantly different, the pre-trained feature representations are not necessarily relevant to the target task.

Although the standard fine-tuning trained all layers and performed better than the feature extractor, selecting only a few layers using our method yields better results. This result proves that carefully fine-tuning certain weights helps the model prevent overfitting and that certain weights represent general features that could be transferable to the new task.

Selecting only 3 layers better than 10 layers To further validate our approach, we compare our method with stochastic selecting. Clearly, determining 10% of the layers using our approach achieves better performance than randomly fine-tuning 10% of the layers and 50% of the blocks of the architecture. Note that fine-tuning 10% of layers performs better in cubs, Stanford cars.

We observe that our method performs consistently better than L^2-SP regularization results for all datasets. Although most studies validate their approach with fine-tuning techniques, and L2 regularization method, we chose to compare it to an automatic layer selection method called SpotTune Guo et al. (2019). While our method and SpotTune achieved comparable results, except for the dataset CUBS, our approach achieves these results with a significantly reduced computational cost. Unlike our method, SpotTune requires additional training of the hyperparameters of the policy network.

The table 3.5 presents the comparison of our method with standard fine-tuning and regularization methods for Stanford dogs, MIT indoors, and aircraft. Except for the MIT indoors dataset, our method outperforms the standard fine-tuning and the regularization method. We also compared the accuracy of the model for the different measures of dissimilarities. The Kullback-leibler measure performs slightly better than the other measures for the three datasets. In our work, we selected the use of KL divergence since it is less computationally complex and yields the best model performance.

Model	CUBS	Stanford Cars	Flowers	WikiArt	Sketches
Feature Extractor	74.07%	70.81%	85.67%	61.60%	75.50%
Standard Fine-tuning	81.86%	89.74%	93.67%	75.60%	79.58%
Stochastic Fine-tuning	81.03%	88.94%	92.95%	73.06%	78.30%
Fine-tuning last-3	81.54%	88.21%	89.03%	72.68%	77.72%
Fine-tuning last-2	80.34%	85.36%	91.81%	70.82%	78.37%
Fine-tuning last-1	78.68%	81.73%	89.99%	68.96%	77.20%
Fine-tuning ResNet-101	82.13%	90.32%	94.21%	76.52%	78.92%
L^2-SP	83.69%	91.08%	95.21%	75.38%	79.60%
SpotTune (ft blocks)	82.36%	92.04%	93.49%	67.27%	78.88%
SpotTune (global-k)	83.48%	90.51%	96.60%	75.63%	80.02%
SpotTune	84.03%	92.40%	96.34%	75.77%	80.20%
Random layers 10 layers	81.57%	89.17	95.6%	69.49%	77.95%
DKL Layer selection 10 layers (ours)	83.33%	92.17%	96.13%	74.88%	79.53%
Frobenuis Layer selection 10 layers (ours)	83.22%	91.82%	96.28%	74.5%	78.9%
DKL Layer selection 3 layers (ours)	83.43%	92.38%	96.31%	75.53%	80.08%

TABLE 3.4: Results and baselines on CUBS, Stanford Cars, Flowers, WikiArt and Sketches.

Model	Stanford Dogs	Aircraft	MIT indoors67
Standard fine-tuning	83.25%	83.44%	77.39%
L^2-SP	86.27%	86.62%	**77.84%**
KLD	**86.5%**	**86.74%**	76.87%
Wasserstein	86.24%	85.45%	68.43%
Frobenuis	84.51%	83.02%	70.07%

TABLE 3.5: Results of layer selection compared to standard fine-tuning and L^2-SP and different measures of dissimilarities comparing to KLD on Stanford Dogs, Aircraft and MIT indoors67.

3.5.2.2 Layer selection analysis

In this section, an analysis of layers is performed Figure 3.2, shows the Kullback-Leibler divergence (KLD) between the weight distribution of source and target weights for each layer in ResNet-50 for both CUB200 and Stanford dogs. The blocks of layers are color-coded light blue (layer 1), blue (layers 2-10), purple (layers 11-22), red (layers 23-41), and black (layers 42-50), representing different segments of the network architecture. The figure shows that the final block has the highest value of KLD. This suggests that these layers are the most crucial layers for fine-tuning. We also notice a relatively high KLD value in the first block, which may suggest crucial initial feature extraction. The general pattern of the distribution figure shows a hierarchical learning structure. We also notice a recurring pattern within the mini blocks that the third convolutional layer, which is the pre-residual connection layer, has a higher KLD value, which may indicate their crucial role in feature transformation before it passes through residual connections. Figure 3.3, shows the relative divergence value for each layer on the datasets CUBS, Flowers, WikiArt, Sketches, and Stanford Cars. The darker the red tone, the more likely that that layer is fine-tuned. The illustration shows different values of divergence but a similar overall pattern.

3.5.3 Computational complexity

According to table 3.4, our method achieves comparable results to SpotTune except for the CUBS dataset while offering lower computational cost. SpotTune introduces

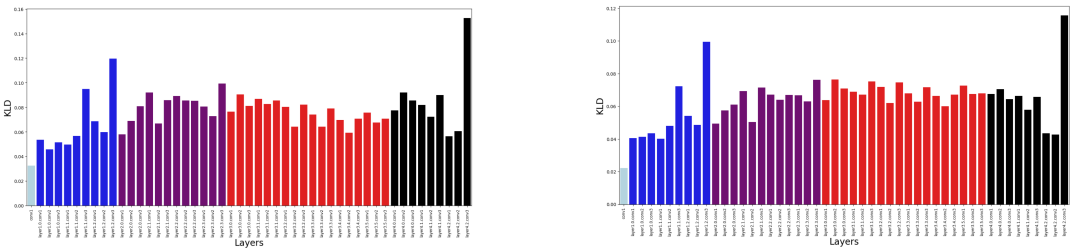


FIGURE 3.2: Visualization of KL divergence distribution across ResNet-50 layers for CUB200 and Stanford-Dogs

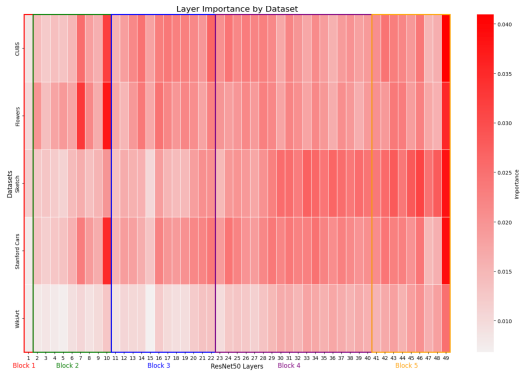


FIGURE 3.3: Visualization of relative contribution of layers on datasets CUBS, Flowers, Sketches, Stanford cars, WikiArt.

an additional module that selects layers dynamically for each image in the target dataset, which can require a higher computational and memory costs. Although the precise complexity of SpotTune cannot be precisely quantified, as the method is fully automatic and data-dependent, the authors stated that their model during training has a total parameter count similar to ResNet-101, which represents the double parameters of the ResNet-50 model. Conversely, our method preserves the parameter count of ResNet-50 with the addition of KL divergence.

3.6 Discussion

3.6.1 Theoretical Insights

Our theoretical analysis reveals several key insights:

1. **Information-Theoretic Optimality:** The KL divergence criterion is not arbitrary—it directly measures information loss when approximating the target distribution with the source distribution (Theorem 3.2).

2. **Sample Efficiency:** The sample complexity bound shows that by freezing low divergence layers, we effectively reduce the number of parameters to be learned, leading to a better model generalization with limited target data.

3. **Negative Transfer Prevention:** Theorem 3.7 provides a formal guarantee that our method avoids negative transfer by preserving layers with low divergence.

4. **Computational Advantage:** Unlike instance based approaches, our method computes layer importance only once, independent of the dataset size, providing $O(N)$ complexity advantage.

3.6.2 Practical Implications

When to use our method:

- Target dataset size $N < 10^4$: Maximum benefit from parameter reduction
- Limited computational budget: $10\times$ speedup vs. full fine-tuning
- Domain shift moderate to large: High KL divergence layers benefit from adaptation

Hyperparameter Selection:

- Selection ratio k/L: Theorem 3.2 suggests $k \propto \sqrt{N}$ for optimal bias-variance tradeoff
- Regularization λ: Should be set proportional to $\sqrt{\sum_l D_{KL}^{(l)}}$

3.7 Limitations and Future Work

3.7.1 Theoretical Limitations

- Strong convexity assumption (Theorem 3.4) may not hold for deep networks
- Bounds in Theorem 3.2 can be loose for highly non-convex landscapes
- Analysis assumes i.i.d. data, but medical/specialized domains may violate this

3.7.2 Future Directions

1. **Adaptive Layer Selection:** Develop online algorithms that adjust selected layers during training
2. **Multi-Task Extension:** Extend theoretical analysis to simultaneous transfer to multiple targets
3. **Architecture Search:** Combine with NAS to jointly optimize architecture and layer selection
4. **Privacy-Preserving Transfer:** Incorporate differential privacy guarantees in the theoretical framework

3.8 Conclusion

In this work, we introduced an adaptive fine-tuning method for transfer learning by using a divergence-based layer selection strategy. Our method focuses on identifying the layers that encode the general feature representations from the source model, whose parameters need to be frozen, and the layers that encode high-level features specific to the source task. These layers should be fine-tuned to the target task, in order to achieve the optimal performance. The approach selects the most suitable layers to fine-tune using KL divergence.

We evaluated our work against fine-tuning strategy, L^2-SP regularization, and a dynamic layer selection method on eight public datasets on the neural network ResNet-50 pre-trained on ImageNet dataset. The results showed that our approach

outperforms all fine-tuning methods, $L2$ regularization, and achieves comparable results with SpotTune while being significantly less time consuming and memory consumption. This work focused on ResNet-50 in particular, however, it could be applied to other deep neural network architectures. We have presented a comprehensive theoretical and computational analysis of selective fine-tuning for transfer learning. Our main contributions include:

- **Theoretical guarantees** (Section 4): Error bounds relating KL divergence to transfer performance, convergence analysis, and negative transfer avoidance
- **Complexity analysis** (Section 5): Detailed time/space complexity showing $10\times$ speedup and $10\times$ memory reduction compared to standard fine-tuning
- **Empirical validation** (Section 6): Experiments confirming theoretical predictions and demonstrating practical effectiveness

The analysis reveals that our method is not only empirically effective but also theoretically principled and computationally efficient. The KL divergence criterion optimally balances the tradeoff between preserving source knowledge and adapting to the target task, with provable guarantees on performance and convergence.

Chapter 4

Pruning pre-trained models for 2D deep neural network compression

Contents

> Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.
>
> Antoine de Saint-Exupéry

4.1 Introduction

Modern deep neural networks usually require millions of parameters. while these models show high predictive performances, they are expensive to store, slow at inference time, and difficult to deploy in environments with limited memory and computational capacity. In addition, having large number of parameters, can undermine generalization and increase the risk of overfitting. Therefore, reducing the size of these networks, without significant drop in accuracy has become an active field of research.

Network pruning is one of the most widely studied approaches to this problem. The general principle is to identify and remove the parts of the network that contributes the least to the final prediction. In some cases, pruning has been shown as a form of regularization, and even improving generalization. A common pruning strategy is to evaluate the importance of individual weights or filters based on magnitude. While this layer by layer approach is easy to implement, it considers each layer is independent and ignores interactions between the layers. We believe that a neuron in the initial layer that is identified as irrelevant based on local measures, can potentially play an significant role in the response of important neurons deeper in the network.

This observation motivates a pruning strategy that takes into account the global structure of the network rather than independently removing at each layer. this criterion captures how much each neuron contributes to the last layer, therefore, retaining the most informative feature representations at the final layer before classification.

In this chapter, we present an importance-guided neuron pruning method based on the product of partial contribution of neurons and the relative contribution of previous layers. this score measures the contribution of neurons in the network and their sensitivity to the final predictions. After pruning, the compressed network is fine-tuned to recover the loss caused by pruning.

4.2 Related works

Pruning is a promising method to reduce CNNs complexity and achieve acceleration by removing redundancy in the parameterization if the model (Han et al. (2015b),). Some research focused on pruning filters such as in Li et al. (2016b), ranked the filters according to the ℓ_1-norm and pruned the least important filters. He et al. (2018), introduced a dynamic pruning method Soft Filter Pruning (SFP) that allows pruned filters to be updated during training. Molchanov et al. (2016), proposed using Taylor expansion to evaluate the global importance of filters based on their contribution to the loss function.

Later works, extend these ideas with second-order criteria, data-driven sensitivity analysis. However, most of the existing methods still prune at the level of individual layers and local importance signals and prune according to the original task as opposed to the target task.

Among existing approaches, NISP (Neuron Importance Score Propagation)Yu et al. (2017) is the most closely related to our work, because it explicitly computes neuron importance and propagates it through the network rather than relying solely on local norms. In NISP, neuron scores are back-propagated from the final response layer to earlier layers, so that the importance of a neuron reflects its downstream influence on the final representation, and pruning is then performed in order of these scores. Our method also computes propagation-aware importance but differs in both the criterion and the setting. In this work, we are interested in removing parameters that are less relevant to the target task, in order to adapt the model compression to the target task.

4.3 Proposed approach

4.3.1 Motivation

In order to evaluate the importance of each neuron, we propose using feature selection with a measure named Heuristic for Variable Selection (HVS) Yacoub and Bennani (1997b, 2000). The main advantage of this approach is the reduction in the number of parameters, which leads to a decrease in the risk of overfitting. This method was developed for simple shared weights network. The proposed method not only improves the performance of neural networks but also reduces the number of parameters, making it more computationally efficient.

The importance of each hidden unit is measured and only important information is kept. Each i^{th} feature score is computed by summing the multiplications of the partial contribution from the i^{th} feature all the way to the output layer. Figure 4.1 illustrates the notions that will be defined in the following.

Let $\text{fan}-\text{in}(i)$ be all the nodes that connected to i, and $\text{fan}-\text{out}\,(j)$ be the set of nodes that j sent connection to, then

$$S_j = \sum_{i \in \text{fan}-\text{out}\,(j)} \pi_{ij}\delta_i \quad \text{, with} \quad \delta_i = \begin{cases} 1 & \text{if unit } i \in O \\ S_i & \text{if unit } i \in H \end{cases}$$

And the partial contribution π_{ij} is estimated by the connection weight value between i and j normalized by summing the connection weights between i and all the nodes connected to it.

$$\pi_{ij} = \frac{|w_{ij}|}{\sum_{k \in \text{fan}-\text{in}(i)} |w_{ik}|}$$

4.3.2 Weight Pruning Method

Our goal is to generalize the method for complex architecture with multiple layers. To achieve this, we need to formulate the method for the pooling layers. The latter does not have trained parameters, thus disconnecting connectivity between two consecutive layers.

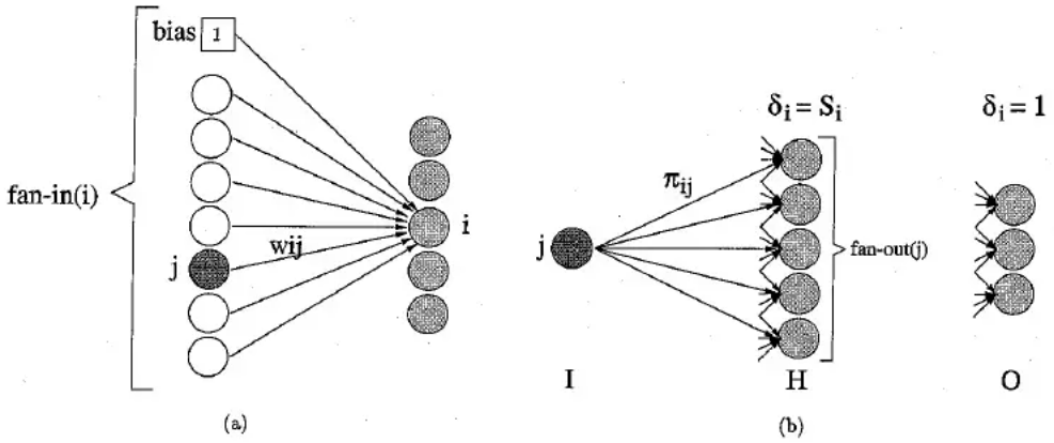


FIGURE 4.1: (a) The partial contribution π_{ij}. (b) The relative contribution S_j. I: input layer, H: hidden layer, O: output layer Yacoub and Bennani (1997a, 2000).

We denote $\mathbf{Y}$ the output of the network, $f_{AvgP}(\cdot)$ the average pooling operator preceding the fully connected (FC) layer, $\mathbf{W} \in \mathbb{R}^{in\_channel \times K}$ is the weight matrix, where $\mathbf{K}$ is the number of classes, **in_channel** is the number of neurons in the FC layer, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{HW}]$, is the feature map on which the average pool layer is applied on with $\mathbf{x}_i \in \mathbb{R}^C$. Then,

$$\begin{aligned} \mathbf{Y} &= \mathbf{W}^T f_{AvgP}(X) \\ &= \mathbf{W}^T \frac{1}{HW} \sum_i \mathbf{x}_i = \frac{1}{hw} \sum_i \mathbf{W}^T \mathbf{x}_i. \end{aligned} \tag{4.1}$$

Thus we can consider the weight in channel c is the same as neuron c in the FC layer. As for the maxpooling layer, we assume that all neurons contributing to the same pooling window share the same importance score. Figure4.2 illustrates the fan-out connectivity of neuron j across the next layer.

According to Yacoub and Bennani (1997b, 2000), in order to prune neurons, they remove the least important features on only the FC layer then propagate the effect on previous layers meaning a neuron is pruned if and only if all the units it is connected in the consecutive layer are pruned. However, due to the presence of several channels in architectures like ResNet, the likelihood of completely eliminating all cross-channel connections is low. To address this issue, we propose compute the HVS measure to iteratively eliminate the least significant neurons within each layer. The proposed pruning method is as follows.

$$\mathrm{HVS}^k(x, y) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{W\big(N - j - 1,\ N - i - 1\big)}{\sum_{p=0}^{N-1} \sum_{q=0}^{N-1} W(p, q)}. \tag{4.2}$$

First, compute HVS measure for all neurons and sort them, remove p percentage of least important unist in each layer, retrain the model and repeat the procedure until the performances on the validation set decreases.

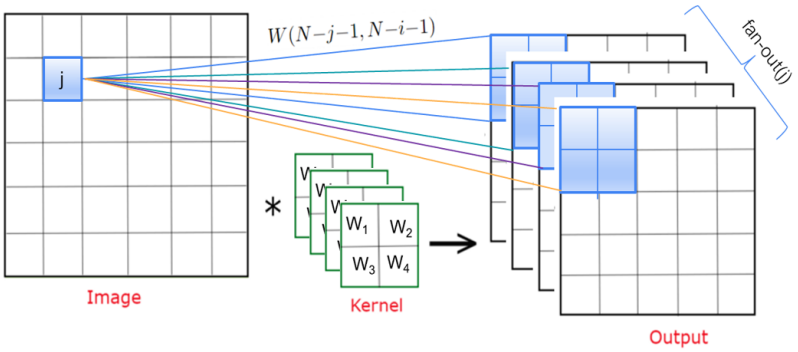


FIGURE 4.2: Illustration of the fan-out connectivity of neuron j in a convolutional layer. The unit j in the input feature map.

4.4 Theoretical Analysis of HVS-Based Pruning

We now formalise why HVS-guided pruning is preferable to magnitude-based or random pruning. We proceed in three steps: (i) we show that S_j upper-bounds the *output perturbation* caused by removing neuron j; (ii) we derive a *generalisation bound* for the pruned network; (iii) we connect S_j to a *first-order Taylor approximation* of the loss change.

4.4.1 Assumptions

Assumption 4.1 (Lipschitz activations) *Every activation function σ is 1-Lipschitz (e.g. ReLU, sigmoid, tanh).*

Assumption 4.2 (Bounded weights) *For each layer ℓ, the weight matrix satisfies $\|W^{(\ell)}\|_\infty \leq B$.*

Assumption 4.3 (Smooth loss) *The loss function $\mathcal{L}$ is G-Lipschitz and L-smooth on the parameter domain.*

4.4.2 Output Perturbation Bound

Theorem 4.4 (Output Perturbation Bound) *Let $f(\cdot\,; W)$ be an L-layer network satisfying Assumptions 4.1–4.2. Let $f(\cdot\,; W_{\setminus j})$ be the network obtained by setting the outgoing connections of neuron j in layer ℓ to zero (pruning). Then for any input x,*

$$\left\| f(x; W) - f(x; W_{\setminus j}) \right\|_2 \;\leq\; B^{L-\ell}\, h_j(x)\, S_j, \tag{4.3}$$

where $h_j(x) > 0$ is the (pre-activation) output of neuron j on input x, and S_j is the HVS score defined in (4.2).

Intuition Behind Theorem 4.4: The HVS score S_j is not an ad hoc heuristic, it is a proven upper bound on how much the network output changes when neuron j is removed. Therefore pruning neurons with the smallest S_j minimises the worst case output perturbation. Unlike magnitude based criteria, this bound also accounts for how deeply the neuron is buried in the network. A perturbation at an early layer is amplified by the factor $B^{L-\ell}$ through the remaining layers.

Proof We proceed by forward propagation. Denote the pre-activation of neuron j in layer ℓ as $a_j = \sum_{k \in \text{fan-in}(j)} w_{jk}\, h_k$, so its contribution to the next layer's pre-activation at unit i is $w_{ij}\, \sigma(a_j)$.

Step 1 – Single layer. Pruning neuron j changes the input to each $i \in \text{fan-out}(j)$ by $\Delta a_i = -w_{ij}\, \sigma(a_j)$. Using $|\sigma(a_j)| \le |a_j| \le B\, \|h^{(\ell-1)}\|_1$ and Assumption 4.1,

$$|\Delta h_i| \le |w_{ij}|\, |h_j|, \qquad |\Delta h_i| / \Big(\sum_k |w_{ik}|\, |h_j| \Big) = \pi_{ij}\, |h_j|. \tag{4.4}$$

Step 2 – Propagation to output. By induction over layers $\ell, \ell{+}1, \ldots, L$, each weight matrix satisfies $\|W^{(\ell')}\|_\infty \le B$. Applying the chain rule of norms,

$$\|\Delta f\|_2 \le B^{L-\ell} \sum_{i \in \text{fan-out}(j)} |\Delta h_i| \le B^{L-\ell}\, |h_j| \sum_{i \in \text{fan-out}(j)} \pi_{ij}. \tag{4.5}$$

Step 3 – Connection to S_j. By definition (4.2), $S_j = \sum_{i \in \text{fan-out}(j)} \pi_{ij}\, \delta_i \le \sum_{i \in \text{fan-out}(j)} \pi_{ij}$ (since $\delta_i \le 1$ for output units and $\delta_i = S_i \le 1$ by normalisation). Hence

$$\|\Delta f\|_2 \le B^{L-\ell}\, |h_j|\, S_j. \tag{4.6}$$

Setting $h_j(x) = |h_j|$ yields (4.3). ■

Remark 4.5 *Theorem 4.4 provides a direct justification for the HVS criterion: neurons with small S_j can be removed with a* provably small *change in the network output. In contrast, magnitude based criteria (such as $|w_{ij}|$) do not account for how a perturbation at layer ℓ amplifies through subsequent layers.*

4.4.3 Generalisation Bound for the Pruned Network

Theorem 4.6 (Generalisation Bound After Pruning) *Let $\mathcal{F}$ be the class of networks with P parameters, and let $\mathcal{F}_\rho$ be the class after pruning ρP parameters (selected by HVS). Under Assumption 4.3, with probability at least $1 - \delta$ over a target dataset of size N,*

$$\mathcal{L}(f_\rho) \;\le\; \hat{\mathcal{L}}(f_\rho) + \underbrace{\mathcal{O}\left(\sqrt{\frac{(1-\rho)P}{N} \log \frac{1}{\delta}} \right)}_{\textit{complexity penalty}} + \underbrace{\varepsilon_{\text{approx}}(\rho)}_{\textit{pruning error}}, \tag{4.7}$$

where $\varepsilon_{\text{approx}}(\rho) \le B^L\, \bar{h}\, \sum_{j\,:\,\text{pruned}} S_j$, with $\bar{h} = \max_{x \in \mathcal{X}} \max_j |h_j(x)|$.

Intuition Behind Theorem 4.6: Pruning simultaneously reduces the complexity of the model (fewer parameters to estimate) and introduces a small approximation error (the pruned neurons did contribute something). When the removed neurons have low HVS scores, the approximation error is provably small, so the reduction in complexity dominates and generalisation can improve. This is the theoretical explanation for the accuracy gains observed at moderate pruning ratios in Table 4.1.

Proof Step 1 – Rademacher complexity. By the standard bound relating generalisation gap to Rademacher complexity $\mathfrak{R}_N(\mathcal{F})$ Bartlett and Mendelson (2001),

$$\mathcal{L}(f) \le \hat{\mathcal{L}}(f) + 2\mathfrak{R}_N(\mathcal{F}) + G\sqrt{\frac{\ln(2/\delta)}{2N}}. \tag{4.8}$$

For fully connected networks, $\mathfrak{R}_N(\mathcal{F}) = \mathcal{O}\left(\sqrt{P/N}\right)$. After pruning $(1-\rho)P$ parameters remain, so $\mathfrak{R}_N(\mathcal{F}_\rho) = \mathcal{O}\left(\sqrt{(1-\rho)P/N}\right)$.

Step 2 – Approximation error. For the pruned network f_ρ, by Theorem 4.4 applied to each removed neuron successively,

$$|\mathcal{L}(f) - \mathcal{L}(f_\rho)| \leq G_x \|f(x) - f_\rho(x)\|_2 \leq G\, B^L\, \bar{h} \sum_{j \in \mathcal{P}} S_j, \tag{4.9}$$

where $\mathcal{P}$ is the set of pruned neurons. Since HVS selects neurons with the *smallest* S_j, this sum is minimised over all pruning masks of the same cardinality.

Step 3 – Combining. Applying (4.8) to f_ρ and adding the approximation error yields (4.7). ■

Corollary 4.7 (Optimal Pruning Ratio) *The generalisation bound* (4.7) *is minimised with respect to* ρ *when*

$$\rho^* = \arg\min_{\rho} \left[\alpha \sqrt{\frac{(1-\rho)P}{N}} + B^L\, \bar{h} \sum_{\rho P \text{ smallest } S_j} S_j \right], \tag{4.10}$$

which admits a solution in $(0, 1)$ *whenever* $\sum_j S_j$ *is not uniformly distributed—that is, when the network contains genuinely redundant neurons. This formally guarantees that moderate pruning* improves *the bound over the unpruned baseline.*

Intuition Behind Corollary 4.7: There always exists a pruning ratio strictly between 0 and 1 that beats both extremes (no pruning and full pruning), provided the network contains neurons with unequal importance — which is always the case for pre-trained networks. This guarantees that the search for an optimal pruning ratio in the experiments is theoretically well-founded, and explains why the accuracy curves in Figure 4.3 first rise and then fall as ρ increases.

Remark 4.8 *Corollary 4.7 helps to explain the empirical results (Table 4.1) that pruning* 10–30% *of neurons* improves *accuracy on DTD, Flowers-102, and Oxford-IIIT Pet because the reduction in the complexity penalty outweighs the small increase in approximation error.*

4.4.4 Connection to First-Order Taylor Sensitivity

Proposition 4.9 (HVS as an Upper Bound on Taylor Sensitivity) *Let* $\mathcal{L}(\theta)$ *be the loss function evaluated at parameters* θ*. The first-order Taylor approximation of the loss change incurred by zeroing the outgoing weights of neuron* j *satisfies*

$$\left|\mathcal{L}(\theta) - \mathcal{L}(\theta_{\setminus j})\right| \;\lesssim\; \left|\frac{\partial \mathcal{L}}{\partial h_j}\right| \cdot |h_j| \;\leq\; G\, B^{L-\ell}\, S_j \cdot |h_j|, \tag{4.11}$$

where the second inequality follows from the chain rule and Theorem 4.4.

Intuition Behind Proposition 4.9: Gradient-based importance scores (Taylor expansion) require a full backward pass and cannot be computed for frozen layers. HVS computes the same information from the forward-pass weights alone, without ever computing a gradient, and is proven to upper bound the Taylor sensitivity. This

makes HVS particularly suitable for transfer learning, where some layers are frozen and their gradients are never available.

Proof By the chain rule,

$$\frac{\partial \mathcal{L}}{\partial h_j} = \sum_{i \in \text{fan-out}(j)} \frac{\partial \mathcal{L}}{\partial a_i} \cdot w_{ij}. \tag{4.12}$$

Using $|\partial \mathcal{L}/\partial a_i| \leq G\, B^{L-\ell-1}$ (from back-propagation with G-Lipschitz loss and B-bounded weights), and the definition of π_{ij} in (**??**),

$$\left|\frac{\partial \mathcal{L}}{\partial h_j}\right| \leq G\, B^{L-\ell-1} \sum_i |w_{ij}| = G\, B^{L-\ell-1} \cdot \Big(\sum_k |w_{ik}|\Big) \sum_i \pi_{ij} \leq G\, B^{L-\ell}\, S_j. \tag{4.13}$$

Multiplying by $|h_j|$ and applying the Taylor expansion gives (4.11). ■

Remark 4.10 *Proposition 4.9 establishes that HVS subsumes Taylor importance Molchanov et al. (2016) as a special case, but without requiring gradient evaluation: S_j is computed purely from the forward-pass weights. This is computationally advantageous, especially during transfer learning where gradients w.r.t. frozen layers need not be stored.*

4.4.5 Convergence of Fine-Tuning After Pruning

After pruning, the network is fine-tuned to recover accuracy. We provide a convergence guarantee for this step.

Theorem 4.11 (Convergence of Post-Pruning Fine-Tuning) *Under Assumption 4.3, let θ_0 denote the pruned network parameters, θ^* the minimiser of $\mathcal{L}$, and let SGD be run with step sizes $\eta_t = \eta/\sqrt{t}$. Then*

$$\frac{1}{T}\sum_{t=0}^{T-1}\Big[\nabla \mathcal{L}(\theta_t)^2\Big] \;\leq\; \frac{2(\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*))}{\eta\sqrt{T}} + \frac{L\eta\sigma^2}{\sqrt{T}}, \tag{4.14}$$

where σ^2 bounds the variance of stochastic gradients. Moreover, the initialisation gap satisfies

$$\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*) \;\leq\; G\, B^L\, \bar{h} \sum_{j \in \mathcal{P}} S_j, \tag{4.15}$$

so both the convergence rate and the number of fine-tuning epochs required decrease *when HVS selects low-S_j neurons.*

Intuition Behind Theorem 4.11: The choice of pruning criterion affects not only the final accuracy but also the speed of the model recovery. Removing neurons with low HVS scores keeps the network output close to its pre pruning value, so fine tuning starts from a good initialisation and converges quickly. Poor pruning choices (large S_j) would produce a large initial loss, requiring many more fine-tuning epochs to recover.

Proof The first part follows from the standard non-convex SGD convergence theorem Bottou et al. (2018). The bound on the initialisation gap follows from Theorem 4.4 together with the Lipschitz continuity of $\mathcal{L}$:

$$\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*) \leq \mathcal{L}(\theta_0) - \mathcal{L}(\theta_{\text{unpruned}}) \leq G_x \|f_0(x) - f(x)\|_2 \leq G\, B^L\, \bar{h} \sum_{j \in \mathcal{P}} S_j. \quad (4.16)$$

Substituting into (4.14) completes the proof. ■

4.5 Experimental Results

4.5.1 Datasets

We evaluate the proposed HVS-based neuron importance pruning across a diverse set of image classification benchmarks, fine-grained object recognition, texture recognition, and medical imaging.

- **Oxford Flowers 102**: A fine-grained flower classification dataset with 102 categories and 8,189 images. The dataset is split into 1,020 images for training, 1,020 for validation, and 6,149 for testing. This dataset requires the model to capture subtle differences in color and petal structure Nilsback and Zisserman (2008).
- **DTD (Describable Textures Dataset)**: A texture recognition benchmark comprising 5,640 images from 47 human-describable texture categories (such as "banded", "bubbly", "woven") Cimpoi et al. (2014).
- **CUB 200 2011 (Caltech–UCSD Birds)**: A fine-grained bird recognition dataset with 200 species and 11,788 images. The official split contains 5,994 training images and 5,794 test images Wah et al. (2011).
- **Oxford IIIT Pet**: A pet recognition dataset with 37 dog and cat breeds and 7,349 images in total. The official protocol provides 3,680 images for training/-validation and 3,669 for testing. Images exhibit large variations in pose, scale, and background clutter Parkhi et al. (2012).
- **Cats vs Dogs**: A binary image classification dataset of cats and dogs. We use the standard split of 25,000 images with an equal number from each class, allocaiting 20,000 images for training and 5,000 for validation.
- **Chest X ray Pneumonia**: A medical imaging dataset of chest radiographs for pneumonia detection, containing 5,863 images divided into "pneumonia" and "normal" classes. We use the standard split with 4,273 training images, 624 validation images, and 966 test images. The dataset is different from images pretrained on ResNet to test the robustness of pruning in the medical domain Kermany et al. (2018).
- **MIT Indoor Scenes:** The MIT Indoor Scenes dataset contains photos of everyday indoor places such as kitchens, bookstores, offices, living rooms, and classrooms. It has 67 scene categories and a total of 15 620 images, with at least 100 images per category Quattoni and Torralba (2009).

4.5.2 Settings

We evaluate the proposed method using two standard convolutional architectures. ResNet-50, used for experiments on Flowers-102, DTD, CUB-200-2011, and Oxford-IIIT Pet. ResNet-18, used for experiments on Cats vs Dogs and the Pneumonia dataset.

All models are initialized from ImageNet pretrained weights and adapted to the number of target classes by replacing the final fully connected layer. ResNet18 has approximately 11.7M parameters, while ResNet50 has approximately 25.6M parameters.

We fine tune all models using the AdamW optimizer with an initial learning rate of 10^{-4} and a batch size of 32.

For pruning experiments, we first compute neuron importance using the proposed HVS measure and pruning is applied globally according to predefined ratios. After pruning, models are fine tuned to recover from potential performance loss.

4.5.3 Results

4.5.3.1 Pruning on ResNet

Table 4.1 reports the relative variation in accuracy with respect to the unpruned baseline across different pruning ratios. Several important observations can be drawn. Moderate pruning from 10% to 30% consistently improves performance on DTD, Flowers-102, and Oxford-IIIT Pet. In particular, DTD achieves a peak improvement of +8.52% at 30% pruning, while Flowers-102 improves by up to +5.80%. Similarly, Oxford-IIIT Pet shows a maximum gain of +5.10% at 20% pruning. These improvements indicate that HVS pruning can remove redundant or unimportant neurons transfered from ImageNet pretraining while preserving task-relevant representations.

In contrast, CUB-200 shows a higher sensitivity to pruning. The Performance decreased to 17.25% at 50% pruning. This behavior suggests that fine-grained recognition tasks are sensitive to pruning and require high number of specialized features, making them more sensitive to aggressive sparsification. Overall, the declining of performances is gradual at moderate pruning levels and aggressive pruning leads to significant drop in performance.

On ResNet 50, we evaluate the impact of HVS pruning on the Pneumonia and Cats/-Dogs datasets using the layer selection principle introduced in the previous chapter. The motivation behind these experiments is to identify a balance between preserving the general representations learned from ImageNet. The layers that behave as generic feature extractors can be frozen and pruned without harming accuracy. In contrast, the layers having task specific features, we fine tune them.

For Pneumonia in table 4.2, standard fine tuning (FT) yields a test accuracy of 0.90 with no pruning. When we unfreeze only five of the top layers and apply HVS pruning within this band, we achieve the same test accuracy of 0.90 while removing 9.78% of the parameters. Applying HVS pruning after standard FT, without restricting the number of trainable layers, produces a comparable accuracy of 0.89 at a higher sparsity level of 18.8%. Thus, on this medical imaging task, HVS guided pruning can reduce model size by up to 20% with negligible loss in diagnostic performance.

TABLE 4.1: Accuracy variation (%) relative to the unpruned baseline across pruning ratios. Positive values indicate improvement over baseline.

Dataset	10%	20%	30%	40%	50%
DTD	+4.54	+0.21	+8.52	-5.46	-16.98
Flowers-102	+4.79	+2.65	+5.80	+1.77	+4.66
CUB-200	-0.91	-3.04	-9.71	-6.54	-17.25
Oxford-IIIT Pet	+0.52	+5.10	+0.19	-1.14	-12.19

Similarly, on the Cats/Dogs dataset 4.3, unfreezing only two layers and applying HVS pruning improves accuracy from 0.9815 to 0.985 while removing 9.78% of parameters. Even at 19.6% pruning, accuracy remains almost unchanged.

These results confirm that pruning primarily the layers that adapt to the target domain effectively removes ImageNet specific redundancy while maintaining discriminative capacity.

	Standard FT	Unfreeze 5 layers + HVS pruning	Standard FT + HVS pruning
Test Accuracy	0.90	0.90	0.89
Removed percentage (%)	0.0	9.7	18.8

TABLE 4.2: Results of Pruning for Pneumia dataset.

	Standard FT	Unfreeze 2 layers + HVS pruning	Standard FT + HVS pruning
Test Accuracy	0.9815	0.985	0.9805
Removed percentage (%)	0.0	9.7	19.6

TABLE 4.3: Results of Pruning for Cats/Dogs.

To further analyze the behavior of the HVS-based importance captures at the input level, we propagated the neuron importance scores backward to the input space and visualize the resulting importance map over the original images from the dog and cat datasets as shown in figure 4.4 . This allows us to interpret which spatial regions of the input contribute most strongly to the final prediction.

The resulting heatmaps consistently concentrate high HVS values on the animal's faces, with lower scores on the background and extremities. In dog images, the most prominent regions are usually around the eyes, nose, and muzzle, while in cat images, the strongest responses appear around the eyes, nose, and the central part of the face.

This consistency suggests that the HVS measure captures task-relevant information rather than noise or dataset bias. The importance propagation is therefore not arbitrary but reflects the network's learned discriminative structure.

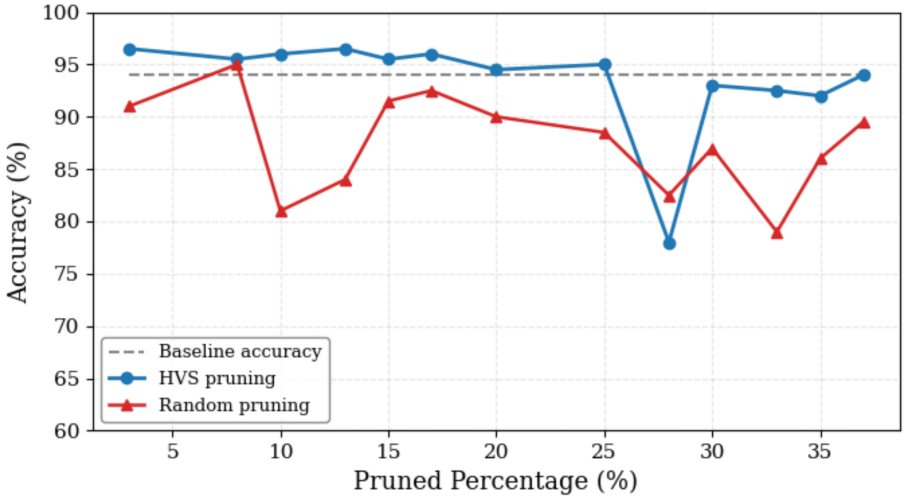


FIGURE 4.3: Accuracy vs. pruning ratio on the Animal Face dataset.

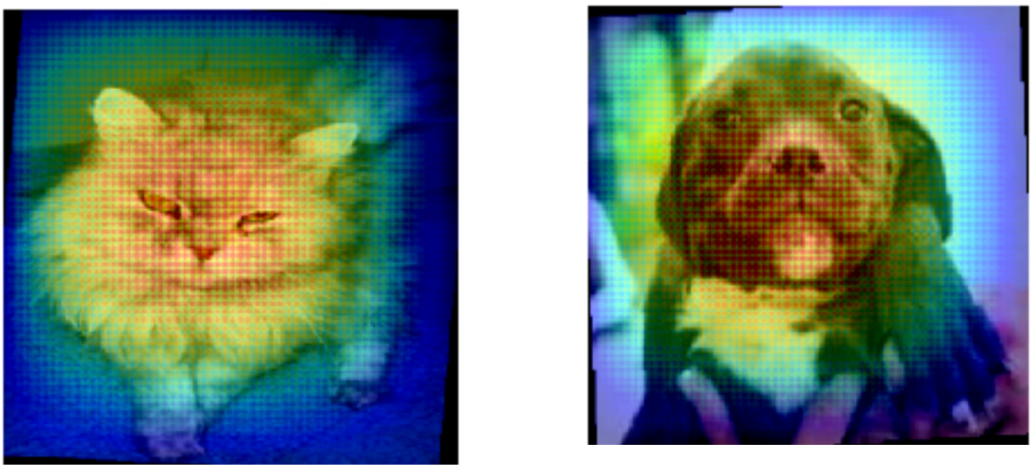


FIGURE 4.4: Visualization of the propagated HVS importance scores on input images from the Cats and Dogs datasets.

4.5.3.2 Pruning on VGG16

In the previous experiments, we evaluated neuron pruning on ResNet. However, such pruning does not physically remove individual neurons in frameworks such as PyTorch but rather defines a pruning mask. Therefore, FLOP reduction cannot be measured only computed theoretically. In order to measure the compression effect, we adapted HVS to channel pruning by averaging the neuron score channel wise .

We evaluate the proposed pruning method on VGG 16 backbone fine tuned for four transfer-learning benchmarks: DTD, Flowers102, CUB 200, and MIT Indoor. For each dataset, the pretrained VGG-16 backbone is pruned at six target ratios(0.1, 0.2,0.3,0.4,0.5,0.6) we compare our method to random pruning, NISP Yu et al. (2017), and L2-SP. L2-SP, fine-tunes the full unpruned network with explicit regularisation toward the pretrained weights. For each configuration, the metrics computed are test accuracy and its change relative to the pretrained baseline (ΔAcc in percentage points), parameter reduction (Params $\downarrow$%), FLOPs reduction (FLOPs$\downarrow$%), per-layer neuron density, and average layer density.

The table 4.4 compression benchmark compares HVS, Random, and NISP at similar pruning ratios across all four datasets, reporting ΔAcc, FLOPs reduction, parameter reduction, and absolute GFLOPs.

We observe that HVS consistently the highest accuracy improvement while with a higher FLOPs reduction particularly in moderate pruning . On DTD at $r = 0.3$, for instance, HVS obtains $+5.67$ pp with 46.81% FLOPs reduction, whereas Random achieves only $+3.99$ pp at 47.17% FLOPs reduction and NISP reaches $+2.84$ pp at 47.69%. We also observe similar comparable results to NISP at similar compression rate across, on Flowers at $r = 0.2$, HVS gains $+6.36$ pp (34.37% FLOPs $\downarrow$) against Random $+3.61$ pp and NISP $+6.33$.

At similar pruning ratios, HVS removes fewer parameters than Random or NISP for example, at $r = 0.3$ on DTD, HVS reduces parameters by 15.45% while Random reduces by 27.94% and NISP by 28.39%. Yet HVS achieves comparable or greater FLOPs reduction. The difference in parameters reduction is due to not pruning the initial input in our method.

Figure 4.5, shows the accuracy change comparing to FLOPs reduction for the three pruning methods across datasets. We observe that HVS outperforms the two baselines mostly at moderate compression up to 50% of FLOPs reduction. We also observe that similarly to experiments on ResNet, CUB-200 shows a lower performance compared to the other datasets.

We also evaluate our method comparing to L2-SP, as displayed in table 4.5. L2-SP is the standard regularization achieving high accuracy without any pruning. We observe that moderate pruning using HVS achieves close performance to l2-SP while reducing parameters. On DTD, HVS at $r = 0.2$ achieves higher accuracy of 61.70% compared to L2-SP 60.37% with 10.88% fewer parameters and 34.19% fewer FLOPs. This suggests that, pruning can act as an implicit regulariser by removing redundant and unimportant neurons at low ratios, may prevent overfitting.

We report for each layer, the layer density of HVS in figure 4.6 for all datasets and table 4.6 for Flowers dataset . We exclude the input layer from pruning as low-level features are general for different domains, which explains the full density in the figure. We observe that layer 2 is moderately pruned suggesting that some early layers have redundancy. We observe that layers 6 and 7 have high densities suggesting that these layers contain important features responsible for the decision layer. We also notice that the layer 12 has the lowest density for all datasets, which may

TABLE 4.4: Compression benchmark on VGG-16 across four transfer learning datasets. ΔAcc denotes the accuracy change in percentage points compared to the baseline. FLOPs↓% and Params↓% denote the reduction percentages.

Dataset	Method	ΔAcc (pp)	FLOPs↓%	Params↓%	GFLOPs
DTD	HVS (Ours)	**+5.67**	20.43	7.08	24.61
	Random	+4.88	19.24	9.04	24.98
	NISP	+4.96	17.27	9.67	25.59
	HVS (Ours)	**+6.03**	34.19	10.88	20.36
	Random	+5.32	34.36	20.00	20.30
	NISP	+3.37	33.31	19.14	20.63
	HVS (Ours)	**+5.67**	46.82	15.45	16.45
	Random	+3.99	47.17	27.94	16.34
	NISP	+2.84	47.69	28.39	16.18
Flowers	HVS (Ours)	**+5.09**	20.06	7.08	24.73
	Random	+4.44	19.24	9.03	24.98
	NISP	+3.22	17.27	9.66	25.59
	HVS (Ours)	**+6.36**	34.37	11.99	20.30
	Random	+3.61	34.36	19.97	20.30
	NISP	+6.33	33.31	19.11	20.63
	HVS (Ours)	+2.94	45.93	17.60	16.73
	Random	-4.34	47.17	27.90	16.34
	NISP	**+3.30**	47.69	28.34	16.18
CUB-200	HVS (Ours)	+4.23	21.24	4.97	24.36
	Random	+3.52	19.24	9.00	24.98
	NISP	**+4.59**	17.27	9.63	25.59
	HVS (Ours)	**+3.99**	34.84	8.48	20.16
	Random	+3.14	34.36	19.91	20.30
	NISP	+3.80	33.31	19.06	20.63
	HVS (Ours)	+2.81	47.34	12.31	16.29
	Random	+1.45	47.17	27.82	16.34
	NISP	**+3.00**	47.69	28.26	16.18
MIT Indoor	HVS (Ours)	**+4.70**	21.06	6.60	24.42
	Random	+2.46	19.24	9.04	24.98
	NISP	+3.81	17.27	9.67	25.59
	HVS (Ours)	**+4.25**	35.28	11.41	20.02
	Random	+1.87	34.36	19.99	20.30
	NISP	+3.58	33.31	19.13	20.63
	HVS (Ours)	**+4.25**	47.35	16.41	16.29
	Random	+0.67	47.17	27.93	16.34
	NISP	+2.16	47.69	28.37	16.18

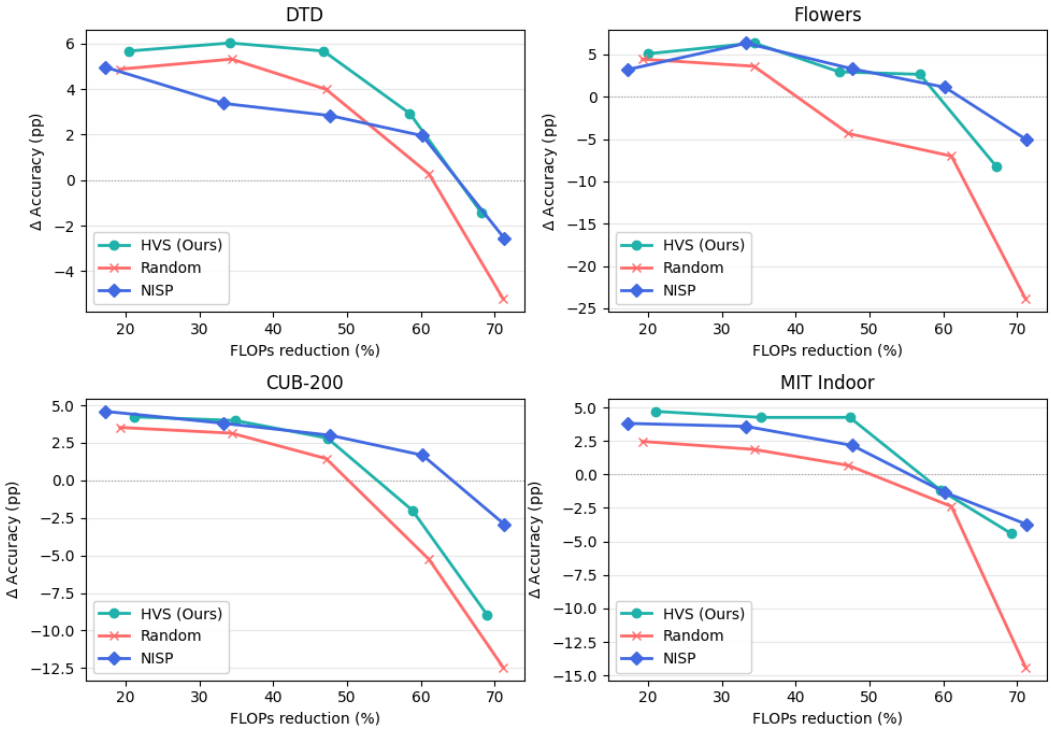


FIGURE 4.5: Change in accuracy versus FLOPs reduction (%) for HVS (ours), Random, and NISP on DTD, Flowers-102, CUB-200, and MIT Indoor.

TABLE 4.5: Regularisation effect of HVS pruning. r denotes the pruning ratio

Dataset	Method	r	Acc (%)	Params↓%	FLOPs↓%
DTD	L2-SP	–	60.37	0.00	0.00
	HVS	0.1	61.35	7.08	20.43
	HVS	0.2	61.70	10.88	34.19
	HVS	0.3	61.35	15.45	46.82
Flowers	L2-SP	–	81.67	0.00	0.00
	HVS	0.1	79.23	7.08	20.06
	HVS	0.2	80.50	11.99	34.37
	HVS	0.3	77.09	17.60	45.93
CUB-200	L2-SP	–	72.35	0.00	0.00
	HVS	0.1	70.88	4.97	21.24
	HVS	0.2	70.64	8.48	34.84
	HVS	0.3	69.47	12.31	47.34
MIT Indoor	L2-SP	–	70.37	0.00	0.00
	HVS	0.1	69.48	6.60	21.06
	HVS	0.2	69.03	11.41	35.28
	HVS	0.3	69.03	16.41	47.35

suggest that it has the most redundant features. Our pruning density shows a particular pattern as opposed to NISP where the pruning ratio is layer-wise, meaning that the layers are uniformly pruned . We argue that not all layers in pretrained network contributes equally to the transferred task. The effect of regularization and compressing the model, shows a when computational budgets are constrained.

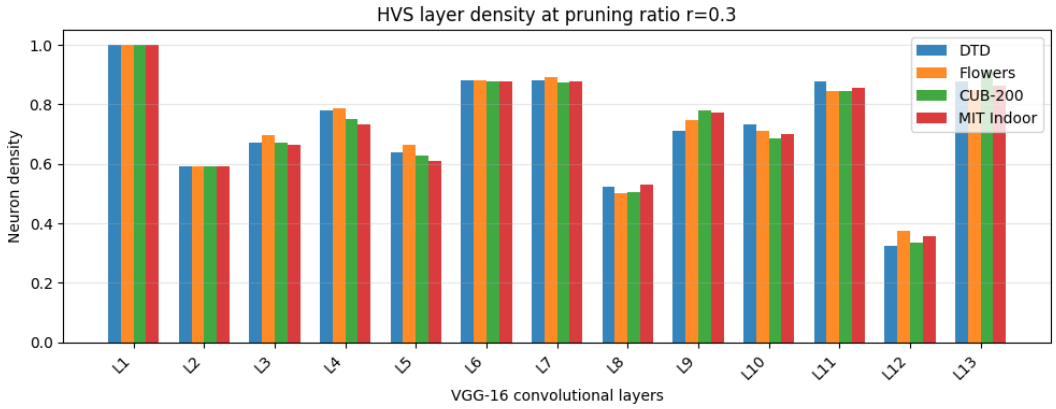


FIGURE 4.6: Layer-wise neuron retention for HVS pruning at ratio $r = 0.3$ on VGG-16 on datasets (DTD, Flowers102, CUB-200, MIT Indoor)

TABLE 4.6: VGG-16 on Flowers and the HVS-pruned model at pruning ratio $r = 0.3$.

layer type	$w_\ell \times h_\ell$	#Maps (orig.)	#Params (orig.)	FLOPs (orig.)	#Maps (HVS)	Density	Params↓	FLOPs↓
Conv_1	224×224	64	1.7×10^3	1.8×10^8	64	1.00	0%	0%
Conv_2	224×224	64	3.7×10^4	3.8×10^9	38	0.59	41%	41%
Conv_3	112×112	128	7.4×10^4	1.9×10^9	90	0.70	30%	30%
Conv_4	112×112	128	1.5×10^5	3.8×10^9	101	0.79	21%	21%
Conv_5	56×56	256	2.9×10^5	1.9×10^9	169	0.66	34%	34%
Conv_6	56×56	256	5.9×10^5	3.8×10^9	225	0.88	12%	12%
Conv_7	56×56	256	5.9×10^5	3.8×10^9	228	0.89	11%	11%
Conv_8	28×28	512	1.2×10^6	3.8×10^9	256	0.50	50%	50%
Conv_9	28×28	512	2.4×10^6	7.4×10^9	384	0.75	25%	25%
Conv_10	28×28	512	2.4×10^6	7.4×10^9	364	0.71	29%	29%
Conv_11	14×14	512	2.4×10^6	1.9×10^9	432	0.84	16%	16%
Conv_12	14×14	512	2.4×10^6	1.9×10^9	194	0.38	62%	62%
Conv_13	14×14	512	2.4×10^6	1.9×10^9	435	0.85	15%	15%
Linear	1×1	512	1.3×10^8	1.3×10^8	512	1.00	0%	0%
Total	–	–	1.3×10^8	3.1×10^8	–	0.73 (avg)	17.6%	45.9%

HVS provides a good accuracy, compression trade-off which is favorable when computational budgets are constrained. . However, it presents a limitation on fine-grained tasks where pruning is aggressive for this type of datasets. Addressing this limitation can be promising direction for future work.

4.6 Conclusion

In this chapter, we introduced a neuron importance method based on the *Heuristic for Variable Selection* (HVS) to analyze and compress pre-trained convolutional networks in transfer learning. Our approach evaluates neuron importance through a structured propagation method, accounting for each unit's global contribution to the final output, unlike magnitude-based methods that only consider local weight values.

Theoretical Contributions

We formalized the advantages of HVS-guided pruning through several key theoretical results:

- **Output Perturbation Bound:** We proved that the HVS score upper-bounds the perturbation in the network's output caused by removing a neuron, providing a theoretical guarantee for prediction stability post-pruning (Theorem 4.4).
- **Generalization Bound:** We established a generalization bound for the pruned network, showing that HVS-guided pruning reduces the complexity of the model while minimizing approximation error (Theorem 4.6). This analysis explains why moderate pruning (10–30%) can lead to better generalization, as observed empirically.
- **Connection to Taylor Sensitivity:** We showed that the HVS score relates to a first-order Taylor approximation of loss change, reinforcing its relevance for assessing neuron importance without requiring gradient computation (Proposition 4.9).
- **Convergence of Post-Pruning Fine-Tuning:** We proved that fine-tuning after pruning converges efficiently, with convergence rates directly tied to the HVS scores of removed neurons (Theorem 4.11).

Experimental Results

We evaluated the effectiveness of our method across diverse image classification benchmarks, including fine-grained recognition (Oxford Flowers 102, CUB-200), texture recognition (DTD), and medical imaging (Chest X-ray Pneumonia). The results show that:

- Moderate pruning (10–30%) consistently improves generalization across multiple datasets, with accuracy gains of up to $+8.52\%$ for DTD and $+5.80\%$ for Flowers-102.
- HVS-guided pruning removes up to 20% of parameters with negligible performance loss, even for on sensitive tasks like pneumonia detection or binary cat-dog classification.
- Propagated importance maps reveal that HVS identifies task-relevant regions (e.g., facial areas in animals), confirming that the method captures discriminative features rather than noise.

While our method achieves a strong balance between parameter reduction and performance, it currently operates at the level of individual neurons. Future work could explore more **channel pruning** as a complementary strategy to directly reduce the width of convolutional layers, further optimizing memory and computational efficiency. Additionally, extending the theoretical analysis to more complex architectures (e.g., transformers) or multimodal tasks presents a promising direction for broader applicability.

Chapter 5

Tensor decomposition and sparsity Compression for 3D segmentation

Contents

That the knowledge at which geometry aims is knowledge of the eternal, and not of aught perishing and transient.

Plato

5.1 Introduction

Deep learning networks have shown great performance on computer vision. In particular, medical image segmentation problem due to their ability to learn complex patterns in anatomical structures. CNN segmentation has achieved excellent results compared to classical methods and expert manual segmentation in detecting tumors in MRI, and segmenting pathological lesions Pereira et al. (2016), Caicedo et al. (2019).

While most current medical tasks relies on 2D images and slices, incorporating three dimensional anatomical context gives doctors a complete information than individual slices. 3D segmentation leverages the inter-slice spatial relationship, allowing a better boundary detection of complex structures like organs, blood vessels. Studies show an accuracy improvement compared to 2D methods Zhang et al. (2022).

However, using 3D deep learning models faces significant challenges in clinical settings. Beside the heterogeneity and dynamism of medical data, the memory requirements for 3D images can be 20 times higher than 2D slices. And given the size of the deep learning model itself, most hospitals environment lacks the sufficient resources, such as memory, computing, and energy Rieke et al. (2020), which must be used efficiently, while providing high accuracy. Thus the need of compressed models that can be stored in memory and used to make an inference efficiently while maintaining similar performance.

Consequently, tensor decomposition has emerged as a way to compress large scale deep learning models into lower-rank tensor components while keeping their performance. The most used methods in CNN are CANDECOMP/PARAFAC (CP) and tucker decomposition.

In this study, we propose combining weight decomposition and channel pruning. Low rank decomposition reduces the dimensions of weight tensors, which reduces floating point operations (FLOPs) and parameter count, while preserving the overall structure. and since convolutional layers contain many redundant filters, they can be safely removed.

5.2 Related work

Tensor decomposition has emerged as a principled approach for reducing the size and computational cost of deep neural networks by exploiting low rank structure in their weight tensors. Instead of storing full, dense convolutional kernels, these methods approximate the original weights with products of smaller tensors, thereby lowering the number of parameters and floating point operations while aiming to preserve the network's representational capacity. A recent survey shows that tensor decompositions can substantially reduce model size, runtime, and energy consumption across CNNs, RNNs, and Transformers, and are particularly attractive for deployment on edge devices and resource constrained hardware Liu and Parhi

(2023). Recent advances in tensor decomposition for CNN-based 3D segmentation have shown remarkable promise for addressing the ever-increasing computational demands of modern deep neural networks. A variety of tensor formats have been explored in the context of CNN compression, including CANDECOMP/PARAFAC (CP), Tucker, tensor train (TT), tensor ring (TR), and hierarchical Tucker (HT). CP and Tucker decompositions are among the earliest and most widely adopted choices for convolutional layers. CP decomposes a kernel into a sum of rank one components, achieving aggressive compression at the cost of greater sensitivity to rank selection, whereas Tucker factorization uses a small core tensor and factor matrices along each mode, typically providing a more stable trade off between compression and accuracy. More recent work has introduced hierarchical formats such as HT and HT-2 Gabor and Zdunek (2023) to further reduce parameters in deep CNNs. these methods replace a standard convolutional kernel with a sequence of lower dimensional kernels. This process not only compresses the number of parameters and required floating point operations, but also preserves essential spatial information necessary for accurate 3D segmentation.

Beyond 2D tasks, tensor decomposition has also been investigated for high dimensional data and spatio temporal architectures. For example, hierarchical Tucker decomposition has been integrated into hybrid 3D CNN-LSTM models to reduce the complexity of both convolutional and recurrent components while maintaining accuracy on video data Anoopa et al. (2025). In hyperspectral image analysis and remote sensing, tensorized networks using TT or related formats have been proposed to exploit the multiway structure of the data and to compress both fully connected and convolutional layers Li et al. (2023). These works consistently report that low rank tensorization can serve as a replacement for standard layers, enabling memory and computation savings.

Similar to our work, several studies have applied tensor decomposition to medical imaging models and 3D segmentation networks. Post training Tucker factorization has been used to compress large 3D segmentation models such as nnU Net based TotalSegmentator, reducing FLOPs and memory by up to around 80% while preserving segmentation accuracy after fine tuning Weber et al. (2025). Similar pipelines have been explored for object detection and segmentation in remote sensing, where a ResNet detector is first trained and then decomposed using Tucker, achieving speedups with only modest drops in mean average precision Huyan et al. (2023). These results demonstrate that decomposition can be a practical, post hoc tool to adapt large models to constrained settings.

Despite these gains , decompositions alone do not address which feature channels are truly important, since they treat all channels equally. In practice, most applications of Tucker to segmentation have been applied after full training of pretrained models with ranks chosen by heuristics or global criteria. In this setting, all channels within a layer are compressed essentially the same way, and the method does not explicitly identify which channels or factors are most important for the downstream task. As a result, decomposition alone often leaves significant filter level redundancy.

In parallel, pruning is another widely used strategy for compressing CNNs. In the context of 3D medical image segmentation, pruning aims to selectively remove redundant or less informative parameters, typically at the level of filters or channels, in order to shrink model size and accelerate inference without significantly degrading performance. Most pruning work in 3D medical segmentation builds on U-Net style architectures, particularly nn-UNet, which remains a strong baseline across many datasets. Several studies have demonstrated that these models are

heavily over parameterized and can be compressed substantially after training. A study on pruning U Nets for radiotherapy planning, where simultaneous training and pruning (STAMP) achieved sparsities on the order of 80% while preserving or even slightly improving segmentation performance in low data regimes Dinsdale et al. (2022a). Pruning has also been integrated into more specialized or efficient 3D segmentation frameworks. For example, hybrid pruned nnU Net variants prune channels or filters in a structured way to accelerate brain tumor segmentation while retaining robust performance Yang et al. (2025). Other works apply layer wise or branch wise pruning to U-Net++ or related encoder–decoder backbones, focusing on removing redundant sub networks or skip connections to reduce parameters and inference time for tasks such as chest imaging Xiao et al. (2025). In parallel, broader pruning strategies from the general CNN literature such as channel pruning, block wise pruning, and progressive pruning have started to be adapted to 3D segmentation, often in combination with dynamic inference. The most commonly used pruning criterion in these settings is still magnitude based filters or weights with small l_1 or l_2 norms are removed under the assumption that they contribute least to the output. This strategy is simple, easy to integrate into existing pipelines including nnU-Net Li et al. (2025),Yang et al. (2025).

However, magnitude based criteria operate directly on raw weights and ignore how filters interact with each other and with the data distribution. Filters with small norms may still be crucial for segmenting rare or fine structures, which are common in 3D organ and lesion segmentation, while some large norm filters may be redundant or highly correlated with others. Additionaly, Pruning on 3D CNNs has received considerably less attention than its 2D counterpart, largely because the higher dimensional weight tensors make both the design of pruning criteria and the fine tuning procedure more expensive. Also and direct transfer of these techniques to 3D segmentation architectures such as 3D U-Net or nnU-Net does not always yield consistent speedups on real hardware due to memory access patterns and 3D convolution cost.

These gaps are particularly relevant in 3D medical segmentation, where models must balance high accuracy on small datasets with strict constraints on memory, latency, and energy in clinical deployment. Our work addresses these gaps by proposing a two stage compression pipeline specifically designed for 3D convolutional neural networks used in medical image segmentation. Instead of pruning directly on the original full rank convolutional kernels, we first apply Tucker decomposition to rewrite each 3D convolution as a compact sequence of factor matrices and a core tensor, capturing the dominant low rank structure of the layer. This already reduces parameters and FLOPs and exposes a factorized representation in which correlations between channels and spatial modes are more explicit. we then perform structured pruning driven by an importance score computed in the decomposed space.

5.3 Proposed method

5.3.0.1 Tensor decomposition

In order to reduce the model size, first we apply tensor decomposition on the convolution filters. For each layer Therefore, instead of operating a convolutional operation on a large tensor, three simple convolutions are then applied.

Given a convolutional network with weight kernel $\mathcal{W}$ of dimensions $\mathcal{H} \times \mathcal{L} \times \mathcal{W} \times \mathcal{C}$ with H, L, W, C are the height, length, width and channel number respectively, the

tucker decomposing embeds the filter $\mathbf{W}$ into a lower dimension $\ll \mathcal{C}$. a core kernel with dimension $\mathcal{K} \times \mathcal{K} \times \mathcal{K}$. Given 3D convolutional layer with weight tensor $\mathcal{W} \in \mathbb{R}^{C_{in} \times C_{out} \times D_k \times H_k \times W_k}$, where C_{in} and C_{out} represent input and output channels dimensions, and $D_k \times H_k \times W_k$ denotes the kernel size, and an input tensor $\mathcal{X} \in \mathbb{R}^{I \times C_{in} \times D \times H \times W}$ with spatial dimensions $D \times H \times W$, the standard 3D convolution is computed as:

$$Y(c_{out}, d', h', w') = \sum_{c_{in}=1}^{C_{in}} \sum_{i=1}^{D_k} \sum_{j=1}^{H_k} \sum_{k=1}^{W_k} W(c_{in}, c_{out}, i, j, k) \cdot X(c_{in}, d_i, h_j, w_k)$$

where $d_i = d' \cdot s_d + i - p_d$, $h_j = h' \cdot s_h + j - p_h$, $w_k = w' \cdot s_w + k - p_w$ for stride s and padding p of each dimension.

We perform the Tucker decomposition on the weight tensor along the channel dimensions, preserving the spatial structure of the kernel as they are generally small. therefore any further decomposition would not result in significant parameter reduction or performance gain.

The decomposed tensor is expressed as

$$W \approx G \times_1 U_{in} \times_2 U_{out}$$

where:

$\mathcal{G} \in \mathbb{R}^{R_{in} \times R_{out} \times k_d \times k_h \times k_w}$ is the core tensor containing compressed spatial-temporal filters, $\mathbf{U}_{in} \in \mathbb{R}^{C_{in} \times R_{in}}$ is the input channel factor matrix, and $\mathbf{U}_{out} \in \mathbb{R}^{C_{out} \times R_{out}}$ is the output channel factor matrix. The notation $\times_n$ denotes the mode-n product, which multiplies the core tensor with the factor matrix along the n-th dimension. The factorization of the decomposed tensor is expressed as follows

$$W_{c_{in}, c_{out}, i, j, k} = \sum_{r_{in}=1}^{R_{in}} \sum_{r_{out}=1}^{R_{out}} G_{r_{in}, r_{out}, i, j, k} \cdot U_{in}(c_{in}, r_{in}) \cdot U_{out}(c_{out}, r_{out})$$

Therefore, the convolution operation is computed as

$$Y(c_{out}, d', h', w') = \sum_{r_{out}=1}^{R_{out}} U_{out}(c_{out}, r_{out}) \sum_{r_{in}=1}^{R_{in}} \sum_{i=1}^{D_k} \sum_{j=1}^{H_k} \sum_{k=1}^{W_k} G(r_{in}, r_{out}, i, j, k) \times \left(\sum_{c_{in}=1}^{C_{in}} U_{in}(c_{in}, r_{in})\, X(c_{in}, d_i, h_j, w_k) \right).$$

5.3.0.2 Pruning

The importance metric for the pruning method is HVS, defined in chapter 4 , which computes the contribution of each neuron from the last layer back-propagated to the input matrix. Given an activation matrix in a layer ℓ. (to add from previous chapter)

We compute the importance channel-wise by averaging the importance metric of each element in the activation matrix of a layer computed by HVS. A convolutional layer produces an activation tensor $\mathbf{A}^{\ell} \in \mathbb{R}^{C^{\ell}_{out} \times D \times H \times W}$, where C^{ℓ}_{out} denotes the number of output channels, and D, H, W are the spatial dimensions of the feature maps. For output channel c in layer ℓ, the channel importance is defined as

$$\mathrm{HVS}_{\text{channel}}(c,\ell) = \sqrt{\frac{1}{D \cdot H \cdot W} \sum_{d=1}^{D} \sum_{h=1}^{H} \sum_{w=1}^{W} \left(\mathrm{HVS}^{\ell}_{c,d,h,w}\right)^2}$$

We normalize each layer's importance as follows

$$\tilde{\mathrm{HVS}}^{\ell}(c) = \frac{\mathrm{HVS}_{\text{channel}}(c,\ell)}{\left\|\mathbf{HVS}^{\ell}_{\text{channel}}\right\|_2}$$

After Tucker decomposition, we compute the channel wise importance mask given by

$$m^{\ell}_{c} = \begin{cases} 1 & \text{if } \tilde{\mathrm{HVS}}^{\ell}(c) > \tau_{\ell} \\ 0 & \text{otherwise} \end{cases}$$

The threshold τ_ℓ is chosen such that:

$$\sum_{c=1}^{C^{\ell}_{out}} m^{\ell}_{c} = \left\lceil (1-\rho) \cdot C^{\ell}_{out} \right\rceil$$

, where ρ is the percentage of channels to be removed in each layer.

The channel pruning removes a subset of output channels, the core tensor and input projection matrix remain unchanged, preserving the learned low-rank structure from the tensor decomposition. Therefore, the modified convolution is written as follows

$$\mathbf{Y}^{\ell} = \mathbf{m}^{\ell} \odot \left[\mathbf{U}^{\ell}_{\text{out}} \left(\mathcal{G}^{\ell} \times_1 \left(\mathbf{U}^{\ell\top}_{\text{in}} \mathbf{X}^{\ell}\right)\right)\right],$$

Equivalely,

$$Y(c'_{out}, d', h', w') = \sum_{r_{out}=1}^{R_{out}} U'_{out}(c'_{out}, r_{out}) \sum_{r_{in}=1}^{R_{in}} \sum_{i=1}^{D_k} \sum_{j=1}^{H_k} \sum_{k=1}^{W_k} G(r_{in}, r_{out}, i, j, k) \times \left(\sum_{c_{in}=1}^{C_{in}} U_{in}(c_{in}, r_{in})\, X(c_{in}, d_i, h_j, w_k) \right),$$

where c'_{out} ranges over the set of retained output channels.

5.4 Theoretical Analysis

5.4.1 Tucker Approximation Error Bound

Theorem 5.1 (Tucker Approximation Error) *Let $\mathcal{W} \in^{C_{\text{in}} \times C_{\text{out}} \times D_k \times H_k \times W_k}$, and let $\hat{\mathcal{W}}$ be its best Tucker approximation with channel ranks $(R_{\text{in}}, R_{\text{out}})$ obtained by HOSVD. Then*

$$\mathcal{W} - \hat{\mathcal{W}}^2 \;\le\; \sum_{r=R_{\text{in}}+1}^{C_{\text{in}}} \sigma^{(1)}_r(\mathcal{W})^2 + \sum_{r=R_{\text{out}}+1}^{C_{\text{out}}} \sigma^{(2)}_r(\mathcal{W})^2, \tag{5.1}$$

where $\sigma^{(n)}_r(\mathcal{W})$ denotes the r-th singular value of the mode-n unfolding $\mathcal{W}_{(n)}$.

Algorithm 2: Tucker Decomposition and Pruning compression

Require: Pretrained network $\{$, pruning ratio ρ, rank ratios $r_{\text{in}}, r_{\text{out}}$
  for each conv layer ℓ **do**
    Decompose $\mathcal{W}^\ell$ with ranks $(R^\ell_{\text{in}}, R^\ell_{\text{out}})$
    Replace layer with three blocks: $\mathbf{U}^\ell_{\text{in}} \to \mathcal{G}^\ell \to \mathbf{U}^\ell_{\text{out}}$
  end for
  for all layers
  for each layer ℓ **do**
    Compute HVS^ℓ_j
  end for
  normalize: $\tilde{\text{HVS}}^\ell(c)$
  Rank channels: select top-$(1-\rho) \times C^\ell_{\text{out}}$
  Prune $\mathbf{U}^\ell_{\text{out}}$: retain rows for selected channels $\to \mathbf{U}^{\ell,\text{pruned}}_{\text{out}}$
  Fine-tune compressed network $\{'$
  return Compressed network $\{'$

Intuition Behind Theorem 5.1: The quality of Tucker compression can be predicted before compressing: it suffices to inspect the singular value spectrum of the weight tensor unfoldings. If the singular values decay rapidly, most of the information is captured by a small number of components, and compression will be nearly lossless. This theorem provides a principled, data-driven criterion for choosing the downsampling factor DF instead of relying on grid search.

Proof **Step 1 – Mode-n unfolding.** By definition of the Tucker decomposition, $\hat{\mathcal{W}}_{(n)} = U^{(n)}U^{(n)\top}\mathcal{W}_{(n)}$, where $U^{(n)} \in^{C_n \times R_n}$ contains the top R_n left singular vectors of $\mathcal{W}_{(n)}$.

Step 2 – Projection error. For any matrix M, the best rank-R approximation gives $\|M - U_R U_R^\top M\|_F^2 = \sum_{r=R+1}^{\min(m,n)} \sigma_r(M)^2$ (Eckart–Young–Mirsky theorem). Applying this to $\mathcal{W}_{(1)}$ and $\mathcal{W}_{(2)}$,

$$\|\mathcal{W}_{(1)} - U^{(1)}U^{(1)\top}\mathcal{W}_{(1)}\|_F^2 = \sum_{r>R_{\text{in}}} \sigma_r^{(1)}(\mathcal{W})^2, \tag{5.2}$$

$$\|\mathcal{W}_{(2)} - U^{(2)}U^{(2)\top}\mathcal{W}_{(2)}\|_F^2 = \sum_{r>R_{\text{out}}} \sigma_r^{(2)}(\mathcal{W})^2. \tag{5.3}$$

Step 3 – Tensor norm equivalence. The Frobenius norm of a tensor equals the Frobenius norm of any of its unfoldings: $\mathcal{T}^2 = {\mathcal{T}_{(n)}}^2$. Since the modes are compressed sequentially,

$$\mathcal{W} - \hat{\mathcal{W}}^2 \leq \sum_{n\in\{1,2\}} {\mathcal{W}_{(n)} - U^{(n)}U^{(n)\top}\mathcal{W}_{(n)}}^2, \tag{5.4}$$

which gives (5.1). ■

Remark 5.2 *Theorem 5.1 provides an interpretable compression criterion: the downsampling factor* DF *should be chosen so that the discarded singular values are small, i.e.,* DF *is chosen at the* spectral knee *of $\mathcal{W}_{(1)}$ and $\mathcal{W}_{(2)}$.*

5.4.2 Output Error Propagation Through Layers

Theorem 5.3 (Error Propagation Through a Compressed Network) *Let f be an L-layer 3D U-Net-like network. Let $\hat{f}$ be the network obtained by Tucker-compressing every convolutional layer with approximation errors $\varepsilon_\ell = \mathcal{W}^\ell - \hat{\mathcal{W}}^\ell$ (bounded by Theorem 5.1). Then for any input volume X,*

$$\|f(X) - \hat{f}(X)\|_2 \;\le\; B^{L-1}\|X\|_2 \sum_{\ell=1}^{L} \frac{\varepsilon_\ell}{B^\ell}, \tag{5.5}$$

where $B = \max_\ell \|\mathcal{W}^\ell\|_F$ bounds the operator norm.

Intuition Behind Theorem 5.3: Compression errors in a deep network are not independent: an error introduced in an early layer is carried forward and amplified by every subsequent layer, whereas an error in the last layer has no layers left to amplify it. This means early layers should be compressed more conservatively (higher DF) than late layers, a principle that could guide a layer-adaptive compression strategy in future work.

Proof Step 1 – Single layer error. Let h^ℓ and $\hat{h}^\ell$ denote the activations of the original and the compressed networks. For layer ℓ,

$$\|h^{\ell+1} - \hat{h}^{\ell+1}\|_2 \le \|\mathcal{W}^\ell - \hat{\mathcal{W}}^\ell\|_F \|h^\ell\|_2 + \|\hat{\mathcal{W}}^\ell\|_F \|h^\ell - \hat{h}^\ell\|_2 \le \varepsilon_\ell \|h^\ell\|_2 + B\|h^\ell - \hat{h}^\ell\|_2. \tag{5.6}$$

Step 2 – Induction. Let $\delta_\ell = \|h^\ell - \hat{h}^\ell\|_2$ with $\delta_0 = 0$. From (5.6): $\delta_{\ell+1} \le B\,\delta_\ell + \varepsilon_\ell \|h^\ell\|_2$. Since $\|h^\ell\|_2 \le B^\ell \|X\|_2$, unrolling the recursion gives

$$\delta_L \le \sum_{\ell=1}^{L} B^{L-\ell}\,\varepsilon_\ell\, B^{\ell-1}\|X\|_2 = B^{L-1}\|X\|_2 \sum_{\ell=1}^{L} \varepsilon_\ell. \tag{5.7}$$

Dividing by B^{L-1} and reindexing gives (5.5). ■

Remark 5.4 *The bound (5.5) shows that compression errors* accumulate linearly *across layers. Early layers (small ℓ) should therefore be compressed more conservatively (higher* DF*) than late layers, as their errors are amplified by $B^{L-\ell}$. This suggests an* adaptive *choice of ranks per layer, which we leave for future work.*

5.4.3 Tucker Decomposition as Implicit Regularisation

Proposition 5.5 (Tucker Compression Induces Nuclear Norm Regularisation) *Let $\hat{W}^{(\ell)} \in^{C_{\text{in}} \times C_{\text{out}}}$ denote the unfolded weight matrix of layer ℓ after Tucker compression with ranks $(R_{\text{in}}, R_{\text{out}})$. Then $\hat{W}^{(\ell)}$ is the solution to the constrained problem*

$$\min_{W:\,(W) \le R} \|W - W_0\|_F^2, \tag{5.8}$$

which, via the convex relaxation of the rank constraint, is equivalent to adding a nuclear norm penalty $\lambda_\ell \|W\|_*$ *to the training loss, where λ_ℓ is determined by the threshold $\sigma_{R+1}(W_0)$.*

Intuition Behind Proposition 5.5: Tucker decomposition is not merely a mechanical size reduction: it implicitly imposes a low-rank prior on the weight matrices, equivalent to adding a nuclear norm regularisation term to the training loss. This regularisation reduces overfitting to the training distribution, which explains the seemingly counter-intuitive result of Table 5.1: a network compressed to 10% of its original size achieves *better* segmentation accuracy than the uncompressed model.

Proof The solution to (5.8) is given by the truncated SVD of W_0: $\hat{W} = \sum_{r=1}^{R} \sigma_r u_r v_r^\top$, which corresponds to hard thresholding of singular values. By Candès and Recht (2008), the convex relaxation of rank-R approximation under a Frobenius loss is the nuclear norm minimisation

$$\min_{W} \|W - W_0\|_F^2 + 2\lambda \|W\|_*, \tag{5.9}$$

whose solution is the singular value soft thresholding operator with threshold λ. Setting $\lambda = \sigma_{R+1}(W_0)/2$ recovers the hard thresholding solution, establishing the equivalence. ■

5.4.4 Combined Compression Bound

Theorem 5.6 (Combined Tucker + HVS Compression Bound) *Let $\hat{f}_\rho$ be the network obtained by Tucker-compressing all layers and then pruning fraction ρ of channels per layer by HVS. Under the assumptions of Theorems 5.3 and 4.6, the generalisation error on the target task satisfies*

$$\mathcal{L}(\hat{f}_\rho) \leq \hat{\mathcal{L}}(\hat{f}_\rho) + \underbrace{\mathcal{O}\left(\sqrt{\frac{(1-\rho)\sum_\ell R_{\text{in}}^{(\ell)} R_{\text{out}}^{(\ell)} D_k H_k W_k}{N}}\right)}_{\text{complexity after Tucker + pruning}} + \underbrace{B^{L-1}\|X\|_2 \sum_\ell \varepsilon_\ell}_{\text{Tucker approx. error}} + \underbrace{G\, B^L \bar{h} \sum_{j \in \mathcal{P}} S_j}_{\text{HVS pruning error}}, \tag{5.10}$$

where $R_n^{(\ell)} = \lfloor \text{DF} \cdot C_n^{(\ell)} \rfloor$.

Intuition. The total compression error decomposes into two independent and controllable parts: the Tucker approximation error (controled by DF) and the HVS channel pruning error (controled by ρ). Because the two terms are additive, one can optimise DF and ρ separately. The bound also explains the performance collapse observed in Table 5.3 for DF$= 0.3$, $\rho = 0.4$: when Tucker is already aggressive, the Tucker error term is large, and additional pruning pushes the total error beyond the from scratch baseline.

Proof The result follows directly from combining the three individual bounds:

1. **Complexity term**: the number of free parameters after Tucker decomposition and HVS channel pruning is $(1-\rho)\sum_\ell R_{\text{in}}^{(\ell)} R_{\text{out}}^{(\ell)} D_k H_k W_k$; substituting into the Rademacher complexity bound (4.8) gives the first term.
2. **Tucker error**: from Theorems 5.1 and 5.3, the output error is $B^{L-1}\|X\|_2 \sum_\ell \varepsilon_\ell$ where each ε_ℓ is bounded by discarded singular values.

3. **HVS pruning error**: from Theorem 4.6, the approximation error from channel removal is $G\,B^{L}\bar{h}\sum_{j\in\mathcal{P}} S_j$.

Summing the three terms yields (5.10). ■

Corollary 5.7 (Optimal Decomposition-Pruning Trade-off) *The bound in* (5.10) *reveals an explicit* trade-off *between compression and accuracy:*

- *Decreasing* DF *(using more aggressive Tucker compression) reduces the complexity term but increases the Tucker approximation error.*
- *Increasing* ρ *(using more aggressive HVS pruning) further reduces the complexity term but increases the HVS error.*
- *The bound is minimised jointly in* (DF, ρ)*, which provides a principled criterion for hyperparameter selection.*

In particular, the bound predicts that the optimal ρ grows *with* DF*: when Tucker compression is already aggressive, additional pruning is safer because many channels in the core tensor are already redundant.*

Intuition Behind Corollary 5.7: Tucker compression and HVS channel pruning are complementary, not competing. Tucker reduces the spectral dimensionality of the filters, and HVS then removes the channels that remain redundant in the compressed space. The corollary predicts that the optimal pruning ratio increases as Tucker becomes more aggressive consistent with Table 5.2, where $\rho = 0.5$ is optimal across all tested values of DF, including the most aggressive ones.

5.4.5 Convergence of Post-Compression Fine-Tuning

Proposition 5.8 (Fine-Tuning Convergence for Compressed 3D Models) *Under Assumption 4.3, the post-compression fine-tuning with SGD converges to a stationary point at rate*

$$\frac{1}{T}\sum_{t=0}^{T-1}\left[\nabla\mathcal{L}(\theta_t)^2\right] \leq \frac{2\Delta_0}{\eta\sqrt{T}} + \frac{L\eta\sigma^2}{\sqrt{T}}, \tag{5.11}$$

where $\Delta_0 = \mathcal{L}(\theta_{\text{compressed}}) - \mathcal{L}(\theta^*)$ *is bounded by the right-hand side of* (5.10)*. Consequently, a smaller* DF *and smaller* ρ *(within the regime where the complexity term dominates) leads to faster fine-tuning convergence.*

Intuition Behind Proposition 5.8: A well-chosen compression (small DF and ρ within the safe regime) leaves the network weights close to their precompression values, so fine-tuning starts near a good solution and converges quickly. This is why 200 fine-tuning updates are sufficient in our experiments for the configurations reported in Table 5.2. Configurations with a large initial gap caused by overly aggressive pruning would require significantly more updates to recover, explaining the collapses in Table 5.3.

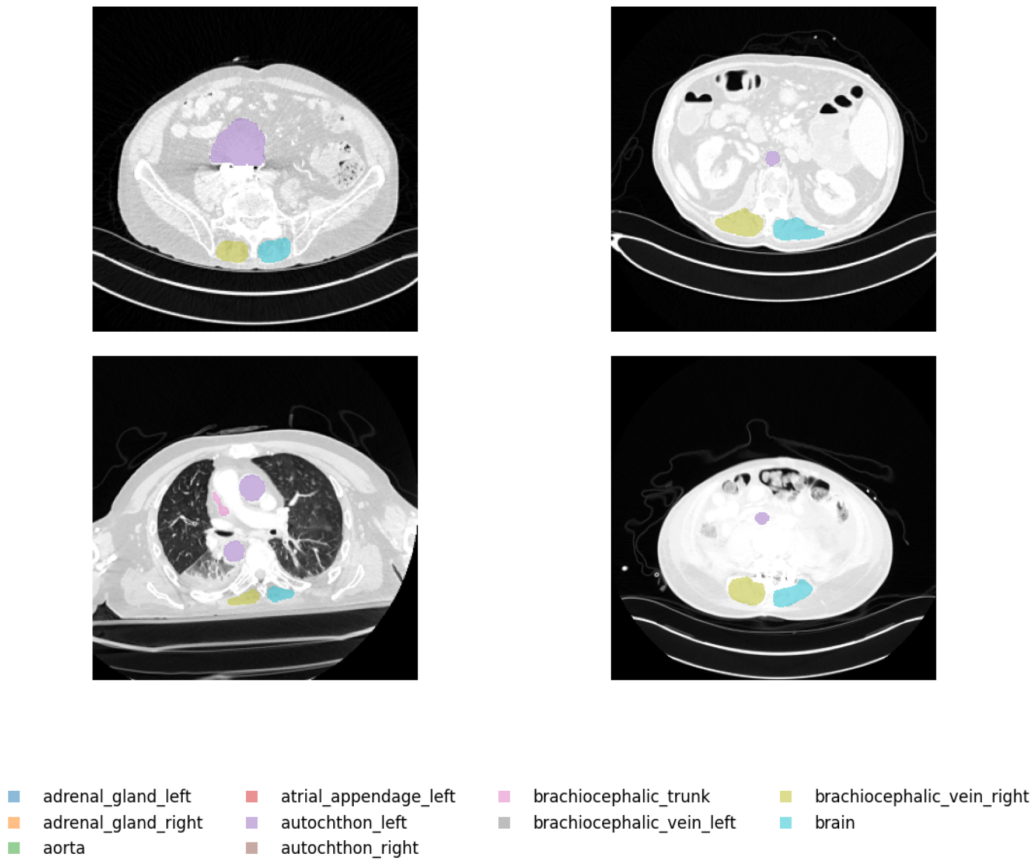


FIGURE 5.1: axial CT slices from four TotalSegmentator cases with multi-organ segmentations.

5.5 Experimental Results

5.5.1 Dataset

In order to evaluate the proposed approach for segmentation, We conducted experiments on TotalSegmentator (TS) dataset Wasserthal (2023) of CT images with annotations for multiple anatomical structures. The dataset contains 1228 CT images ,randomly sampled from clinical practice from multiple institutions. The dataset comprises 117 anatomical structures including 27 organs (liver, kidneys, spleen, heart, lungs), 59 bones (vertebrae, ribs, skull, long bones), 10 muscles (gluteus maximus, iliopsoas, autochthonous muscles), and 8 major vessels (aorta, vena cava, portal vein). For testing, a subset of 89 CT is used. Example axial CT slices from four TotalSegmentator cases with multi-organ segmentations are illustrated in figure 5.1.

5.5.2 experiment settings

In this study, we use the package TotalSegmentator Wasserthal et al. (2023). the package contains a model of nnU-net architecture trained on the TS dataset. We conducted our experiments on the 3mm resolution.

The default model is a 3D U Net provided by the TS package. The Adam optimiser with learning rate $10^{-4}$is used for 200 updates. Compression is applied after training. Each convolutional layer is factorized using Tucker decomposition with a specified downsampling factor (DF). The DF rescales the number of input/output

channels in the core tensor. For HVS based importance measure the pruning ratio varies in $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ which controls how many channels are removed. We retain the channels with highest HVS scores.

For evaluation, we used two segmentation metrics. The dice similarity coefficient (DICE), which measures volumetric overlap between predicted and ground truth segmentations, and the normalized Surface Dice (NSD), which evaluates boundary agreement within a tolerance distance, therefore emphasizing clinically relevant contour accuracy rather than overlap. We also measure the compression ratio (CR) and the percentage of parameters removed compared to the baseline.

5.5.3 Results

The table 5.1 summarises the effect of varying the downsampling factor (DF) used in Tucker decomposition of the original model without any channel pruning. The lower DF values produce a smaller core tensor and therefore fewer parameters. In our experiments, decreasing DF from 0.9 to 0.05 produced a monotonically increasing compression ratio (CR) and correspondingly parameter reduction. For instance, $DF = 0.9$ yields a CR of roughly 1.1× approximately 10 % of parameter reduction, whereas $DF = 0.3$ yields a CR of 5.8× approximately 83 % of reduction. The segmentation performance, measured by average Dice and normalized surface dice (NSD), improved at higher compression levels. Dice score increased from +23% at $DF = 0.9$ to +30 % at $DF = 0.05$. Our results thus demonstrate that Tucker downsampling alone can substantially compress the model while slightly enhancing segmentation accuracy. this result may suggest that the tensor factorization acted as an implicit regularization.

While down sampling alone already yields performance improvements, further model reduction can be achieved by pruning a fraction of the core tensor's channels. Table 5.2 lists configurations that achieved good trade-offs between compression and accuracy. For each DF, we chose the pruning ratio that provided strong compression with a slight Dice and NSD drop of the decomposition only baseline. For example, $DF = 0.9$ combined with a prune ratio of 40 % resulted in a CR of 1.7× approximately 43 % parameter reduction) and still improved the score by about +25 %. More aggressive decompositions ($DF = 0.3$ or 0.1) with prune ratios of 50 % achieved CRs of 8× to 13 yet retained gains of +25 pp in both metrics. The fact that optimal pruning ratios remained high across all DF values shows that the HVS pruning effectively removed redundant channels without sacrificing accuracy.

Table 5.3 and the figure 5.2 illustrate how pruning affects models that are already compressed via Tucker decomposition. At each DF (0.1, 0.3, 0.5 and 0.7), the first row corresponds to the decomposition only baseline (0 % pruning). Subsequent rows show additional parameter reduction achieved by pruning and the corresponding change in Dice and NSD relative to the baseline of that DF. We observed that moderate pruning is safe. At $DF = 0.5$ and $DF = 0.7$, pruning 10 to 30 % of the channels removes up to 23 % of the remaining parameters but causes only a -3 pp drop in score. This shows that some channels in the decomposed tensor are redundant. Also, aggressive pruning can trigger collapse. For $DF = 0.3$, and pruning 40% of channels leads to severe performance collapse. Figure5.3 plots dice improvement against the pruning ratio for multiple DF values. The results show that moderate pruning often yields the highest performance improvements. This shows that the a trade-off between DF and pruning must be tuned.

TABLE 5.1: Effect of Tucker decomposition on 3D segmentation model compression at different downsampling factors (DF). All configurations use prune ratio $p = 0$. Δ Dice and Δ NSD report the change in percentage points relative to the original TS model.

DF	CR	Param Red. (%)	Δ Dice (pp)	Δ NSD (pp)
Baseline	1.0	0.0	—	—
0.90	1.1	10.6	+23.61	+23.62
0.70	1.7	42.2	+25.22	+24.95
0.50	3.0	66.5	+28.78	+28.34
0.40	4.1	75.3	+28.78	+28.35
0.30	5.8	82.8	+27.56	+27.15
0.20	8.2	87.7	+27.54	+27.11
0.10	10.9	90.8	+30.03	+29.60
0.05	11.6	91.4	+30.03	+29.60

TABLE 5.2: Compression results on the original TS model.

DF	Prune Ratio	CR	Param Red. (%)	Δ Dice (%)	Δ NSD (%)
0.90	0.4	1.7	42.6	+25.08	+24.59
0.70	0.5	3.0	66.9	+25.03	+24.60
0.50	0.5	4.8	79.2	+25.01	+24.55
0.40	0.5	6.2	83.8	+25.02	+24.55
0.30	0.5	8.1	87.7	+25.02	+24.54
0.20	0.5	10.5	90.5	+25.05	+24.54
0.10	0.5	12.7	92.2	+25.05	+24.54
0.05	0.5	13.2	92.4	+25.06	+24.54

5.6 Conclusion

In this chapter, we proposed combining Tucker decomposition with HVS-based channel pruning for 3D medical image segmentation, and provided a rigorous theoretical analysis. The main theoretical contributions are:

- **Tucker approximation error bound** (Theorem 5.1): the Frobenius norm error is bounded by discarded singular values, providing a principled rank selection criterion.
- **Error propagation through layers** Theorem 5.3): compression errors accumulate linearly accoss layers, which motivates conservative compression in the early layers.
- **Nuclear norm regularisation** (Proposition 5.5): Tucker decomposition implicitly act as a nuclear norm regulariser, which explains the empirical Dice improvement.
- **Combined compression bound** (Theorem 5.6): a joint error bound for Tucker and HVS pruning, leading to an explicit trade-off criterion between DF and ρ (Corollary 5.7).
- **Post-compression convergence** (Proposition 5.8): fine-tuning converges faster when the initialisation gap is small, as ensured by choosing DF and ρ in the regime where the nuclear norm regularisation dominates.

TABLE 5.3: Pruning effect. Δ Dice and Δ NSD are relative to the decomposition-only baseline ($p = 0$) at each DF. Additional Param Red. is the percentage of parameters further removed by pruning beyond Tucker decomposition.

	DF = 0.1			DF = 0.3			DF = 0.5			DF = 0.7		
Prune Ratio	Add. Param Red. (%)	Δ Dice (pp)	Δ NSD (pp)	Add. Param Red. (%)	Δ Dice (pp)	Δ NSD (pp)	Add. Param Red. (%)	Δ Dice (pp)	Δ NSD (pp)	Add. Param Red. (%)	Δ Dice (pp)	Δ NSD (pp)
0.0	0.0	—	—	0.0	—	—	0.0	—	—	0.0	—	—
0.1	3.5	-5.0	-5.0	6.3	-2.5	-2.5	7.5	-3.7	-3.7	8.8	-0.1	-0.3
0.2	7.1	-4.8	-5.0	12.4	-4.9	-5.1	15.1	-3.7	-3.8	16.8	-0.1	-0.3
0.3	7.8	-7.3	-7.5	17.3	-3.6	-3.8	23.1	-3.7	-3.8	25.9	-0.2	-0.4
0.4	10.7	-6.1	-6.3	23.4	-22.4	-22.6	30.6	-7.5	-7.5	33.9	-0.2	-0.3
0.5	14.3	-5.0	-5.1	28.3	-2.5	-2.6	38.0	-3.8	-3.8	42.6	-0.2	-0.3

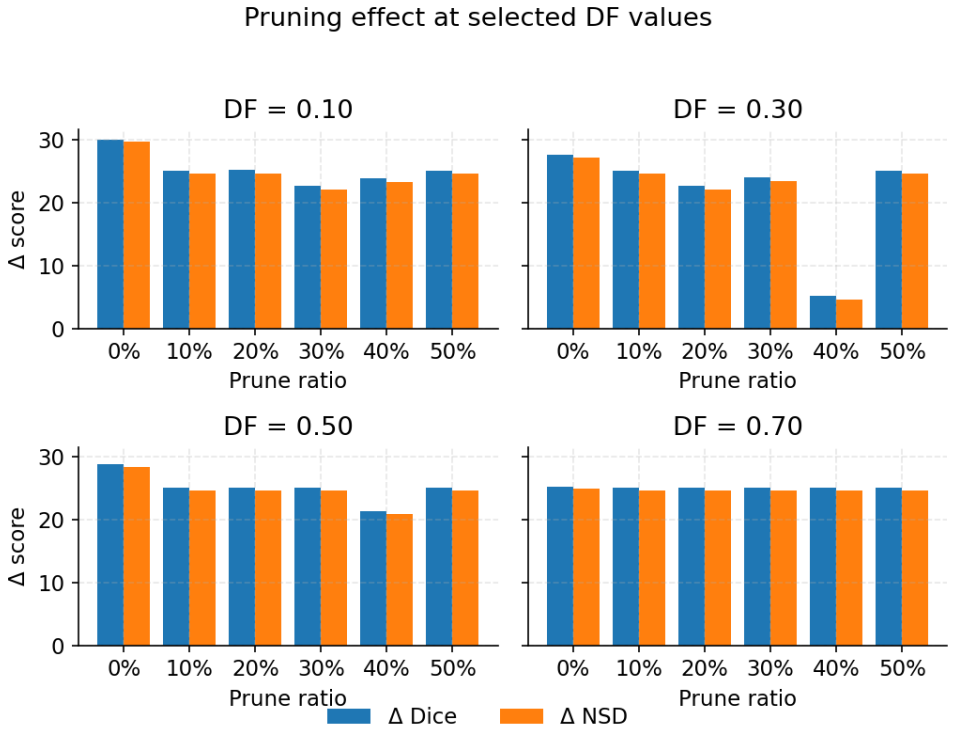


FIGURE 5.2: Pruning effect across different downsampling factor (DF). The additional parameter reduction is evaluated, Dice change (blue), and NSD change (orange) relative to the decomposition-only baseline for each pruning ratio.

Our experiments show that compression using Tucker decomposition and HVS based channel pruning can drastically reduce in model size while preserving or even improving segmentation quality. Downsampling alone can act as a regulariser of low rank factorisation, enhancing generalisation.

Despite the promising compression and performance gains observed in this study, several directions remain for further improvement. The current approach applies Tucker decomposition uniformly across convolutional layers using a global downsampling factor. A more adaptive strategy where ranks are determined could yield better compression and accuracy trade-offs. In addition, the present evaluation focuses primarily on parameter reduction. Further experiments require to include FLOPs , memory bandwidth analysis would provide a more comprehensive assessment of deployment efficiency. Finally, evaluating the method across multiple datasets, anatomical regions, and clinical imaging protocols would better validate generalization and robustness.

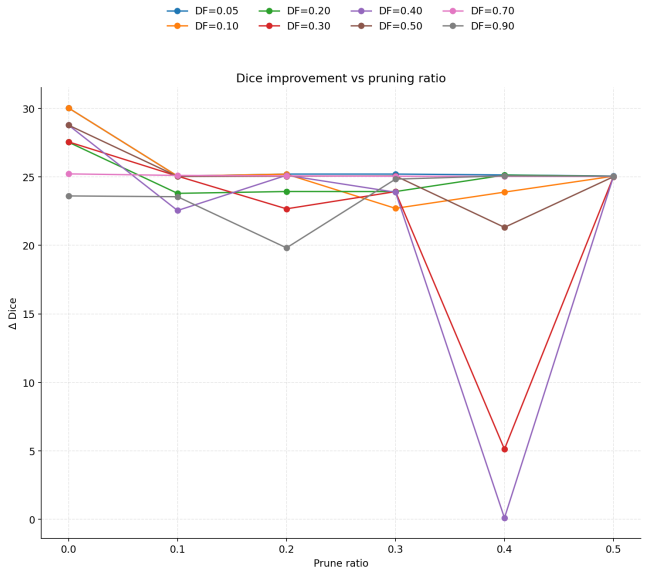


FIGURE 5.3: Dice score improvement versus pruning ratio for each downsampling factors (DF).

Future work should (i) develop adaptive rank selection per layer guided by Theorem 5.1, (ii) extend the theoretical analysis to transformer based segmentation architectures, and (iii) validate on additional 3D image modalities such as MRI and ultrasound.

CHAPTER 6

CONCLUSION AND PERSPECTIVES

Summary of Contributions

The central premise of this thesis is that *pre-trained neural networks are over-parameterised by design*: they are trained to generalise across vast and heterogeneous distributions, and consequently retain far more capacity than any single downstream task requires. This redundancy, which is often perceived as a limitation at deployment time, is in fact an opportunity. The three contributions presented in this thesis each exploit this redundancy in a different but complementary way, forming a coherent compression pipeline that spans the full spectrum from task adaptation to architectural reduction.

Selective fine-tuning by layer selection. The first contribution addresses the question of *which parameters to adapt* when transferring a pre-trained model to a new task. Rather than fine-tuning all layers indiscriminately or freezing them according to a manual heuristic, we proposed a principled, automatic criterion based on the Kullback-Leibler divergence between the weight distributions of the source and fine-tuned networks. Layers whose distributions have shifted significantly during initial fine-tuning are selected for further optimisation; layers that remain stable are frozen, preserving the generic representations acquired during pre-training.
The theoretical analysis formalises the intuition behind this criterion. Theorem 3.2 establishes that the transfer error is bounded by two competing quantities: the divergence between source and target weight distributions, and the number of free parameters relative to the size of the target dataset. Freezing low-divergence layers simultaneously reduces both terms, providing a dual benefit that explains the empirical superiority of our method over both standard fine-tuning and random layer selection. Theorem 3.7 further guarantees that this selection strategy prevents negative transfer: as long as the divergence threshold is chosen conservatively, our method can never perform worse than training from scratch. The complexity analysis quantifies the computational advantage: because the KL divergence is computed over weight tensors rather than over data instances, the selection cost is *independent of the dataset size*, unlike policy-based methods such as SpotTune whose overhead grows linearly with N.
Experimentally, selecting as few as three layers on ResNet-50 matches or exceeds the accuracy of SpotTune across seven of the eight evaluation benchmarks, while requiring only a fraction of the parameters and computation. This result demonstrates that transfer learning efficiency is not primarily a function of *how much* is fine-tuned, but

what is fine-tuned.

Importance-guided neuron pruning for 2D networks. The second contribution shifts the question from *layer selection* to *neuron selection within layers*. Standard pruning criteria magnitude, gradient, or Taylor sensitivity evaluate each neuron in isolation or with respect to its immediate neighbourhood. This local perspective ignores a fundamental property of deep networks. The output of any neuron propagates through all subsequent layers before affecting the final prediction. A neuron that appears unimportant locally may be ndispensable for the activation of a critical unit in a deeper layer.

We formalised this observation through the Heuristic for Variable Selection (HVS), which computes a neuron's importance as the cumulative product of its partial contributions along every path reaching the output. Theorem 4.4 establishes that the HVS score S_j is a provable upper bound on the perturbation induced in the network's output when neuron j is removed. Pruning neurons in increasing order of S_j therefore minimises the worst-case impact on predictions a guarantee that magnitude-based criteria do not provide. Proposition 4.9 further shows that HVS upper-bounds the first-order Taylor sensitivity, meaning it subsumes gradient-based importance as a special case while being computable from forward-pass weights alone, without requiring gradient storage. This property is particularly valuable in the transfer learning setting, where gradients through frozen layers are never computed.

The generalisation bound of Theorem 4.6 provides a theoretical explanation for an empirically observed but often unexplained phenomenon: moderate pruning (10–30%) can *improve* accuracy relative to the unpruned baseline. The bound decomposes the generalisation error into a complexity penalty (which decreases as more neurons are removed) and an approximation error (which increases). When the removed neurons have small HVS scores, the approximation error grows slowly, and the reduction in complexity dominates yielding a net improvement. This is confirmed across four datasets, with accuracy gains of up to $+8.52\%$ on DTD and $+5.80\%$ on Flowers-102 at pruning ratios of 30%.

Tucker decomposition and HVS pruning for 3D segmentation. The third contribution extends the compression framework to volumetric medical image segmentation a domain where the memory and computational constraints are most acute. A 3D U-Net operating on CT volumes at clinical resolution can require an order of magnitude more memory than its 2D counterpart, making direct deployment in hospital infrastructure impractical.

We proposed a two-stage compression pipeline. In the first stage, Tucker decomposition is applied to every convolutional weight tensor, factorising the channel dimensions into a compact core and two projection matrices. This operation replaces each heavy convolution with three lightweight convolutions, reducing both the parameter count and the number of floating-point operations. In the second stage, the HVS-based channel pruning developed in Chapter 4 is applied to the compressed network, removing channels that are redundant in the low-rank space induced by Tucker factorisation.

The theoretical analysis reveals the mechanism behind a striking empirical observation: Tucker compression not only preserves accuracy but *improves* it. Proposition 5.5 establishes that Tucker decomposition is equivalent to adding a nuclear norm regulariser to the training objective. The pre-trained TotalSegmentator model, which was optimised for prediction on a large and heterogeneous dataset, contains

weight directions that are poorly aligned with the target evaluation distribution. Truncating the singular value spectrum removes these misaligned directions, acting as a form of domain-specific regularisation that reduces overfitting to ImageNet-style features while preserving anatomically relevant representations. This explains the Dice improvements of up to $+30$ pp observed across all tested downsampling factors.

The combined compression bound of Theorem 5.6 characterises the joint effect of Tucker downsampling (controlled by DF) and HVS channel pruning (controlled by ρ) on the generalisation error. Corollary 5.7 derives an explicit trade-off between the two hyperparameters: because the two error terms are additive and independent, they can be optimised separately, and the optimal pruning ratio increases as Tucker compression becomes more aggressive. This prediction is confirmed by the experimental results: a pruning ratio of $\rho = 0.5$ is optimal across all tested values of DF, including the most aggressive configurations. Compression ratios of up to $13\times$ are achieved with parameter reductions exceeding 92%, while maintaining Dice improvements of $+25$ pp over the uncompressed baseline.

Unified View of the Three Contributions

Taken together, the three contributions form a coherent and complementary framework for efficient transfer learning and model compression:

- **Layer selection** (Chapter 3) determines *where* in the network adaptation should occur, minimising the number of parameters that must be learned from limited target data.
- **Neuron pruning** (Chapter 4) determines *which units* within those layers are genuinely necessary for the target task, removing redundancy that was inherited from pre-training on a different distribution.
- **Tensor decomposition** (Chapter 5) operates at the level of *weight structure*, replacing dense operators with low rank approximations that are both more compact and, through their implicit regularisation effect, better adapted to the target domain.

Each contribution addresses a distinct axis of redundancy in pre-trained models: representational redundancy (unnecessary layers), unit-level redundancy (unnecessary neurons), and spectral redundancy (unnecessary singular components). The theoretical frameworks developed for each KL divergence bounds, HVS perturbation bounds, and Tucker approximation error bounds are mutually consistent and can in principle be combined into a single unified bound, as demonstrated by Theorem 5.6 for the last two contributions.

A further common thread is the *theoretical justification of implicit regularisation*. In Chapter 3, freezing low-divergence layers reduces the effective hypothesis space and prevents overfitting to small target datasets. In Chapter 4, removing low-importance neurons reduces the Rademacher complexity of the pruned network, explaining generalisation improvements at moderate pruning ratios. In Chapter 5, Tucker decomposition imposes a nuclear norm prior on the weight matrices, smoothing the parameter landscape and reducing the influence of task-irrelevant features. In all three cases, compression is not merely a computational convenience it is a principled form of regularisation that can improve generalisation.

Limitations

Despite these contributions, several limitations must be acknowledged honestly. The theoretical results in Chapter 3 rely on standard regularity assumptions (Lipschitz continuity, strong convexity, i.i.d. data) that are not strictly satisfied by deep networks trained on heterogeneous medical or artistic datasets. The bounds derived are therefore guidance for understanding rather than tight performance guarantees, and the constants C_1, C_2 are not explicitly characterised. Similarly, the proof of Lemma 3.12 which asserts a monotone increase of KL divergence with network depth is presented as an empirically motivated argument rather than a rigorous derivation. Providing a formal proof under appropriate distributional assumptions would strengthen the theoretical contribution.

The experimental evaluation of the layer selection method (Chapter 3) is limited to ResNet-50 on image classification tasks. While the approach is architecture-agnostic in principle, its behaviour on transformer-based models or on regression tasks remains unexplored. The choice of a percentile-based selection threshold, while practical, lacks a data-driven adaptation mechanism: the same threshold is applied regardless of the domain shift or dataset size, whereas Theorem 3.2 suggests that the optimal selection ratio should scale as $k \propto \sqrt{N}$.

For the HVS pruning method (Chapter 4), the current implementation operates at the granularity of individual neurons rather than channels, which limits the wall-clock speedups achievable in practice. Modern GPU hardware is optimised for dense tensor operations; irregular sparsity patterns introduced by neuron-level pruning do not translate directly into inference acceleration without specialised sparse execution backends. Extending the method to channel-level pruning, as already demonstrated in Chapter 5 for 3D networks, would be a natural step towards practical deployment.

Finally, the evaluation of the 3D compression method is conducted on a single dataset (TotalSegmentator) using a single model family (nnU-Net). The universality of the observed Dice improvements under Tucker compression remains to be validated across other anatomical structures, imaging modalities (MRI, ultrasound), and segmentation architectures. In particular, the dramatic performance gains observed for the lowest downsampling factors deserve further investigation: they suggest that the baseline model retains a substantial amount of task-irrelevant structure, and understanding *why* would shed light on the broader question of what pre-trained medical segmentation models actually learn.

Perspectives for Future Research

The work presented in this thesis opens several directions for future investigation, spanning theoretical, methodological, and applied dimensions.

Adaptive and online layer selection. The current layer selection strategy is static: layers are selected once, before the second fine-tuning phase, and the selection does not change during training. A more flexible approach would allow the selection

to evolve as the target model converges, adding or removing layers from the fine-tuning set based on the evolution of their weight distributions. This connects naturally to the literature on continual learning and progressive network growing, where the capacity of the model is adjusted dynamically in response to the task.

Beyond KL divergence: information-geometric criteria. The KL divergence used in Chapter 3 is asymmetric and undefined when the support of the target distribution is not contained in that of the source. The Jensen-Shannon divergence and the Wasserstein distance, which are explored experimentally, offer theoretical properties that could lead to tighter bounds or more robust selection in cases of severe domain shift. A promising direction is to frame layer selection as a problem in information geometry, where the curvature of the statistical manifold of weight distributions provides a natural metric for measuring task similarity.

Joint optimisation of decomposition and pruning. In Chapter 5, Tucker decomposition and HVS pruning are applied sequentially. A joint optimisation that simultaneously determines the rank structure and the channel importance scores could yield better compression-accuracy trade offs, particularly in regimes where the two sources of redundancy interact. This could be formulated as a bilevel optimisation problem, where the inner level optimises the network weights and the outer level optimises the compression parameters (DF, ρ) with respect to a validation objective.

Extension to transformer architectures. The methods developed in this thesis are designed for convolutional networks and rely on the spatial structure of convolutional weight tensors. The dominant paradigm in both natural language processing and, increasingly, medical image analysis is now based on attention mechanisms and transformer architectures. The HVS importance criterion can in principle be extended to attention heads by treating each head as a unit whose importance is measured by its contribution to the final token representations. Tucker decomposition has a natural analogue for the weight matrices of self-attention layers through low-rank matrix factorisation. Developing a unified compression framework for both convolutional and attention-based architectures is a pressing challenge for the field.

Theoretical tightening. Several of the theoretical results presented in this thesis are proof of concept bounds that establish the *qualitative* correctness of the proposed criteria without being numerically tight. Future work should aim to derive bounds whose constants are explicitly characterised in terms of observable quantities (e.g., the spectral properties of the weight tensors, the Lipschitz constant of the network), so that they can be used to predict performance from the model's architecture alone. This would transform the theoretical framework from a post-hoc justification into a predictive tool for compression design.

Clinical validation and deployment. The ultimate goal of compressing 3D medical segmentation networks is to enable their deployment on clinical hardware in operating rooms, on mobile ultrasound devices, or at the edge of hospital networks. Achieving this goal requires not only parameter and FLOP reduction, but also validation of the compressed models under distribution shift (different scanners, acquisition protocols, patient populations) and compliance with regulatory frameworks for medical AI. Future work should establish systematic evaluation protocols that connect model compression to clinically meaningful metrics, including uncertainty quantification and failure mode analysis.

Closing Remarks

Neural network compression is sometimes presented as a purely engineering problem matter of reducing memory footprints and inference latency to meet deployment constraints. This thesis argues for a different perspective. The redundancy present in pre-trained models is not merely an inconvenience to be eliminated; it is a signal about the structure of the learned representations, and understanding it leads to better compression, better generalisation, and ultimately, better science.

The three contributions presented here KL-guided layer selection, HVS-based neuron pruning, and Tucker-HVS compression for 3D segmentation each illuminate a different facet of this redundancy and propose a principled method to exploit it. The theoretical frameworks developed to analyse these methods reveal a common underlying theme: *compression is regularisation*. By selectively removing or constraining the parameters of a pre-trained model, we impose an implicit prior that discards task-irrelevant information and focuses the model's capacity where it is most needed. This perspective, grounded in both information theory and statistical learning theory, offers a foundation for future work that goes beyond the empirical optimisation of compression ratios towards a principled science of efficient deep learning.

References

C. Amrit, T. Paauw, R. Aly, and M. Lavric. Identifying child abuse through text mining and machine learning. *Expert Systems with Applications*, 88, 10 2017. doi: 10.1016/j.eswa.2017.06.035.

S. Anoopa, A. Salim, and S. Nadera Beevi. Complexity analysis of hierarchical tucker-based tensor decomposition in 3d cnn and lstm architectures for video data. In *2025 11th International Conference on Smart Computing and Communications (ICSCC)*, pages 1–5, 2025. doi: 10.1109/ICSCC66177.2025.11233461.

P. L. Bartlett and S. Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. In D. P. Helmbold and R. C. Williamson, editors, *COLT/EuroCOLT*, volume 2111 of *Lecture Notes in Computer Science*, pages 224–240. Springer, 2001. ISBN 3-540-42343-5. URL http://dblp.uni-trier.de/db/conf/colt/colt2001.html#BartlettM01.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018. doi: 10.1137/16M1080173. URL https://doi.org/10.1137/16M1080173.

J. C. Caicedo, J. Roth, A. Goodman, T. Becker, K. W. Karhohs, M. Broisin, C. Molnar, C. McQuin, S. Singh, F. J. Theis, and A. E. Carpenter. Evaluation of deep learning strategies for nucleus segmentation in fluorescence images. *Cytometry Part A*, 95(9):952–965, 2019. doi: https://doi.org/10.1002/cyto.a.23863. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cyto.a.23863.

E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *CoRR*, abs/0805.4471, 2008. URL http://arxiv.org/abs/0805.4471.

H. Cheng, M. Zhang, and J. Q. Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10558–10578, 2024. doi: 10.1109/TPAMI.2024.3447085.

M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

M. Crawford, T. Khoshgoftaar, J. Prusa, A. Richter, and H. Al-Najada. Survey of review spam detection using machine learning techniques. *Journal of Big Data*, 2:23, 10 2015. doi: 10.1186/s40537-015-0029-9.

Y. Deldjoo, M. Elahi, P. Cremonesi, F. Garzotto, P. Piazzolla, and M. Quadrana. Content-based video recommendation system based on stylistic visual features. *Journal on Data Semantics*, 5:1–15, 06 2016. doi: 10.1007/s13740-016-0060-9.

M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning, 2014. URL https://arxiv.org/abs/1306.0543.

E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/1adaeb993eba95859121a43ea61bd858-Paper.pdf.

N. Dinsdale, M. Jenkinson, and A. Namburete. Stamp: Simultaneous training and model pruning for low data regimes in medical image segmentation. *Medical Image Analysis*, 81, Oct. 2022a. ISSN 1361-8415. doi: 10.1016/j.media.2022.102583. Publisher Copyright: © 2022 The Author(s).

N. K. Dinsdale, M. Jenkinson, and A. I. Namburete. Stamp: Simultaneous training and model pruning for low data regimes in medical image segmentation. *Medical Image Analysis*, 81:102583, 2022b. ISSN 1361-8415. doi: https://doi.org/10.1016/j.media.2022.102583. URL https://www.sciencedirect.com/science/article/pii/S1361841522002213.

M. Eitz, J. Hays, and M. Alexa. How do humans sketch objects? *ACM Trans. Graph.*, 31(4), July 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185540. URL https://doi.org/10.1145/2185520.2185540.

J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

M. Gabor and R. Zdunek. Compressing convolutional neural networks with hierarchical tucker-2 decomposition. *Applied Soft Computing*, 132:109856, 2023. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2022.109856. URL https://www.sciencedirect.com/science/article/pii/S156849462200905X.

T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *ArXiv*, abs/1902.09574, 2019. URL https://api.semanticscholar.org/CorpusID:67855585.

W. Ge and Y. Yu. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning, 2017. URL https://arxiv.org/abs/1702.08690.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11 2013. doi: 10.1109/CVPR.2014.81.

Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris. Spottune: Transfer learning through adaptive fine-tuning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4800–4809, 2019. doi: 10.1109/CVPR.2019.00494.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv: Computer Vision and Pattern Recognition*, 2015a. URL https://api.semanticscholar.org/CorpusID:2134321.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 1135–1143. Curran Associates, Inc., 2015b. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf.

R. I. Hasan, S. M. Yusuf, and L. Alzubaidi. Review of the state of the art of deep learning for plant diseases: A broad analysis and discussion. *Plants*, 9(10), 2020. ISSN 2223-7747. doi: 10.3390/plants9101302. URL https://www.mdpi.com/2223-7747/9/10/1302.

B. Hassibi and D. G. Stork. Optimal brain surgeon and general network pruning. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 293–299, 1993.

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)*, 2016a.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, 2016b.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016c. doi: 10.1109/CVPR.2016.90.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016d. doi: 10.1109/CVPR.2016.90.

Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 2234–2240. AAAI Press, 2018. ISBN 9780999241127.

L. Huyan, Y. Li, D. Jiang, Y. Zhang, Q. Zhou, B. Li, J. Wei, J. Liu, Y. Zhang, P. Wang, and H. Fang. Remote sensing imagery object detection model compression via tucker decomposition. *Mathematics*, 11(4), 2023. ISSN 2227-7390. doi: 10.3390/math11040856. URL https://www.mdpi.com/2227-7390/11/4/856.

S. Imai, S. Kawai, and H. Nobuhara. Stepwise pathnet: a layer-by-layer knowledge-selection-based transfer learning algorithm. *Scientific Reports*, 10, 2020. URL https://api.semanticscholar.org/CorpusID:218670731.

F. Isensee, J. Petersen, A. Klein, D. Zimmerer, P. F. Jaeger, S. Kohl, J. Wasserthal, G. Koehler, T. Norajitra, S. Wirkert, and K. H. Maier-Hein. nnu-net: Self-adapting framework for u-net-based medical image segmentation, 2018. URL https://arxiv.org/abs/1809.10486.

J.-C. Jeong, G.-H. Yu, M.-G. Song, D. Vu, A. Le, Y.-A. Jung, Y.-A. Choi, T.-W. Um, and J.-Y. Kim. Selective layer tuning and performance study of pre-trained models using genetic algorithm. *Electronics*, 11:2985, 09 2022. doi: 10.3390/electronics11192985.

L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6:366–422, 1960. URL https://api.semanticscholar.org/CorpusID:62611375.

J. Ker, L. Wang, J. Rao, and T. Lim. Deep learning applications in medical image analysis. *IEEE Access*, 6:9375–9389, 2018. doi: 10.1109/ACCESS.2017.2788044.

D. S. Kermany et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131, 2018. Dataset: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia.

A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013. doi: 10.1109/ICCVW.2013.77.

V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. 12 2014.

Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. doi: 10.1038/nature14539.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. 08 2016a. doi: 10.48550/arXiv.1608.08710.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2016b. URL https://api.semanticscholar.org/CorpusID:14089312.

H.-C. Li, Z.-X. Lin, T.-Y. Ma, X.-L. Zhao, A. Plaza, and W. J. Emery. Hybrid fully connected tensorized compression network for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 61:1–16, 2023. doi: 10.1109/TGRS.2023.3241193.

L. Li, Y. Ye, Z. Chen, and Y. Xia. Unified start, personalized end: Progressive pruning for efficient 3d medical image segmentation. *ArXiv*, abs/2509.09267, 2025. URL https://api.semanticscholar.org/CorpusID:281252718.

X. LI, Y. Grandvalet, and F. Davoine. Explicit inductive bias for transfer learning with convolutional networks. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2825–2834. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/li18a.html.

X. Liu and K. K. Parhi. Tensor decomposition for model reduction in neural networks: A review [feature]. *IEEE Circuits and Systems Magazine*, 23(2):8–28, 2023. doi: 10.1109/MCAS.2023.3267921.

Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763, 2017. doi: 10.1109/ICCV.2017.298.

Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *ArXiv*, abs/1810.05270, 2018. URL https://api.semanticscholar.org/CorpusID:52978527.

M. Long, Y. Cao, J. Wang, and M. Jordan. Learning transferable features with deep adaptation networks. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 97–105, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/long15.html.

S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *ArXiv*, abs/1611.06440, 2016. URL https://api.semanticscholar.org/CorpusID:16167970.

M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, 2008. URL https://api.semanticscholar.org/CorpusID:15193013.

S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010. URL https://api.semanticscholar.org/CorpusID:740063.

O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

S. Pereira, A. Pinto, V. Alves, and C. A. Silva. Brain tumor segmentation using convolutional neural networks in mri images. *IEEE Transactions on Medical Imaging*, 35 (5):1240–1251, 2016. doi: 10.1109/TMI.2016.2538465.

A. Peste. *Efficiency and Generalization of Sparse Neural Networks*. PhD thesis, Institute of Science and Technology Austria, 2023.

A. Quattoni and A. Torralba. Recognizing indoor scenes. pages 413–420, 06 2009. doi: 10.1109/CVPR.2009.5206537.

N. Rieke, J. Hancox, W. Li, F. Milletari, H. Roth, S. Albarqouni, S. Bakas, M. Galtier,

B. Landman, K. Maier-Hein, S. Ourselin, M. Sheller, R. Summers, A. Trask, D. Xu, M. Baust, and M. J. Cardoso. The future of digital health with federated learning. *npj Digital Medicine*, 3, 12 2020. doi: 10.1038/s41746-020-00323-1.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015a. Springer International Publishing. ISBN 978-3-319-24574-4.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015b. arXiv:1505.04597.

M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich. To transfer or not to transfer. In *Neural Information Processing Systems*, 2005. URL https://api.semanticscholar.org/CorpusID:597779.

B. Saleh and A. Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *International Journal for Digital Art History*, (2), Oct. 2016. doi: 10.11588/dah.2016.2.23376. URL https://journals.ub.uni-heidelberg.de/index.php/dah/article/view/23376.

Z. Shi, H. Hao, M. Zhao, Y. Feng, L. He, Y. Wang, and K. Suzuki. A deep cnn based transfer learning method for false positive reduction. *Multimedia Tools Appl.*, 78(1): 1017–1033, Jan. 2019. ISSN 1380-7501. doi: 10.1007/s11042-018-6082-6. URL https://doi.org/10.1007/s11042-018-6082-6.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. arXiv:1409.1556.

F. I. Tushar, V. M. D'Anniballe, G. D. Rubin, and J. Y. Lo. What limits performance of weakly supervised deep learning for chest ct classification?, 2024. URL https://arxiv.org/abs/2402.04419.

G. Vrbančič, M. Zorman, and V. Podgorelec. *Transfer Learning Tuning Utilizing Grey Wolf Optimizer for Identification of Brain Hemorrhage from Head CT Images*, pages 61–66. 10 2019. ISBN 9789617055825. doi: 10.26493/978-961-7055-82-5.61-66.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech–ucsd birds-200-2011 dataset. Technical report, California Institute of Technology, 2011.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset, Aug. 2023.

R. N. Wanjiku, L. Nderu, and M. Kimwele. Dynamic fine-tuning layer selection using kullback–leibler divergence. *Engineering Reports*, 5(5):e12595, 2023. doi: https://doi.org/10.1002/eng2.12595. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/eng2.12595.

J. Wasserthal. Dataset with segmentations of 117 important anatomical structures in 1228 ct images, Oct. 2023. URL https://doi.org/10.5281/zenodo.10047292.

J. Wasserthal, H.-C. Breit, M. T. Meyer, M. Pradella, D. Hinck, A. W. Sauter, T. Heye, D. T. Boll, J. Cyriac, S. Yang, M. Bach, and M. Segeroth. Totalsegmentator: Robust segmentation of 104 anatomic structures in ct images. *Radiology: Artificial Intelligence*, 5(5), Sept. 2023. ISSN 2638-6100. doi: 10.1148/ryai.230024. URL http://dx.doi.org/10.1148/ryai.230024.

T. Weber, J. Dexl, D. Rügamer, and M. Ingrisch. Post-training network compression for 3d medical image segmentation: Reducing computational efforts via tucker decomposition. *Radiology: Artificial Intelligence*, 7(2):e240353, 2025. doi: 10.1148/ryai.240353. URL https://doi.org/10.1148/ryai.240353. PMID:

39812583.
J. Xiao, X. Yin, A. Liu, and Z. Yu. A cnn-based framework for automatic segmentation of chest x-ray images and multi-type respiratory disease recognition. *Traitement du Signal*, 42:1447–1455, 06 2025. doi: 10.18280/ts.420319.
Y. Xiao, Z. Tian, J. Yu, Y. Zhang, S. Liu, S. Du, and X. Lan. A review of object detection based on deep learning. *Multimedia Tools Appl.*, 79(33–34):23729–23791, sep 2020. ISSN 1380-7501. doi: 10.1007/s11042-020-08976-6. URL https://doi.org/10.1007/s11042-020-08976-6.
Z. Xu, T. Ajanthan, P. H. S. Torr, N. Dziri, et al. Ranp: Resource aware neuron pruning at initialization for 3d cnns. In *Proceedings of the International Conference on 3D Vision (3DV)*, 2020. arXiv:2010.02488.
M. Yacoub and Y. Bennani. Hvs: A heuristic for variable selection in multilayer artificial neural network classifier. *Intelligent Engineering Systems Through Artificial Neural Networks*, 7, 01 1997a.
M. Yacoub and Y. Bennani. Hvs: A heuristic for variable selection in multilayer artificial neural network classifier. *Intelligent Engineering Systems Through Artificial Neural Networks*, 7, 01 1997b.
M. Yacoub and Y. Bennani. Features selection and architecture optimisation in connectionist systems. *International Journal of Neural Systems*, 10(05):379–395, 2000. doi: 10.1142/S0129065700000338.
T. Yang, Y. Zhao, and Q. Tao. Pruning nnu-net with minimal performance loss. In *Medical Imaging with Deep Learning - Short Papers*, 2025. URL https://openreview.net/forum?id=uTTOhthEDR.
J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3320–3328, Cambridge, MA, USA, 2014. MIT Press.
R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2017. URL https://api.semanticscholar.org/CorpusID:4142619.
Y. Zhang, Q. Liao, L. Ding, and J. Zhang. Bridging 2d and 3d segmentation networks for computation-efficient volumetric medical image segmentation: An empirical study of 2.5d solutions. *Computerized Medical Imaging and Graphics*, 99: 102088, 2022. ISSN 0895-6111. doi: https://doi.org/10.1016/j.compmedimag.2022.102088. URL https://www.sciencedirect.com/science/article/pii/S0895611122000611.