DISS. ETH NO. 28256

TOWARDS EFFICIENT DEEP NEURAL NETWORKS

A dissertation submitted to
ETH ZÜRICH

for the degree of
DOCTOR OF SCIENCES OF ETH ZÜRICH
(Dr. sc. ETH Zürich)

presented by
YAWEI LI
Master of Science, University of Electronic Science and Technology of China
born 14 September 1990
citizen of China

accepted on the recommendation of
Prof. Dr. Luc Van Gool, examiner
Prof. Dr. Thomas Brox, co-examiner
Prof. Dr. Ming-Hsuan Yang, co-examiner
Prof. Dr. Radu Timofte, co-examiner

2022

ABSTRACT

Computational efficiency is an essential factor that influences the applicability of computer vision algorithms. Although deep neural networks have reached state-of-the-art performances in a variety of computer vision tasks, there are a couple of efficiency related problems of the deep learning based solutions. First, the overparameterization of deep neural networks results in models with millions of parameters, which lowers the parameter efficiency of the designed networks. To store the parameters and intermediate feature maps during the computation, a large device memory footprint is required. Secondly, the massive computation in deep neural networks slows down their training and inference. This limits the application of deep neural networks to latency-demanding scenarios and low-end devices. Thirdly, the massive computation consumes significant amount of energy, which leaves a large carbon footprint of deep learning models.

The aim of this thesis is to improve the computational efficiency of current deep neural networks. This problem is tackled from three perspective including neural network compression, neural architecture optimization, and computational procedure optimization.

In the first part of the thesis, we reduce the model complexity of neural networks by network compression techniques including filter decomposition and filter pruning. The basic assumption for filter decomposition is that the ensemble of filters in deep neural networks constitutes an overcomplete set. Instead of using the original filters directly during the computation, they can be approximated by a linear combination of a set of basis filters. The contribution of this thesis is to provide a unified analysis of previous filter decomposition methods. On the other hand, a differentiable filter pruning method is proposed. To achieve differentiability, the layers of neural networks is reparameterized by a meta network. Sparsity regularization is applied to the input of the meta network, *i.e.* latent vectors. Optimizing with the introduced regularization leads to an automatic network pruning method. Additionally, a joint analysis of filter decomposition and filter pruning is presented from the perspective of compact tensor approximation. The hinge of the two techniques is the introduced sparsity inducing

matrix. By simply changing the way the group sparsity regularization is enforced to the matrix, the two techniques can be derived accordingly.

Secondly, we try to improve the performance of a baseline network by a fine-grained neural architecture optimization method. Different from network compression methods, the aim of this method is to improve the prediction accuracy of neural networks while reducing their model complexity at the same time. Achieving the two targets simultaneously makes the problem more challenging. In addition, a nearly cost-free constraint is enforced during the architecture optimization, which differs from current neural architecture search methods with bulky computation. This can be regarded as another efficiency-improving technique.

Thirdly, we optimize the computational procedure of graph neural networks. By mathematically analyzing the operations in graph neural network, two methods are proposed to improve the computational efficiency. The first method is related to the simplification of neighbor querying in graph neural network while the second involves shuffling the order of graph feature gathering and an feature extraction operations.

To summarize, this thesis contributes to multiple aspects of improving the computational efficiency of neural networks during the optimization, training, and test phase.

ZUSAMMENFASSUNG

Die Recheneffizienz ist einer der wichtigsten Faktoren der die Anwendbarkeit von Computer-Vision-Algorithmen beeinflusst. Obwohl tiefe neuronale Netze die zurzeit besten Leistungen in verschieden Bereichen der Bildverarbeitung erzielen, gilt es einige Schwachstellen zu verbessern, wie der hohe Rechenaufwand der verwendeten Deep Learning Methoden. Neuronale Netze sind häufig überparametrisiert und bestehen aus Millionen Parametern, sodass sich die Parameter-Ausnutzungsziffer des konstruierten Netzes verringert. Um alle Parameter und Zwischenresultate des Netzes während des Lernvorgangs verarbeiten zu können, wird daher ein grosses Volumen an Arbeitsspeicher benötigt. Weiter reduziert der hohe Rechenaufwand eines neuronalen Netzes die Laufzeit während des Lernvorgangs und später in der Produktion deutlich. Insbesondere in Latenz-kritischen Szenarien oder für Low-End Geräte schränkt der hohe Rechenaufwand die Anwendbarkeit von neuronalen Netzen ein. Ausserdem führt der erhebliche Rechenaufwand zu einem hohen Energieverbrauch und damit zu einem nicht zu vernachlässigen ökologischen Fussabdruck.

Das Ziel dieser Dissertation ist es die Recheneffizienz von aktuellen tiefen neuronalen Netzen zu verbessern. Die Lösungsstrategie besteht darin neuronale Netze zu komprimieren und deren Berechnungsverfahren zu optimieren, wie in den folgenden drei Punkten erläutert.

Zuerst wird die Anzahl an Parametern von neuronalen Netzen reduziert durch ein neuartiges Filterkern-Zerlegungsverfahren und eine Filterkern-Reduktionsmethode. Die Annahme, dass die die Filter-Kerne eines Netzes eine übervollständige Menge darstellen, bildet die Grundlage der Filter-Kern-Zerlegung. Dabei werden Filterkerne aus einer linearen Kombination von erlernten Basisvektoren geformt, sodass deren Werte eine Annäherung an die ursprünglichen Filterkern Werte darstellen. Ein erster wesentlicher Beitrag dieser Dissertation liegt in einer vereinheitlichten Analyse der bestehenden Filterkern-Zerlegungsverfahren. Weiter wird eine differenziertbare Filterkern-Reduktionsmethode eingeführt. Die Differenzierbarkeit wird dadurch erreicht, dass die Parameter der verschiedenen Stufen eines neuronalen Netzes, die die Filterkerne repräsentieren, durch ein Meta-Netz bestimmt werden. Zusätzlich

wird eine vereinheitlichte Untersuchung von Filterkern-Zerlegung und Reduzierung aus dem Blickwinkel der kompakten Tensorapproximation vorgestellt. Die Verbindung der beiden Techniken bilden dabei die vorgeschlagenen dünnbesetzten Matrizen. Beide Techniken können hergeleitet werden, indem die Regulierung angepasst wird, die die dünnbesetzten Matrizen erzwingt.

Als zweites wird ein Optimierungsverfahren vorgestellt, dass es erlaubt die Struktur eines neuronalen Netzes zu verändern und damit dessen Leistung zu verbessern. In Gegensatz zu Komprimierungsverfahren, die nur die Anzahl Parameter eines Netzes reduzieren, zielt das vorgeschlagene Verfahren gleichzeitig auf die Verbesserung der Schätzgenauigkeit und der Reduzierung der Komplexität des neuronalen Netzes ab. Allerdings gestaltet sich das Erreichen beider Ziele zur gleichen Zeit relative schwierig. Das Verfahren setzt zusätzlich eine nahezu kostenfreie Bedingung während des Optimierungsschrittes durch. Hierbei unterscheidet sich das vorgestellte Verfahren von Herkömmlichen, die meistens deutlich rechenaufwändiger sind. Daher kann dieser Punkt als ein weiterer Beitrag zur Verbesserung der Recheneffizienz betrachtet werden.

Als drittes wird ein optimiertes Berechnungsverfahren für graphbasierten neuronalen Netzen eingeführt. Anhand einer Analyse der mathematischen Operationen, die zugrunde von graphbasierten neuronalen Netzen liegen, werden zwei neue Methoden vorgeschlagen, um die Recheneffizienz dieser Netzarten zu steigern. Die erste Methode ähnelt der Vereinfachung von Nachbarschaftsabfragen in graphbasierten neuronalen Netzen. Die Zweite hingegen besteht darin die Reihenfolge der Operationen zur Informationssammlung im Graph und Informationsextraktion zu verändern.

Diese Dissertation trägt in mehreren Punkten zur Steigerung der Recheneffizienz von neuronalen Netzen während des Optimierungsschrittes, der Lern- und Testphase bei.

PUBLICATIONS

The following publications are included in parts or in an extended version in this thesis:

- Yawei Li et al. „Learning Filter Basis for Convolutional Neural Network Compression.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 5623–5632.
- Yawei Li et al. „Group Sparsity: The Hinge Between Filter Pruning and Decomposition for Network Compression.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020.
- Yawei Li et al. „DHP: Differentiable meta pruning via hypernetworks.“ In: *Proceeding of the European Conference on Computer Vision*. Springer. 2020, pp. 608–624.
- Yawei Li et al. „The Heterogeneity Hypothesis: Finding Layer-Wise Differentiated Network Architectures.“ In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2144–2153.
- Yawei Li et al. „Towards Efficient Graph Convolutional Networks for Point Cloud Handling.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2021, pp. 2144–2153.

Furthermore, the following publications were part of my PhD research, are however not covered in this thesis. The topics of these publications are outside of the scope of the material covered here:

- Yawei Li et al. „CARN: convolutional anchored regression network for fast and accurate single image super-resolution.“ In: *Proceeding of the European Conference on Computer VisionW*. Springer. 2018, pp. 166–181.
- Yawei Li et al. „3D Appearance Super-Resolution with Deep Learning.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019

- Yawei Li et al. „LocalViT: Bringing locality to vision transformers.“ In: *arXiv preprint arXiv:2104.05707* (2021).
- Yawei Li et al. „Spatio-Temporal Gated Transformers for Efficient Video Processing.“ In: *Advances in Neural Information Processing Systems Workshops*. 2021.
- Yawei Li et al. „Revisiting Random Channel Pruning for Neural Network Compression.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2022
- Shuhang Gu et al. „Self-guided network for fast image denoising.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 2511–2520.
- Yunxuan Wei et al. „Unsupervised real-world image super resolution via domain-distance aware training.“ In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13385–13394.
- Rui Gong et al. „Cluster, Split, Fuse, and Update: Meta-Learning for Open Compound Domain Adaptive Semantic Segmentation.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2021, pp. 8344–8354
- Kai Zhang et al. „Plug-and-play image restoration with deep denoiser prior.“ In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
- Jiezhang Cao et al. „Video Super-Resolution Transformer.“ In: *arXiv preprint arXiv:2106.06847* (2021).

ACKNOWLEDGMENTS

During the PhD study, I have received help and support from many brilliant people, without whom this thesis would not be possible. I would like to express my gratitude to them here.

First and foremost, I would like to thank my supervisor Prof. Luc Van Gool for offering me the opportunity to study at Computer Vision Lab, for building up the environment of free exploration, for his patience of polishing papers before the deadlines, and his importance guidance during the turning point of my life and education.

I would like to express my gratitude to my co-supervisor Prof. Dr. Radu Timofte. We know each other during the first NTIRE workshop in 2016 when I was considering a PhD study. I thank him for the help discussion at that time and for introducing me to Prof. Luc Van Gool, which is an indispensable factor for me to get the opportunity at CVL. I thank him for his persistence on polishing my first CVPR paper that I almost gave up and for his continuous guidance during my later works.

Dr. Shuhang Gu is deeply involved with my works on network compression. I thank him for providing important ideas in those works, for the beautiful moments of attending conferences, and for the home gathering during festivals. During the later phase of PhD study, I am deeply influenced by Dr. Kai Zhang. I thank him for broadening my horizon and inspiring my ambition of working on valuable projects and producing influential works. I also thank Dr. Radu Timofte, Shuang Gu, and Kai Zhang for being a fantastic room mate. I love our apartment in Erlenbach and enjoy the hiking, swimming, and exploration near our apartment.

I thank Dr. Vagia Tsiminaki for the supervision of my first CVPR paper, which is important for building up my confidence. I thank Dr. Eirikur Agustsson for introducing the necessary knowledge and solving my coding problems in the beginning of my study. I thank Dr. Zhiwu Huang for confirming my performance during my application of the PhD position. I thank Dr. Wen Li, Dr. Martin Danelljan, Dr. Zhaopeng Cui for their important guidance and suggestions during my later works on efficient computation. I thank my peer collaborators Christoph Mayer, He Chen, Rui Gong, Jiezhang Cao, Jingyun Liang, and Kamil

Adamczewski. Without their diligent work, some of the publications or submissions would not be possible. I also thank Christoph Mayer for translating the abstract of this thesis.

I thank Amirhossein Habibian, Tijmen Blankevoort, Babak Ehteshami Bejnordi, Bert Moons for mentoring me during my internship at Qualcomm and Denis Demandolx, Rakesh Ranjan, Lucas Young, Yuchen Fan, and Xiaoyu Xiang for mentoring me during my internship at Facebook.

I also thank Yuhua Chen, Dengping Fan, Yulun Zhang, Yun Liu, Guolei Sun, Mengya Liu, Ce Liu, Wenguan Wang, Hao Tang, Danda Pani Paudel, Ajad Chhatkuli, Dengxin Dai, Ren Yang, Matthieu Paul, Chengde Wan, Xiaogang Cheng, Jie Qin, and all of the other colleagues at CVL for the helpful discussions, the joyful gatherings, and the wonderful ping pong times.

My special gratitude goes to Prof. Thomas Brox and Prof. Ming-Hsuan Yang. I thank them for reviewing my thesis during the tight period of the year.

I also thank my parents, all my family members and friends for supporting me during my life and study. I thank JingXian Ye for being on my side.

CONTENTS

Introduction 1

1 INTRODUCTION 3
- 1.1 Overview . . . 6
  - 1.1.1 Neural Network Compression . . . 6
  - 1.1.2 Neural Architecture Optimization . . . 8
  - 1.1.3 Computational Procedure Optimization . . . 9

I NEURAL NETWORK COMPRESSION 11

2 LEARNING FILTER BASIS 13
- 2.1 Introduction . . . 13
- 2.2 Related Work . . . 16
  - 2.2.1 Network Pruning . . . 16
  - 2.2.2 Network Quantization . . . 16
  - 2.2.3 Filter Decomposition . . . 17
  - 2.2.4 Knowledge Distillation . . . 18
- 2.3 Filter Decomposition for Network Compression . . . 18
  - 2.3.1 Decomposing convolution layer with filter basis . . . 18
  - 2.3.2 Compression rate with different filter basis . . . 20
  - 2.3.3 Implementing with convolution . . . 21
  - 2.3.4 Filter basis decomposition for special filter sizes . . . 23
- 2.4 Learning Filter Basis . . . 24
  - 2.4.1 General filter basis learning approach . . . 24
  - 2.4.2 Basis sharing . . . 25
- 2.5 Experimental Results . . . 26
  - 2.5.1 Experiment setup . . . 26
  - 2.5.2 Validation on super-resolution . . . 27
  - 2.5.3 Validation on image classification . . . 29
- 2.6 Conclusion . . . 33

3 DIFFERENTIABLE META PRUNING 37
- 3.1 Introduction . . . 37
- 3.2 Related Works . . . 40
  - 3.2.1 AutoML . . . 40
  - 3.2.2 Neural Architecture Search . . . 41
  - 3.2.3 Meta learning and hypernetworks . . . 41

3.3 Methodology . . . 42
3.3.1 Hypernetwork design . . . 42
3.3.2 Sparsity regularization and proximal gradient . . . 44
3.3.3 Network pruning . . . 46
3.3.4 Latent vector sharing . . . 46
3.3.5 Discussion on the convergence property . . . 49
3.3.6 Implementation consideration . . . 49
3.4 Experimental Results . . . 50
3.5 Ablation Study . . . 51
3.5.1 Image classification . . . 52
3.5.2 Super-resolution . . . 55
3.5.3 Denoising . . . 58
3.6 Conclusion and Future Work . . . 59
4 GROUP SPARSITY 61
4.1 Introduction . . . 61
4.2 Related Work . . . 63
4.2.1 Network Pruning with Group Sparsity . . . 64
4.2.2 Filter Decomposition and Group Sparsity . . . 64
4.3 The proposed method . . . 65
4.3.1 Group sparsity . . . 65
4.3.2 The hinge . . . 67
4.3.3 Proximal gradient solver . . . 68
4.3.4 Binary search of the nullifying threshold . . . 70
4.3.5 Gradient based adjustment of learning rate . . . 70
4.3.6 Group ℓ_2 norm based layer balancing . . . 71
4.3.7 Regularization factor annealing . . . 71
4.3.8 Distillation loss in the finetuning phase . . . 72
4.4 Closed-form Solutions to the Proximal Operators . . . 72
4.5 Implementation Considerations . . . 74
4.5.1 Sparsity-inducing matrix in network blocks . . . 74
4.5.2 Initialization of $\mathbf{W}$ and $\mathbf{A}$ . . . 75
4.6 Experimental Results . . . 75
4.6.1 Results on CIFAR10 . . . 76
4.6.2 Results on CIFAR100 . . . 80
4.7 Conclusion . . . 82

II NEURAL ARCHITECTURE OPTIMIZATION 83
5 THE HETEROGENEITY HYPOTHESIS 85
5.1 Introduction . . . 85

5.2 Related Work . . . 88
5.2.1 The lottery ticket hypothesis . . . 88
5.3 Preliminaries . . . 88
5.3.1 Hints from network compression . . . 88
5.3.2 Notations and definitions . . . 89
5.3.3 Problem formulation and recast . . . 90
5.4 Methodology . . . 91
5.4.1 Reparameterizing with hypernetworks . . . 92
5.4.2 Single-shot shrinkage . . . 92
5.4.3 Knowledge distillation . . . 93
5.4.4 Constraining model complexity . . . 94
5.5 Experimental Results . . . 94
5.5.1 Image Classification . . . 95
5.5.2 Visual Tracking . . . 103
5.5.3 Image Restoration . . . 104
5.6 Conclusion . . . 104

III COMPUTATIONAL PROCEDURE OPTIMIZATION 107
6 TOWARDS EFFICIENT GCN 109
6.1 Introduction . . . 109
6.2 Related Work . . . 113
6.2.1 Deep Learning for 3D Point Clouds. . . . 113
6.2.2 Efficient Network Design for 3D data . . . 113
6.3 Notations and Preliminaries . . . 114
6.4 Methodology . . . 115
6.4.1 Computational complexity analysis in GCN . . . 116
6.4.2 Propagation of point adjacency . . . 116
6.4.3 Graph convolution with graph feature gathering 122
6.5 Experiments . . . 123
6.5.1 Point Cloud Classification. . . . 125
6.5.2 Point Cloud Segmentation. . . . 128
6.5.3 Surface Reconstruction. . . . 129
6.5.4 Applicability. . . . 129
6.6 Conclusion . . . 132

Conclusion 133
7 CONCLUSION AND OUTLOOK 135
7.1 Contribution . . . 135
7.2 Challenges . . . 136

BIBLIOGRAPHY 139
ACRONYMS 161
INDEX 163

LIST OF FIGURES

Figure 2.1 Illustration of filters in AlexNet 13
Figure 2.2 Low-rank approximation of filters 14
Figure 2.3 Conversion from decomposed matrices to convolutions 14
Figure 2.4 Comparison of different filter decomposition methods 19
Figure 2.5 Illustration of the proposed basis learning method 22
Figure 2.6 Basis sharing for the compression of DenseNet-12-40 25
Figure 2.7 Visual results for compressed EDSR models. . . 30
Figure 2.8 Visual results for compressed SRResNet models. 31
Figure 2.9 Comparison between the proposed basis learning method and KSE 33
Figure 2.10 Training and testing error of different compression methods 34
Figure 3.1 Top-1 error *vs.* FLOPs and parameter compression ratio on MobileNets 38
Figure 3.2 The workflow of the proposed differentiable pruning method 39
Figure 3.3 Illustration of the hypernetwork designed for network pruning 42
Figure 3.4 Top-1 error vs. FLOP and parameter compression ratio on ResNet-164 and ResNet-110 55
Figure 3.5 Single image super-resolution visual results . . . 58
Figure 3.6 Image denoising visual results 59
Figure 4.1 The hinge between filter decomposition and filter pruning 62
Figure 4.2 The flowchart of the proposed algorithm 65
Figure 4.3 Group-sparsity regularization enforcement . . . 66
Figure 4.4 Comparison between KSE [95] and Hinge 78
Figure 4.5 Comparison between SSS [64] and the proposed method 79
Figure 4.6 Layer-wise compression ratio of ResNet56 on CIFAR10 81

Figure 5.1 Pipeline for identifying Layer-Wise Differentiated Network Architecture (LW-DNA) models . . 86
Figure 5.2 Illustration of the configuration space 89
Figure 5.3 Training and testing log of LW-DNA and baseline networks 98
Figure 5.4 Percentage of remaining output channels of LW-DNA models 101
Figure 5.5 The distribution of the latent vectors in MobileNetV2 102
Figure 5.6 Success plot on the LaSOT dataset for visual tracking 103
Figure 6.1 Acceleration performance of a representative GCN 110
Figure 6.2 Optimization of the computation procedure in a conventional GCN 112
Figure 6.3 Empirical weight distribution 118
Figure 6.4 Neighbor sampling 121
Figure 6.5 Qualitative result on Modelnet40 126
Figure 6.6 Renderings of input space and feature space . . 127
Figure 6.7 Visualization of the point distance 130
Figure 6.8 Surface reconstruction results 131

LIST OF TABLES

Table 2.1 Ablation study on compressing EDSR for image SR. . . . 28
Table 2.2 Comparison between Factor [147] and the proposed method . . . 28
Table 2.3 Compression results for EDSR . . . 29
Table 2.4 Ablation study on compressing DenseNet-12-40. 30
Table 2.5 Parameter compression results of networks trained on CIFAR10 . . . 32
Table 2.6 FLOPs compression results of networks trained on CIFAR10 . . . 32
Table 3.1 Ablation study on ResNet56 for CIFAR10 image classification . . . 51
Table 3.2 Comparison between ℓ_1 norm and ℓ_2 norm regularization . . . 52
Table 3.3 Results for CIFAR10 classification . . . 53
Table 3.4 Results for Tiny-ImageNet classification . . . 54
Table 3.5 Results on image super-resolution networks . . 56
Table 3.6 Results on image denoising networks . . . 59
Table 4.1 The solution to the proximal operator for ℓ_1, ℓ_{1-2}, $\ell_{1/2}$, and logsum regularizers . . . 74
Table 4.2 The regularization factor for ℓ_1, ℓ_{1-2}, $\ell_{1/2}$, and logsum . . . 76
Table 4.3 Ablation study . . . 77
Table 4.4 Comparison of CIFAR10 compression results . . 77
Table 4.5 Comparison of CIFAR100 compression results . 81
Table 5.1 Ablation study of the hyper-parameters ρ and τ on MobileNetV1 . . . 95
Table 5.2 Image classification results on ImageNet and Tiny-ImageNet . . . 96
Table 5.3 Image classification results on CIFAR . . . 97
Table 5.4 Tracking test results . . . 103
Table 5.5 Results on single image super-resolution networks 104
Table 6.1 Quantitative comparison for point cloud classification . . . 124

Table 6.2 Breakdown analysis of FLOPs of different networks 125
Table 6.3 Quantitative comparison for part segmentation of point clouds 128
Table 6.4 Comparison for semantic segmentation of point clouds 128
Table 6.5 Quantitative comparison for surface reconstruction 129

INTRODUCTION

1

INTRODUCTION

Computer vision is an essential sub-field of computer science that involves the automatic visual perception and understanding of the physical world via computers and machines. To achieve the goal of automation and intelligence, intensive computation is usually involved in the processing pipeline (*e.g.* encoding and decoding, data compression and decompression, domain transformation, feature extraction, regression and classification) of computer vision algorithms. Thus, it has been a core problem to improve the computational efficiency of algorithms in computer science and especially in computer vision. With the optimization and reduction of computation, massive application of some originally computationally intensive algorithms becomes feasible. For example, the well-know Fast Fourier Transform (FFT) [22] reduces the computational complexity of discrete Fourier transform from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$, which facilitates the wide application of FFT to various subdisciplines of engineering, science, mathematics, and music. The early research in computer vision focuses on edge and corner detectors (*e.g.* Canny detector, Harris detector) which from today's viewpoint are quite computationally efficient. More advanced yet complex feature descriptors are introduced in the later research works such as the well-known Scale-Invariant Feature Transform (SIFT) [105] and Histogram of Oriented Gradients (HOG) [26]. Due to the limitation of hardware, computational efficiency is still an important problem for the classical algorithms. For example, Speeded Up Robust Features (SURF) is several times faster than SIFT and also more robust [7].

Since the advent of deep learning era, Deep Neural Networks (DNNs) have achieved state-of-the-art performances in computer vision tasks including visual recognition [71, 136, 53, 62], object detection [38, 37, 129], semantic segmentation [104], image Super-Resolution (SR) [31, 68, 96], image denoising [169, 170], point cloud processing [120, 121] *etc.* Yet, the success of deep learning based vision algorithms comes with the overparameterization of DNNs and the explosion of computational complexity. And this leads to several problems.

First of all, the overparameterization of DNNs leads to huge model sizes with millions or even billions of parameters. Training such huge DNNs needs to store the parameters and the intermediate feature maps on computing devices such as Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU), which requires significant amount of computational memory. Secondly, the intensive computation slows down the inference speed of DNNs. Moreover, the computational resource on low-end devices such as smartphones and drones is limited. Thus, it is hard to deploy large DNNs to latency-demanding applications such as autonomous driving and low-end devices. Thirdly, the tuning of network architectures, training of the deep models, and inference during service time of DNNs consumes a considerable amount of energy [116, 138, 73]. And this indicates large carbon footprints of DNNs. Improving the computational efficiency and resource utilization efficiency of deep learning algorithms reduces the energy consumption of deep learning systems, which should be an important contributing factor to achieve the goal of carbon neutrality by the mid of this century. In conclusion, improving the computational efficiency of DNNs is beneficial not only from the technical perspective but also from the environmental perspective.

There are a couple of methods to improve the computational efficiency of deep neural networks. And one research direction is to design efficient network architectures. This direction has been thriving since a solution based on DNNs, *i.e.* AlexNet first outperformed the other competitors by a significant amount of margin in the ImageNet large scale visual recognition challenge in 2012 [71]. Later, a deep neural network architecture was proposed which further boosted the performance in the following series of the challenge [136]. Compared with VGG [136], the number of parameters in ResNet [53] and DenseNet [62] are reduced by one magnitude of order while comparable or even more accurate image classification results are achieved. Yet, those networks are still too bulky to be deployed on mobile devices. To overcome this problem, more efficient operations and architectures such as channel attention [60], group convolution [152], depth-wise convolution [59, 133, 58], and channel shuffle [172, 106] are proposed in the later works.

In addition to the manual design of network architectures, another research direction is Neural Architecture Search (NAS). The main drawback of earlier works is their almost insatiable demand for computational resources [182, 127] during the search phase. To alleviate the

computational burden, several methods are proposed including searching for a basic building cell [183, 99], differentiable search [100, 20, 154], weight sharing [100], and uniform sampling [44]. When a search space with efficient designs is adopted, searched networks can be highly efficient in number of parameters and computation [58, 140, 141].

A third research direction for efficient computation is network compression and model acceleration. Compared with the former two research direction, network compression aims at reducing the number of parameters and computational complexity of a given network, making the network suitable for fast inference without losing too much prediction accuracy of the given network. Thus, the trade-off between accuracy and compression rate should be always considered when designing network compression techniques.

There exists many network compression methods and they mainly fall into four categories including network quantization [124, 80, 24], network pruning [47, 56], and filter decomposition [65, 173], and knowledge distillation [57, 144]. Network quantization aims at representing the parameters in a neural network with lower bit width and reducing the model size of neural networks. With the technique, the precision of floating point numbers could be reduced from 32-bit to much lower bit width, and in the extreme case to binary and ternary states. Thus, the floating point operations could be conducted in a lower bit width or using integer-only arithmetic. In the case of binary and ternary quantization, multiplication could be converted to addition. Thus, network quantization could not only reduce the model size but also accelerate the inference of neural networks. Filter decomposition approximates the original filters in a network with a couple of low-rank matrices. Since parameters in the low-rank matrices is much fewer than the that in the original filters, filter decomposition is good at parameter saving in a network. In addition, the decomposed low-rank matrices can be converted to computational efficient operations such as group convolution, lightweight convolution, and linear combination in most cases. Thus, a reduction of computation could also be achieved. Network pruning attempts to remove the less important connections or neurons in a neural network. Thus, according to the pruned element (connections or neurons) in the network, network pruning could be classified into two categories including unstructured pruning (weight pruning) and structured pruning (channel pruning, filter pruning). Weight pruning leads to irregular kernel in the pruned model. Thus, to achieve the

actual acceleration, special hardware needs to be used. On the other hand, structured pruning could remove the entire channels in a layer of network. This leads to a pruned layer with fewer channels, smaller number of parameters, and reduced computation. Thus, structured pruning could can achieve real acceleration. Knowledge distillation is the process of transferring knowledge from a larger teacher model to a smaller student model. This is achieved by supervising the training of a student model with the logits of a pretrained teacher model or intermediate feature maps of the teacher model. Knowledge distillation could be combined with other network compression methods to boost the performance of the compressed network.

1.1 OVERVIEW

The aim of this thesis is improving the computational efficiency of neural networks. This problem is tackled from three different perspectives including neural network compression, neural architecture optimization, and computational procedure optimization. And the thesis is divided into three major parts accordingly.

1.1.1 *Neural Network Compression*

The first attempt of this thesis is to reduce the number of parameters of Convolutional Neural Networks (CNNs). Although CNN based solutions have achieved state-of-the-art performances in various computer vision tasks, the burden of huge parameter count is exaggerated due to the deepening and widening of the networks. Inspired by the fact that the large number of filters in CNNs constitutes a redundant set, a filter basis learning method is introduced in Chapter 2, with the aim of cutting down the number of filters and parameters in convolutional layers. With the basis learning method, the original filters in a convolutional layer can be decomposed to a set of basis and a set of linear combination coefficients. The original set of filters could be approximated by the linear combination of the set of filter basis. Then the decomposed basis and linear combination coefficients could be converted to a lightweight convolution and a linear layer implemented as 1×1 convolution. Different from the previous low-rank approximation methods, a channel split factor is used to balance the

distribution of parameter between the lightweight convolution and the 1×1 convolution. With the proposed channel split factor, a unified formulation of different filter decomposition methods is provided. The effectiveness of the proposed solution is validated on multiple CNN architectures for image classification and image super-resolution. The proposed method performs quite competitive with the other methods in terms of reduction of parameters and preservation of accuracy.

Then in Chapter 3, a differentiable network pruning method is proposed. With the advent of Automated Machine Learning (AutoML) and NAS, pruning has become intertwined with automatic mechanism and searching based architecture optimization. Yet, current automatic designs rely on either reinforcement learning or evolutionary algorithm. Due to the non-differentiability of those algorithms, the pruning algorithm needs a long searching stage before reaching the convergence. To circumvent this problem, a differentiable pruning method via hypernetworks is proposed for automatic network pruning. The specifically designed hypernetworks take latent vectors as input and generate the weight parameters of the backbone network. The latent vectors control the output channels of the convolutional layers in the backbone network and act as a handle for the pruning of the layers. By enforcing ℓ_1 sparsity regularization to the latent vectors and utilizing proximal gradient solver, sparse latent vectors can be obtained. Passing the sparsified latent vectors through the hypernetworks, the corresponding slices of the generated weight parameters can be removed, achieving the effect of network pruning. The latent vectors of all the layers are pruned together, resulting in an automatic layer configuration. Extensive experiments are conducted on various networks for image classification, single image super-resolution, and denoising. And the experimental results validate the proposed method.

In Chapter 2 and Chapter 3, the two major network compression techniques including low-rank approximation and filter pruning are studied. Then in Chapter 4, a unified analysis of the two techniques is provided from the perspective compact tensor approximation. Both of the methods approximate the weight tensor of CNNs with compact representations that keep the accuracy of the network. The core of the analysis is the introduced sparsity-inducing matrix that hinges filter pruning and decomposition. By simply changing the way the sparsity regularization is enforced to the sparsity-inducing matrix, filter pruning and low-rank decomposition can be derived accordingly. The

combination of the two techniques provides another flexible choice for network compression because the techniques complement each other. For example, in popular network architectures with shortcut connections (*e.g.* ResNet [53]), filter pruning cannot deal with the last convolutional layer in a ResBlock while the low-rank decomposition methods can. The compressed network is derived by solving an optimization problem with group sparsity regularization. And Proximal Gradient Descent (PGD) is used to solve the problem. Additionally, a couple of techniques are proposed including binary search, gradient based learning rate adjustment, layer balancing, and annealing methods to stabilize the optimization. Those detailed techniques are obtained by observing the influence of the proximal gradient method on the filter during the optimization. The proposed approach proves its potential as it compares favorably to the state-of-the-art on several benchmarks.

1.1.2 *Neural Architecture Optimization*

In addition to the neural network compression techniques, the second part of the thesis is dedicated to neural architecture optimization. The neural architecture optimization method in Chapter 5 is inspired by an interesting finding in Chapter 3 that slightly compressed networks can outperform the original network. Thus, Chapter 5 introduces a fine-grained architecture optimization method of given networks by adjusting the channel configurations. It is found that this adjustment can be achieved by shrinking widened baseline networks. Based on that, we articulate the "heterogeneity hypothesis": with the same training protocol, there exists a Layer-Wise Differentiated Network Architecture (LW-DNA) that can outperform the original network with regular channel configurations but with a lower level of model complexity.

The LW-DNA models are identified without extra computational cost or training time compared with the original network. The cost-free architecture optimization is advantageous compared with other method such as neural architecture optimization. On the other hand, the cost-free constraint leads to controlled experiments which direct the focus to the importance of layer-wise specific channel configurations. By observing the experimental results, the possible reason for the improved performance of an LW-DNA is directed to overfitting, *i.e.* the relative relationship between model complexity and dataset size. Experiments are conducted on various networks and datasets for image classification,

visual tracking and image restoration. The resultant LW-DNA models consistently outperform the baseline models.

1.1.3 *Computational Procedure Optimization*

In the third part of the thesis, we focus on the optimization of computational procedure in neural networks. Different from the previous chapters, in Chapter 6, a new family of neural networks, *i.e.* Graph Convolutional Networks (GCNs) is studied. In addition to CNNs for 2D vision tasks, GCNs also thrive for the processing of 3D representation such as point clouds. The computational procedure of the basic operation in GCNs, *i.e.* graph convolution is studied Chapter 6. Graph convolution is typically composed of a KNN search and a Multilayer Perceptron (MLP). By mathematically analyzing the operations there, two findings to improve the efficiency of GCNs are obtained. (1) The local geometric structure information of 3D representations propagates smoothly across the network that relies on KNN search to gather neighborhood features. This motivates the simplification of multiple KNN searches in GCNs. (2) Shuffling the order of graph feature gathering and an MLP leads to equivalent or similar composite operations. Based on the two findings, two methods are proposed to optimized the computational procedure in GCNs including KNN simplication and operation shuffling. A series of experiments show that the optimized networks have reduced computational complexity, decreased memory consumption, and accelerated inference speed while maintaining comparable accuracy for learning on point clouds.

Part I

NEURAL NETWORK COMPRESSION

2

LEARNING FILTER BASIS

2.1 INTRODUCTION

Current CNNs are composed of deep convolutional layers and each layer consists of multiple convolutional kernels. In networks such as AlexNet [71], VGG [136], and ResNet [53], there could be hundreds of or even thousands of filters in a convolutional layers. Thus, it is easy to imagine that there could be a lot of redundancies in convolutional layers. For example, in Fig. 2.1, the filters in a convolutional layer of a pretrained AlexNet is shown. It is obvious that many filters are very similar to each other. Thus, an interesting problem arises, *i.e.* whether it is possible to squeeze out the redundancy in CNNs and derive a much more efficient network. A natural solution is to represent the original convolutional filters by the linear combination of a smaller set of basis shown in Fig. 2.2. And this family of methods are referred to as filter decomposition or low-rank approximation since the original filters are decomposed into low-rank matrices. As shown in Fig. 2.3 and Subsec. 2.3.3, the decomposed basis and the linear combination coefficient could be converted a lightweight convolution and a 1×1 convolution.

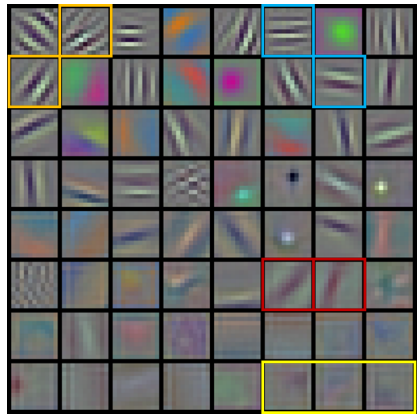


Figure 2.1: **Illustration of filters in AlexNet.**

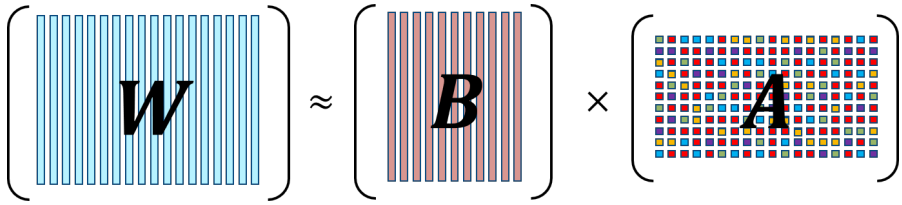
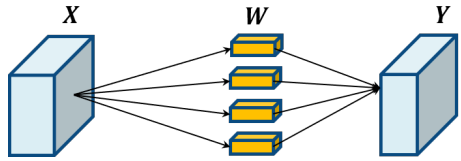
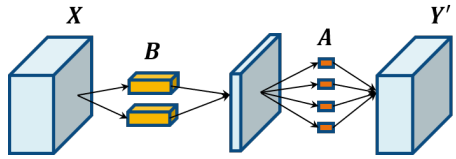


Figure 2.2: **Low-rank approximation of filters.**



(a) Convolution with original filters.



(b) Converted lightweight convolutions.

Figure 2.3: **Conversion from decomposed matrices to convolutions.**

In this chapter, a filter basis learning method is proposed with the main aim to reduce the number of parameters in CNNs. The proposed method is closely related to filter decomposition. Current filter decomposition methods either operate directly on the channel-wise 2D $w \times h$ filters [65, 147, 137] or decompose the intact 3D $c \times w \times h$ filters [173, 117]. For those working on 2D filters, considering that the kernel size is usually small (*e.g.* , 3×3) and a couple of new parameters are introduced to represent the 2D kernel, the compression ratio in terms of reduction of parameters is not impressive [147, 137]. The other filter decomposition methods [173] consider a 3D kernel as an intact element making impossible the reduction of the number of input channels [56]. This prevents the application of the method to narrow networks with much fewer output channels but more input channels. For example, in DenseNet-12-40, there are only 12 output channels which makes it not economic to decompose the 3D filters.

The aforementioned methods either collapse or maintain the 3D filters during decomposition, which can be regarded as 'hard' decomposition.

They are only coarse-grained configurations on the two boundary operating points. The motivation of the filter basis learning method is to provide the missing in-between fine-grained operating points and to balance the parameter distribution between the two decomposed convolutions. Thus, the proposed method circumvents the limitation of the 'hard' filter decomposition methods. We split the 3D filters along the input channel dimension and each split is considered as a basic element. We assume that the ensemble of those basic elements within one convolutional layer can be represented by the linear combinations of a basis. Our aim is to learn the basis and the linear combination together. During inference, the basis can be combined to reconstruct the original element, *i.e.* , the 3D split. Then the splits are stacked along the input channel dimension to form the original 3D filters. Also, as it will be explained in the following of the chapter, the convolutions with respect to the original 3D filters can be converted to convolutions with respect to the learned basis. Thus, our method can be easily and efficiently implemented and embedded into the state-of-the-art networks. Compared with previous works, our basis learning method also generalized easily to 1×1 convolution, which is vital for compressing networks with intensive 1×1 convolutions.

The contribution of this chapter is four-fold.

- A novel basis learning method is proposed. With the introduced channel split factor, a unified formulation of previous filter decomposition methods [65, 173, 147] is provided. By changing the channel split factor, different filter decomposition methods could derived. The proposed method achieves balanced distribution of parameters between the two decomposed matrices.

- By splitting filters along the channel dimension, the available number of filters (actually splits) increases. Thus, with the proposed method, it becomes feasible to compress networks whose output channel number is much smaller than the input channel number. The proposed method can be applied to convolutional layers with different kernel sizes and even 1×1 convolutions.

- The proposed method generalizes well on both high-level vision tasks and low-level vision tasks. Compared with the high-level vision tasks such as image classification where a single class is regressed, the low-level image SR task is more challenging

since the algorithm need to recover every pixel and the content detail in the image. However, none of the previous works apply compression method to networks designed for low-level task. Our experimental results show that network compression method also works well for image SR.

2.2 RELATED WORK

Network compression as a research topic has attracted an increased interest recently. The works in this field can be roughly grouped into four categories, namely, network pruning, network quantization, filter decomposition, and knowledge distillation.

2.2.1 *Network Pruning*

Non-structured pruning. To compress neural networks, network pruning disables the weak connections in a network that have a small influence on its prediction accuracy. Earlier pruning methods explore unstructured network weight pruning by deactivating connections corresponding to small weights or by applying sparsity regularization to the weight parameters [47, 98, 48]. The resulting irregular weight parameters of the network are not implementation-friendly, which hinders the real acceleration rate of the pruned network over the original one.

Structured Pruning. To overcome the problem of non-structured pruning, structured pruning is proposed to remove redundant channels in feature maps which result in regular kernel shapes and implementation-friendly algorithms [4, 178, 150]. Wen *et al.* [150] explored structured sparsity including channel-wise, shape-wise, and depth-wise sparsity in deep neural networks. He *et al.* [56] proposed channel pruning to accelerate deep neural networks. Their method can choose representative channels and prune redundant ones, based on LASSO regression.

2.2.2 *Network Quantization*

Network quantization aims at reducing the model size of neural networks by quantizing the weight parameters. Han *et al.* [47] demonstrated how to quantize weight parameters to a relatively small number of shared weights without loss of accuracy. Chen *et al.* [19] introduced

a hash function to group network connections into hash buckets and forced connections falling into the same buckets to share the same weight. Other works attempt to reduce the precision of parameter by introducing binary [124, 24, 23] and ternary [181] weights.

2.2.3 *Filter Decomposition*

Apart from the two aforementioned methods, filter decomposition is proposed to approximate the original filter with parameter efficient representations [29, 65, 74, 173, 147, 117]. Early low-rank approximation applies matrix decomposition by using SVD [29] or CP-decomposition [74]. Jaderberg *et al.* [65] proposed to approximate the 2D filter set by a linear combination of a smaller basis set of 2D separable filters. Wang *et al.* [147] built on the work of Jaderberg *et al.* and further rearranged the decomposed filter sequentially. In their work, each normal convolution is decomposed into several layers of depth-wise convolution followed by 1×1 convolution. Son *et al.* [137] proposed to use k-means algorithm to cluster the 3×3 convolutional kernels. The kernels that fall in the same cluster share the same weight parameter. However, for each 3×3 kernel, a scale and an index parameter is introduced to represent the kernel. So the compression ratio in terms of number of parameters is fixed and slightly larger than $2/9$. The same problem exists for [147]. Although the compression ratio $1/9$ could be achieved by [147], the classification accuracy is severely diminished. Another drawback of [137] is that it could not be applied to 1×1 convolutions favored by modern networks such as ResNet and DenseNet.

Instead of working on 2D filters as in the previous low-rank approximation, Zhang *et al.* [173] directly dealt with 3D filters by considering the input channel as the third dimension. However, their method cannot reduce the input channel. This prohibits the application of the decomposition method narrow networks with small output channel but large input channel such as DenseNet. In a recent work, Peng *et al.* [117] proposed to approximate a normal convolution by group convolution followed by a linear combination (1×1 convolution). However, they did not apply their approximation methods to DenseNet, which is of particular interest in the newly proposed architectures.

By contrast, our proposed basis learning method can be applied to convolutions with any kernel size and any input/output channel size. This makes our method flexible to compress different modern networks.

2.2.4 *Knowledge Distillation*

Knowledge distillation transfers the knowledge of a teacher network to a student network [57]. Current research in this direction focuses on the architectural design of the student network [25, 6] and the loss function [144]. It can make major modifications to the network components such that the student only needs to mimic some behaviors, *e.g.* , the intermediate feature or output of the original network.

2.3 FILTER DECOMPOSITION FOR NETWORK COMPRESSION

Given an input image $x \in \mathcal{X}$, the aim of supervised learning is to recover the corresponding label $y \in \mathcal{Y}$. For low-level vision tasks such as image SR, the label is the ground-truth high-resolution image corresponding to the low-resolution input image x. For high-level image classification, y is a class label of the image. The regression process can be represented by a simple function

$$\hat{y} = f_\Theta(x), \tag{2.1}$$

where $\hat{y}$ denotes the regressed label and $f_\Theta(\cdot)$ is the regression function of the neural network parameterized by Θ.

2.3.1 *Decomposing convolution layer with filter basis*

We assume that a convolution layer has c input channels and n output channels, and the kernel size is $w \times h$. In order to reduce the number of parameters in neural network, different decomposition methods have been suggested. Zhang *et al.* assumed the parameters of a convolution layer could be approximated by a low-rank matrix [173], *i.e.* ,

$$\mathbf{W} \approx \mathbf{B} \cdot \mathbf{A}, \tag{2.2}$$

where $\mathbf{W} \in \Re^{cwh \times n} = [\mathbf{W}_1, \cdots, \mathbf{W}_n]$ is the matrix that contains the vectorized 3D filters, the multiplication of matrix $\mathbf{B} \in \Re^{cwh \times m}$ and matrix $\mathbf{A} \in \Re^{m \times n}$ is a low-rank matrix with rank $m < n$. Besides formulating the parameters of convolution layer as a $cwh \times n$ matrix, there are also other low-rank approximation works [65, 147] which consider the parameter matrix as a $wh \times cn$ matrix. These works treat each channel in the 3D filter independently.

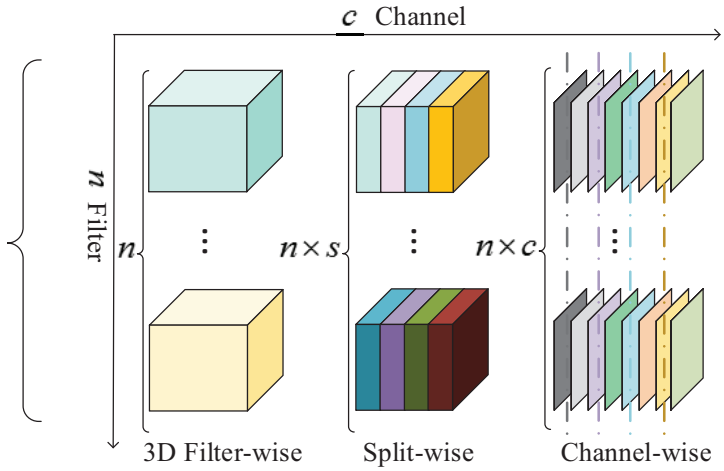


Figure 2.4: **Comparison of different filter decomposition methods.** *Right:* each channel of the 3D filter is considered as a basic element. A unique set of basis is learned for the n 2D filters in each channel. *Middle (the proposed):* the 3D filters are split into s groups along the channel dimension and each group is considered as a basic filter element. A basis set is learned for all of the $n \times s$ splits of all the 3D filters. *Left:* the 3D filter is considered as a whole. A basis set is learned for the 3D filters.

In Eqn. (2.2), the approximation of filter can also be analyzed in a filter basis decompostion perspective. Each 3D filter $\mathbf{W}_i \in \Re^{cwh \times 1}$ (or $\mathbf{W}_i \in \Re^{wh \times 1}$ for the channel-wise decomposition case) is represented by the linear combination of a set of m filter basis $\{\mathbf{B}_j | j = 1, \cdots, m\}$ with the coding coefficient vector $\mathbf{A}_i \in \Re^{m \times 1}$:

$$\mathbf{W}_i \approx \sum_{j=1}^{m} \alpha_{j,i} \mathbf{B}_j, i = 1, \cdots, n. \tag{2.3}$$

where $\mathbf{A}_i$ is the i-th column of $\mathbf{A}$, $\mathbf{B}_j$ is the j-th filter basis with dimension $cwh \times 1$ or $wh \times 1$ for the 3D filter-wise decomposition and 2D channel-wise decomposition cases, respectively. An illustration of direct 3D filter-wise decomposition and channel-wise filter decomposition can be found in the left and right part of Fig. 2.4.

From the viewpoint of filter basis decomposition, more flexible decomposition strategy can be adopted. In the next subsection, we analyze the relationship between the dimension of filter basis and compression rate, and suggest a split-wise decomposition approach for network compression.

2.3.2 *Compression rate with different filter basis*

If we utilize m 3D filter basis as a basic element (Fig 2.4: *Left*) to decompose the parameters of a convolution layer, the compression rate of the parameters is

$$\frac{m \cdot c \cdot w \cdot h + m \cdot n}{n \cdot c \cdot w \cdot h} = \frac{m}{n} + \frac{m}{c \cdot w \cdot h}, \tag{2.4}$$

where $(n \cdot c \cdot w \cdot h)$, $(m \cdot c \cdot w \cdot h)$, and $(m \cdot n)$ is the number of parameters of the original convolution layer, the filter basis, and the coding coefficients, respectively. In most of existing neural networks, $c \cdot w \cdot h$ is much larger than n. Thus, the first term in Eqn. (2.4) dominates the compression rate. For the 2D channel-wise decomposition case, we can similarly get the compression rate, namely,

$$\frac{m \cdot w \cdot h + c \cdot m \cdot n}{n \cdot c \cdot w \cdot h} = \frac{m}{n \cdot c} + \frac{m}{w \cdot h}. \tag{2.5}$$

The major storage budget is used for the coding coefficients.

In order to achieve a better trade-off between compressing the basis and coefficients, we split the 3D filters along the channel dimension by a channel split factor s as illustrated in the middle part of Fig. 2.4. That is, the $c \times w \times h$ filter is divided into s smaller splits and each split has p channels. In this way, the s smaller filters are stacked along the channel dimension to form the original filter and

$$c = s \cdot p. \tag{2.6}$$

As a result, the n 3D $c \times w \times h$ filters can be regarded as $n \cdot s$ filters with size $p \times w \times h$. Then, the problem becomes learning the basis and the representation coefficients of the $n \cdot s$ smaller filters. And the compression rate becomes

$$\frac{m \cdot p \cdot w \cdot h + m \cdot n \cdot s}{n \cdot c \cdot w \cdot h} = \frac{m}{n \times s} + \frac{m}{p \cdot w \cdot h}. \tag{2.7}$$

By adjusting the channel split factor s, Eqn. (2.7) generalizes easily to 3D filter-wise decomposition and channel-wise decomposition. When $s = 1$, the compression rate in Eqn. (2.4) could be derived. And when $s = c$, channel-wise decomposition in Eqn. (2.5) could be derived.

The compression rate equation in Eqn. (2.7) enable us to utilize generalized split-wise decomposition formula to achieve better compression rate. Concretely, the optimal compression rate with respect to the size of filter basis could be achieved by solving the following optimization problem:

$$\begin{aligned}\{s^*, p^*\} &= \arg\min_{\{s,p\}} \left\{ \frac{m}{n \times s} + \frac{m}{p \cdot w \cdot h} \right\} \quad \text{s.t.} \quad c = s \cdot p \\ &= \left\{ \sqrt{\frac{c \cdot w \cdot h}{n}}, \sqrt{\frac{n \cdot c}{w \cdot h}} \right\}. \end{aligned} \tag{2.8}$$

We can further quantize p to the nearest integer that can divide c. For most of the convolutional layers, the input channel c and output channel n are the same or of the same magnitude order, *i.e.* , $c \approx n$. Thus, the optimal group $s^* \approx \sqrt{w \times h}$. That is to say, the optimal configuration of splits is neither Fig. 2.4: *Left* nor Fig. 2.4: *Right* but the middle state between them.

2.3.3 *Implementing with convolution*

In this subsection, we show that filter decomposition can be implemented by convolution in the forward pass. By rearranging the operation, the proposed filter decomposition approach can not only compress network parameters but also alleviate the computation burden in the network.

We start with the case where there is only one split, *i.e.* , $s = 1$. As in Eqn. (2.3), we utilize linear combination of filter basis to reconstruct the 3D filter $\mathbf{W}_i = \sum_{j=1}^{m} \alpha_{j,i} \mathbf{B}_j$. For the simplicity of notation, we use the same notation to represent the original non-vectorized 3D filters and basis, *i.e.* , $\mathbf{W}_i, \mathbf{B}_j \in \Re^{c \times w \times h}$. Thus, the convolution between the input feature map x and the 3D kernel becomes

$$x * \mathbf{W}_i = x * \left(\sum_{j=1}^{m} \alpha_{j,i} \mathbf{B}_j \right) = \sum_{j=1}^{m} \alpha_{j,i} \left(x * \mathbf{B}_j \right). \tag{2.9}$$

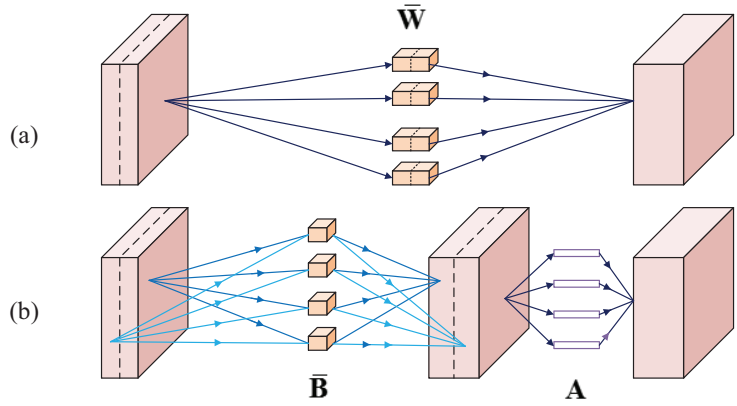


Figure 2.5: **Illustration of the proposed basis learning method.** Operations are converted to convolutions. Unlike the normal convolution, our method splits both the input feature map and the 3D filter along the channel dimension. A set of basis is learned for the ensemble of splits. Every split of the input feature map is convolved with the basis. A final 1×1 convolution generates the output.

The second equality follows the linearity of convolution. Eqn. (2.9) decompose the convolution operation with 3D filter $\mathbf{W}_i$ as linear combination of convolution operations with filter basis $\{\mathbf{B}_j, j = 1, \ldots, m\}$. The linear combination can be implemented by a 1×1 convolution.

For more general split-wise decomposition cases, we use smaller filter basis $\{\overline{\mathbf{B}}_j \in \Re^{p \times w \times h}, j = 1, \ldots, m\}$ to reconstruct each sub-part of the 3D filter, namely,

$$\mathbf{W}_i = \left[\overline{\mathbf{W}}_{i,1}; \ldots; \overline{\mathbf{W}}_{i,s}\right], \tag{2.10}$$

$$\overline{\mathbf{W}}_{i,g} = \sum_{j=1}^{m} \alpha_{j,i,g} \overline{\mathbf{B}}_j, \tag{2.11}$$

where $\overline{\mathbf{W}}_{i,g} \in \Re^{p \times w \times h}$ is a split of the 3D filter, $g = 1, \ldots, s$ is the split index, and $[\cdot]$ is the operator that stacks the basis along the channel dimension. Accordingly, the convolution between x and $\overline{\mathbf{W}}_i$ becomes

$$\begin{aligned} x * \mathbf{W}_i &= [\overline{x}_1, \cdots, \overline{x}_s] * \left[\overline{\mathbf{W}}_{i,1}, \cdots, \overline{\mathbf{W}}_{i,s}\right] \\ &= \sum_{g=1}^{s} \overline{x}_g * \overline{\mathbf{W}}_{i,g} = \sum_{g=1}^{s} \overline{x}_g * \sum_{j=1}^{m} \alpha_{j,i,g} \overline{\mathbf{B}}_j \\ &= \sum_{g=1}^{s} \sum_{j=1}^{m} \alpha_{j,i,g} \left(\overline{x}_g * \overline{\mathbf{B}}_j\right). \end{aligned} \tag{2.12}$$

where $\{\overline{x}_g, g = 1, \ldots, s\}$ in Eqn. (2.12) are the splits of the input x. As revealed by Eqn. (2.12), in the split-wise decomposition case, each split of feature map is firstly convolved with the filter basis, and then the final output is achieved by a weighted summation of the convolution results. This operation on feature map splits could be implemented as a 3D convolution as in Pytorch [115] or Tensorflow [1] with stride $p = c/s$ and no padding along the channel dimension. But we find the 3D convolution implementation is not efficient. In this way it takes 121 ms to run the compressed EDSR model for one iteration with batch size 16 and patch size 48×48. Instead, we implement the operation with s 2D convolutions that share the same weight parameter and the running time drops to 62 ms. The linear combination is again converted to a 1×1 convolution. Thus, no matter how many splits there are, a standard convolution can be decomposed into a convolution with respect to the basis and a 1×1 convolution. The implementation is illustrated in Fig. 2.5.

2.3.4 *Filter basis decomposition for special filter sizes*

As shown in the above analysis, our basis learning method follows a general setting of filter size, *i.e.* , $n \times c \times w \times h$. This means that the proposed basis learning method can be applied to any convolutional filters. Here we emphasize two special filter sizes.

1×1 **convolution:** The first one is 1×1 convolution which is favored by modern neural networks [53, 62]. When the input/output channels are quite large, considerable parameters and computation are consumed by 1×1 convolution. For example, in DenseNet-12-40 architecture [62], 12.1% of the parameters is in the two large 1×1 convolutions. Unfortunately, prior filter decomposition works [65, 29,

147] could not be applied to this kind of convolution. Following our formulation Eqn. (2.9) through Eqn. (2.12), a 1×1 convolution with large n and c can be decomposed into two cheaper ones.

$c \gg n > m$ convolution: In some networks such as DenseNet, the output channel n is much smaller than the input channel c. In this case, according to Eqn. (2.4), we are in the dilemma of either choosing a even smaller basis size m at the risk of losing too much accuracy or selecting an m comparable with n thus resulting in uneconomic compression. As revealed by Eqn. (2.5), by splitting the 3D kernels along the channel dimension, we can have s times more filters. So we can gracefully choose a comfortable basis size that leads to both economic compression and high accuracy.

2.4 LEARNING FILTER BASIS

In the previous section, we have shown that decomposing filter splits into linear combinations of filter basis could reduce the computational burden and parameter number of networks. In this section, we present our learning method for learning filter basis.

2.4.1 *General filter basis learning approach*

For the purpose of notation simplicity, we only introduce the simple case of using $\mathbf{B} \cdot \mathbf{A}$ to approximate filter $\mathbf{W}$. The training method for the split-wise case of approximating $\overline{\mathbf{W}}$ with $\overline{\mathbf{B}} \cdot \overline{\mathbf{A}}$ is exactly the same. We jointly minimize the approximation error $\|\mathbf{W} - \mathbf{B} \cdot \mathbf{A}\|_F^2$ and the network target loss $\mathcal{L}(y, f(x))$. For example, to compress image restoration network with mean square error (MSE) loss, our training objective function is

$$\min_{\mathbf{B}^l, \mathbf{A}^l} \left\| y - f_{\mathbf{B},\mathbf{A}|\Theta}(x) \right\|_F^2 + \gamma \sum_{l=1}^{L} \left\| \mathbf{W}^l - \mathbf{B}^l \cdot \mathbf{A}^l \right\|_F^2, \tag{2.13}$$

where $f_{\mathbf{B},\mathbf{A}|\Theta}(\cdot)$ denotes the CNN with parameter $\{\mathbf{B}, \mathbf{A}\}$ conditioned that the other parameters Θ are known and the superscript l indexes the l-th layer of an L-layer network.

After having learned the basis and the coding matrices $\{\mathbf{B}, \mathbf{A}\}$, there is no need to store the original filters. During inference, $\{\mathbf{B}, \mathbf{A}\}$ is used as the weight parameter as the lightweight and 1×1 convolution,

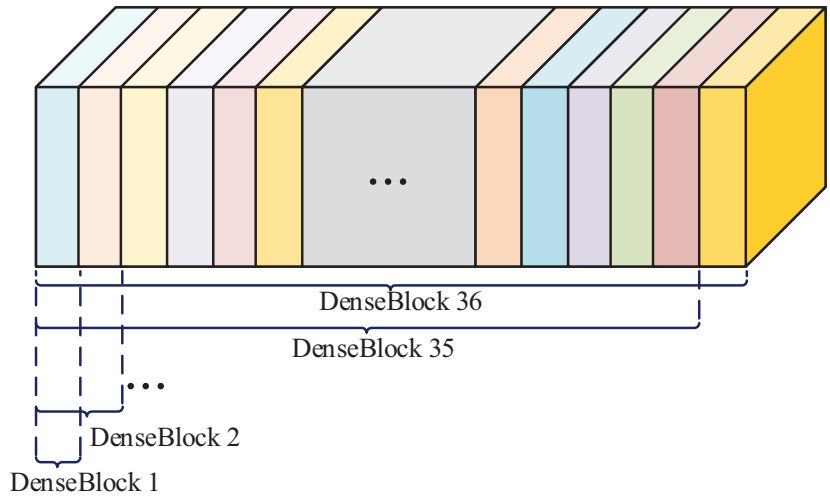


Figure 2.6: **Basis sharing for the compression of DenseNet-12-40.** The basis set is shared by all of the DenseBlocks in DenseNet-12-40. The shared basis is split into 36 splits. The basis of a certain DenseBlock is sliced from the shared basis set. Starting from the lower DenseBlock, every DenseBlock adds a new split from the shared basis set to the basis of the previous block forming the basis of current block. Thus, the basis channel of the DenseBlock grows gradually.

respectively. The total number of parameters of the basis and the coding coefficient is much fewer than those of the original filters, thus achieving a reduction of the number of parameters.

2.4.2 *Basis sharing*

To compress the networks further, we can force several or all convolutional layers to share the same basis set depending on the compression degree we want to achieve. The weight sharing strategy can be customized to the networks. For example, in ResNet [53] and the following works SRResNet [76], EDSR [96], there are two convolutions in the residual block. We can let the two convolutions share the basis. In ResNet-56 architecture for CIFAR10, the residual blocks are grouped into three groups, each with 9 residual blocks and increasing feature map channels. The channels in the lower residual block groups are

relatively small (16 and 32 for the first and second group). To achieve a satisfying compression rate, we have the convolutions within the same group share a common basis set. Moreover, in DenseNet, the input channel grows gradually with a step 12. There is no clear sign like in ResNet to indicate which convolution should share the basis. In this case, all of the convolutional layers share the same basis. For the lower layers, only a slice of the basis is used while only for the last convolutional layer the whole basis is used. The basis sharing strategy for DenseNet-12-40 is shown in Fig. 2.6.

In conclusion, we can apply block-wise, group-wise, or network-wise basis sharing flexibly according to the architecture of the target network.

2.5 EXPERIMENTAL RESULTS

2.5.1 *Experiment setup*

We show the experimental results in this section and compare with the state-of-the-art methods on both image classification and image SR. For classification, we applied our basis learning method to various networks including VGG [136], ResNet [53], and DenseNet [62]. We evaluate the performance of compressed models on CIFAR10 [70] dataset. The training and testing subset contains 50,000 and 10,000 images, respectively. As is done by prior works [53, 62], we normalize all images using channel-wise mean and standard deviation of the the training set. Standard data augmentation is also applied. We train the compressed networks for 300 epochs with Stochastic Gradient Descent (SGD) optimizer and an initial learning rate of 0.1. The learning rate is decayed by 10 after 50% and 75% of the epochs.

For image SR, we applied our method to two typical SR networks, namely, SRResNet [76] and EDSR [96]. SRResNet is a middle-level network with 1.5M parameters while EDSR is quite a huge network with 43M parameters but much higher Peak Signal-to-Noise Ratio (PSNR) accuracy. For fast training, we also compressed a lighter version of EDSR with 8 residual blocks and 128 channels per convolution in the residual block. We denote this network as EDSR-8-128. The networks are trained on DIV2K [2] dataset that contains 1,000 2K images. We test the networks on five datasets: Set5 [9], Set14 [166], B100 [108], Urban100 [63], and DIV2K validation set. Adam optimizer [69] is used for training SR networks. We use the default hyper-parameter. The

networks are trained for 300 epochs. The learning rate starts from 1×10^{-4} and decays by 10 after 200 epochs.

We reimplemented the network compression method Factor [147] and Group [117]. For the Factor method, to compare the methods fairly, we use two and three single intra-channel convolutional layers (SIC layer) [147] in Table 2.2, two SIC layers in Table 2.5 and Table 2.6, and one SIC layer in Table 2.3 to substitute one standard convolutional layer. To keep the number of parameter of the Group method [117] at the same level with other methods, the group size is set to 8, and 64 to approximate ResNet and VGG, respectively. To compress DenseNet, 3 groups are used for the first 20 DenseBlocks while 6 groups are used for the rest DenseBlocks.

2.5.2 *Validation on super-resolution*

The compression results of SR networks are shown in Table 2.1, Table 2.2, and Table 2.3. In Table 2.1, we explore different operating points applied to EDSR. We use 4 splits for the convolution in the residual block. For a clearer comparison, we report the number of parameters and compression ratio for one residual block since all of the other blocks has the same parameter. And we keep to this setting in Table 2.2 and Table 2.3. By default, each convolution layer uses an unique basis set, *i.e.* , without basis sharing. In Table 2.1 and Table 2.3, the corresponding basis sharing result is also shown.

In Table 2.1, there are several noticeable points. Firstly, the compressed models with and without basis sharing technique achieve almost the same PSNR for different m. But with basis sharing, the model size is further compressed. Secondly, when $m = 64$ and basis sharing is used, the compressed model only accounts for 9% of the parameters of the original network. When basis sharing is further used, an impressive compression rate of 5.9% is achieved while the PSNR result is not far away from the baseline. Thirdly, the most aggressive compression ratio is 1.5%. Considering that there are 32 and 16 residual blocks in EDSR and SRResNet respectively, this operating point brings the model size from EDSR level to SRResNet level while the PSNR of the resulting model is higher than that of SRResNet.

In Table 2.2, the results of Factor [147] and our basis learning method are shown for the SRResNet and EDSR-8-128. A lighter and a heavier operating point are reported for each compression method. The pro-

Metrics		Basis Share			Baseline
		No / Yes $m = 16$	No / Yes $m = 32$	No / Yes $m = 64$	
PSNR (dB)	Set5	32.14 / 32.16	32.22 / 32.20	32.33 / 32.30	32.48
	Set14	28.58 / 28.57	28.66 / 28.64	28.72 / 28.73	28.81
	B100	27.58 / 27.57	27.62 / 27.61	27.66 / 27.64	27.72
	Urban100	26.05 / 26.00	26.20 / 26.20	26.38 / 26.38	26.65
	DIV2K	28.96 / 28.93	29.06 / 29.04	29.14 / 29.14	29.25
#Params		27k / 17k	53k / 35k	106k / 70k	1180k
Comp. Ratio (%)		2.3 / 1.5	4.5 / 3.0	9.0 / 5.9	100

Table 2.1: **Ablation study on compressing EDSR for image SR.** The upscaling factor is $\times 4$. m is the number of basis. The number of splits p for a convolution is 4.

SRResNet [76]		Factor SIC2	Factor SIC3	Basis-64-14 (ours)	Basis-32-32 (ours)	Baseline
PSNR (dB)	Set5	31.68	31.86	31.84	31.90	32.03
	Set14	28.32	28.38	28.38	28.43	28.50
	B100	27.37	27.40	27.39	27.44	27.52
	Urban100	25.47	25.58	25.54	25.65	25.88
	DIV2K	28.59	28.65	28.63	28.69	28.85
#Parameters		19k	28k	18k	27k	74k
Comp. Rate (%)		25.3	38.0	24.3	36.1	100
EDSR-8-128 [96]		Factor SIC2	Factor SIC3	Basis-128-27 (ours)	Basis-128-40 (ours)	Baseline
PSNR (dB)	Set5	31.82	31.96	31.95	32.03	32.10
	Set14	28.40	28.47	28.42	28.45	28.55
	B100	27.43	27.49	27.46	27.50	27.55
	Urban100	25.63	25.81	25.76	25.81	26.02
	DIV2K	28.70	28.81	28.76	28.82	28.93
#Parameters		70k	105k	69k	102k	295k
Comp. Rate (%)		23.8	35.7	23.4	34.7	100

Table 2.2: **Comparison between Factor [147] and the proposed method.** The upscaling factor is $\times 4$. 'SIC*' denotes the number of SIC layers in Factor. 'Basis-N-S' means that the number of basis is S and each basis has N input channels.

Method	PSNR [dB]					#Param [k] / Ratio (%)
	Set5	Set14	B100	Urban100	DIV2K	
Upscaling Factor $\times 2$						
Factor [147]	37.95	33.53	32.15	31.99	34.60	136 /11.5
Basis-S (ours)	38.09	33.75	32.23	32.38	34.77	90 /7.6
Basis (ours)	38.12	33.72	32.27	32.46	34.84	164 /13.9
Baseline	38.19	33.95	32.35	32.97	35.03	1180 /100
Upscaling Factor $\times 3$						
Factor [147]	34.33	30.31	29.08	28.10	30.91	136 /11.5
Basis-S (ours)	34.47	30.41	29.15	28.39	31.06	90 /7.6
Basis (ours)	34.55	30.46	29.18	28.51	31.11	164 /13.9
Baseline	34.68	30.53	29.26	28.81	31.26	1180 /100
Upscaling Factor $\times 4$						
Factor [147]	32.05	28.54	27.55	25.98	28.92	136 /11.5
Basis-S (ours)	32.29	28.63	27.62	26.25	29.06	90 /7.6
Basis (ours)	32.39	28.69	27.64	26.36	29.13	164 /13.9
Baseline	32.48	28.81	27.72	26.65	29.25	1180 /100

Table 2.3: **Compression results for EDSR.** Basis-S uses basis sharing for the two convolutions within the same residual block. The number of basis in the proposed method is 32.

posed method outperforms Factor under the two settings. To further compare Factor and the proposed method, we apply the compression method to the fully-fledged EDSR model. As shown in Table 2.3, the compressed model Basis-S is much better than Factor and in the meanwhile with fewer parameters. The PSNR result of our Basis-S is slightly worse than that of Basis. The visual results for image super-resolution are shown in Fig. 2.8 and Fig. 2.7 for compressing SRResNet and EDSR-8-128 respectively. Compared with Factor [147] and Group [117], the SR images from our compressed model are very close to the baseline in terms of both visual quality and PSNR values.

2.5.3 *Validation on image classification*

In Table 2.4, we show different operating points for the proposed method applied on DenseNet-12-40 architecture. When the basis size increases from 24 to 32, the corresponding error rate decreases from

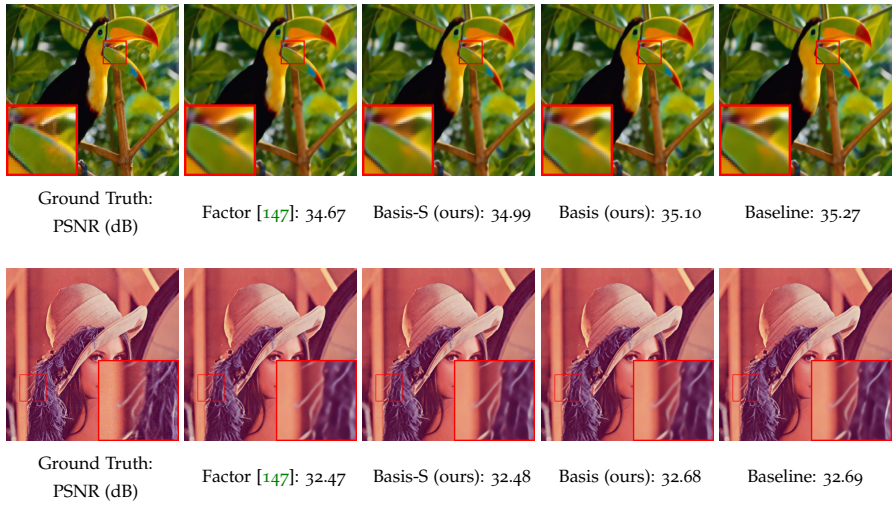


Figure 2.7: **Visual results for compressed EDSR models.** The upscaling factor $\times 4$.

Configuration	Top-1 Error (%)	#Params	Comp.(%)
M24	5.69	320k	30.8
M26	5.70	336k	32.3
M32	5.57	383k	36.8
M36T6	5.32	331k	31.8
M38T12	5.56	326k	31.3
Baseline	5.26	1041k	100

Table 2.4: **Ablation study on compressing DenseNet-12-40.** Top-1 error is reported for CIFAR10 image classification. "M*" means the number of basis in DenseBlock. "T*" means the number of splits in the transition layers.

5.69% to 5.57%. In addition, by applying compression to the 1×1 convolution in the transition layer, we can save some parameter budget for the DenseBlock, which is relatively more important in the network. Thus, for 'M36T6' and 'M38T12', we can utilize more basis and at the same time with smaller number of parameters. Compared with 'M32', 'M36T6' further reduces the error rate by 0.25%. Interestingly, although our 'M38T12' model uses two more basis than 'M36T6', the error rate rises a little bit. This is because 'M38T12' uses an aggressive compres-

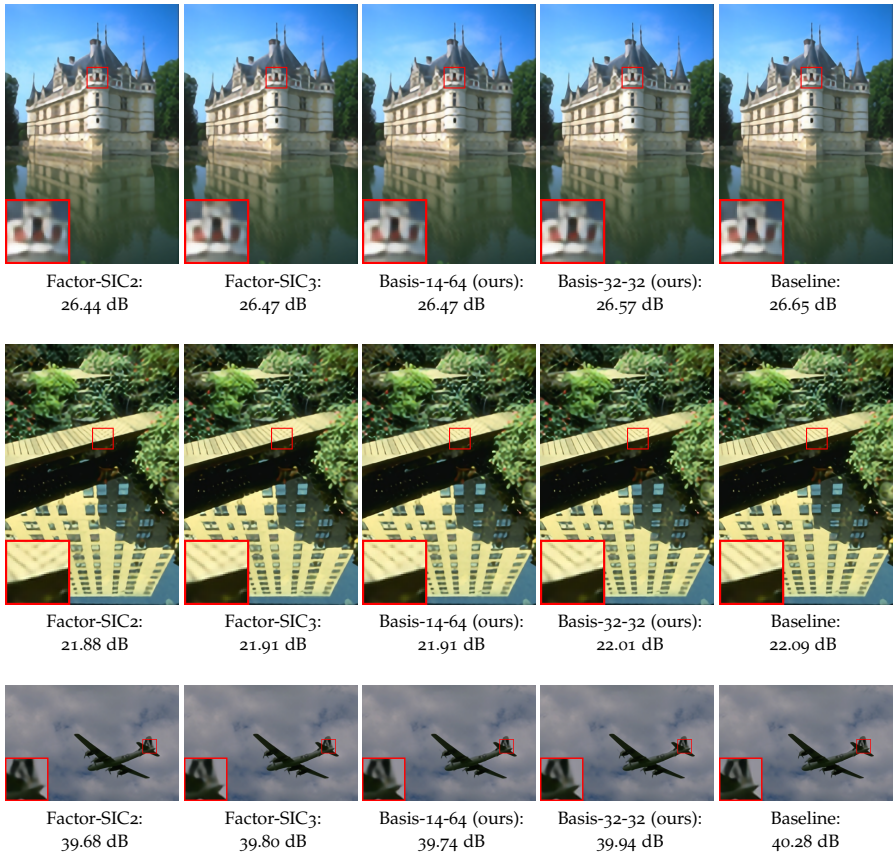


Figure 2.8: **Visual results for compressed SRResNet models.** The upscaling factor $\times 4$.

sion, *i.e.* , $s = 12$ in the transition block. Therefore, the compression degree of the DenseBlock and the transition block should be balanced to obtain the best trade-off between compression ratio and accuracy.

The compression results of different methods for VGG-16, DenseNet-12-40, and ResNet-56 are shown in Table 2.5. For a fair comparison, we follow the setting in [137] for VGG-16. That is, only one instead of three fully-connected layer is appended after the last pooling layer. On VGG-16, our method shows the most aggressive compression and the lowest error rate. For our compressed model, we only suffer 0.2% increase of error rate, which is quite small compared with 0.71% 1.14% increase of Group [117] and Factor. And our model has the smallest size. Our compression method and KSE [95] shoots the lowest error

Model	Method	Top-1 Error (%) / Baseline	#Param.	Comp. Rate(%)
VGG-16	K-means [137]	6.24 / 5.98	3.27M	22.2
	Factor [147]	7.12 / 5.98	3.34M	22.7
	Group [117]	6.69 / 5.98	3.80M	25.9
	Basis (ours)	6.18 / 5.98	3.21M	21.8
DenseNet-12-40	K-means	5.44 / 5.26	335k	32.2
	Factor	6.71 /5.26	317k	30.4
	Group	6.65 / 5.26	337k	32.4
	KSE [95]	5.30 / 5.19	390k	37.5
	[102](70%)	5.65 / 5.19	350k	33.6
	Simple-SVD	7.14 / 5.26	360k	34.6
	Basis (ours)	5.32 / 5.26	331k	31.8
ResNet-56	K-means [137]	6.76 / 6.28	190k	22.4
	Factor	8.70 / 6.28	212k	24.9
	Group	6.45 / 6.28	206k	24.3
	KSE	7.12 / 6.97	360k	42.4
	Basis (ours)	6.60 / 6.28	186k	21.9

Table 2.5: **Parameter compression results of networks trained on CIFAR10.** For a fair comparison, the model size from different methods is kept to the same level.

Model	Method	Top-1 Err.(%) / Baseline	FLOPs (%)
VGG	K-means [137]	6.24 / 5.98	100
	Factor [147]	7.12 / 5.98	36.6
	Group [117]	6.69 / 5.98	46.1
	Basis (ours)	**6.23** / 5.98	23.5
ResNet56	CaP [110]	6.78 / 6.49	50.2
	ENC [67]	7.00 / 6.90	50.0
	AMC [55]	8.10 / 7.20	50.0
	KSE [95]	6.77 / 6.97	48.0
	Basis (ours)	**6.08** / 7.05	50.0

Table 2.6: **FLOPs compression results of networks trained on CIFAR10.** For K-means, the practical FLOPs in the authors' code rather than the theoretical is reported.

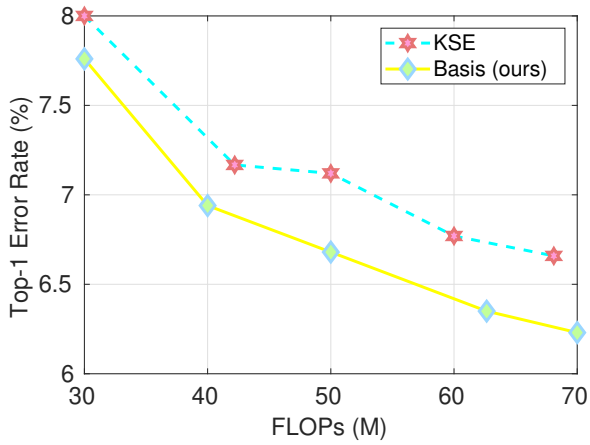


Figure 2.9: **Comparison between the proposed basis learning method and KSE.** Experiments are done on ResNet-56 trained CIFAR10.

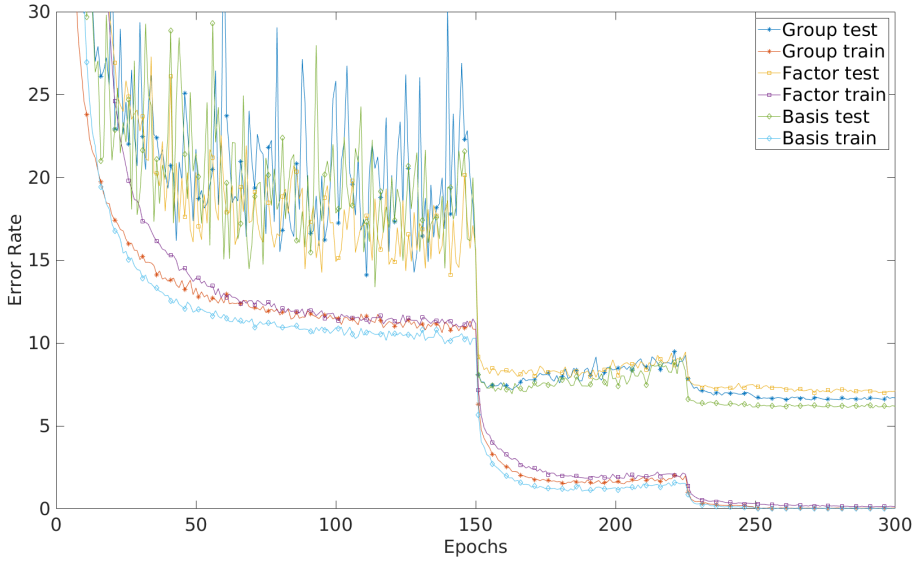
rate on DenseNet-12-40. As for the compression ratio, although Factor is slightly lower than ours, its accuracy is the worst among all the compared methods. For ResNet-56, our method performs comparable with Group in terms of accuracy while with 20k fewer parameters. Table 2.6 and Fig. 2.9 compare the computational cost of the proposed method and the state-of-the-art. Results are reported at operating points different from those in Table 2.5. For VGG-16, our method achieves the lowest error rate under the severest FLOPs reduction. For ResNet-56, the proposed method outperforms the others by a significant margin under the same FLOPs reduction. In Fig. 2.9, our method always shoots a lower error rate than KSE.

The error curves during training and testing for DenseNet-12-40, ResNet-56, and VGG-16 on CIFAR10 are shown in Fig. 2.10c, Fig 2.10b, and Fig 2.10a, respectively. Our method shoots the lowest stable error rate for all the three networks during training and testing.

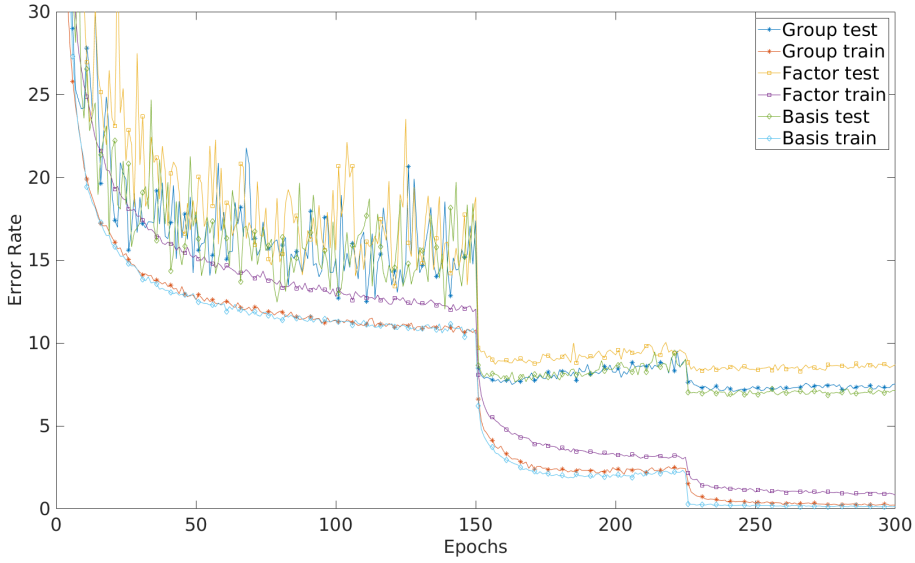
2.6 CONCLUSION

In this chapter, we explored how to learn a filter basis set for convolution operations in modern CNNs. Our method is not limited by the filter

size. Thus, it can be applied to 1×1 convolution and convolution with large input channel and smaller output channel. We applied our

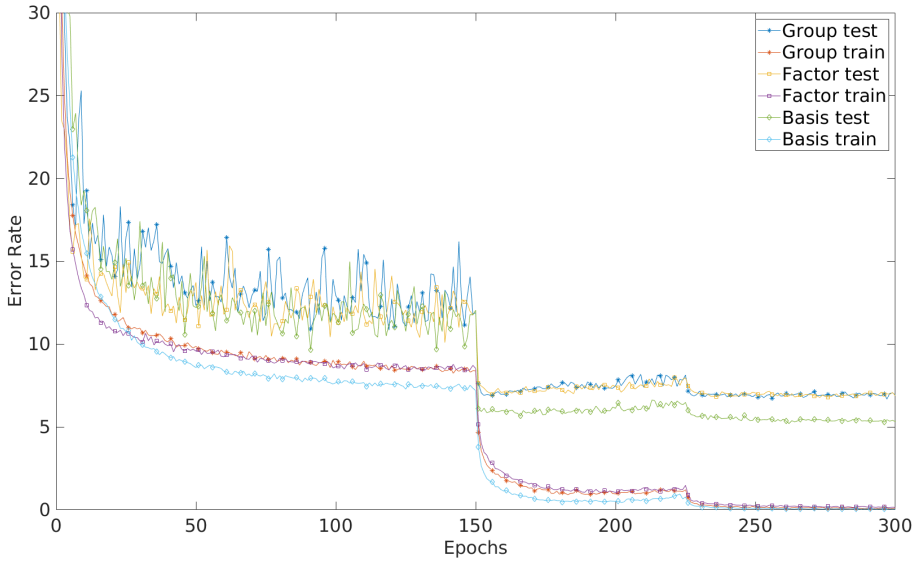


(a) VGG-16.



(b) ResNet-56.

Figure 2.10: Training and testing error of different compression methods.



(c) DenseNet-12-40.

Figure 2.10: Training and testing error of different compression methods (continued).

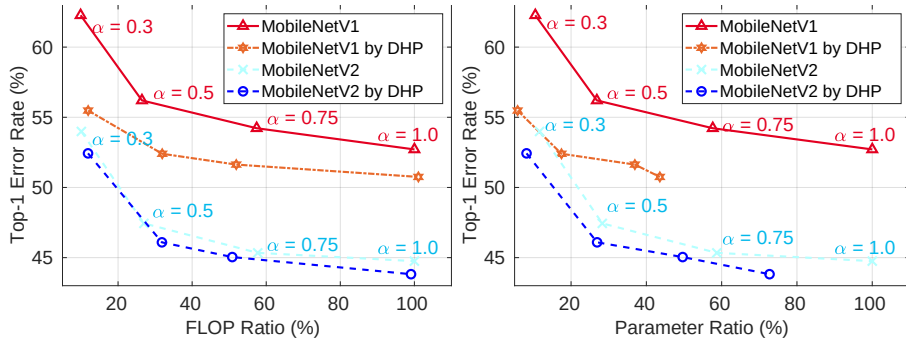
basis learning method to image classification and SR networks. The experiments validate the advantage of our basis learning method. Our compressed SRResNet and EDSR outperforms the models from the previous filter decomposition method. For the image SR network EDSR, the most aggressive compression our method brings the model size from EDSR level to SRResNet level while being more accurate than SRResNet. For VGG-16, the error rate of the model compressed by our method is within 0.2% the baseline error, which is much better than the results of other compression methods. Our filter basis learning method leads to state-of-the-art performance on ResNet and DenseNet.

3 DIFFERENTIABLE META PRUNING

3.1 INTRODUCTION

These days, network pruning has become the workhorse for network compression, which aims at lightweight and efficient model for fast inference [47, 56, 103, 89]. This is of particular importance for the deployment of tiny Artificial Intelligence (AI) algorithms on smart phones and edge devices [50]. Since the emerging of network pruning a couple of methods have been proposed based on the analysis of gradients, Hessians or filter distribution [75, 51, 32, 111, 54]. With the advent of AutoML and NAS [182, 17], a new trend of network pruning emerges, *i.e.* pruning with automatic algorithms and targeting distinguishing sub-architectures. Among them, reinforcement learning and evolutionary algorithm become the natural choice [55, 101]. The core idea is to search a certain fine-grained layer-wise distinguishing configuration among the all of the possible choices (population in the terminology of evolutionary algorithm). After the searching stage, the candidate that optimizes the network prediction accuracy under constrained budgets is chosen.

The advantage of these automatic pruning methods is the final layer-wise distinguishing configuration. Thus, hand-crafted design is no longer necessary. However, the main concern of these algorithms is the convergence property. For example, reinforcement learning is notorious for its difficulty of convergence under large or even middle level number of states [139]. Evolutionary algorithm needs to choose the best candidate from the already converged algorithm. But the dilemma lies in the impossibility of training the whole population till convergence and the difficulty of choosing the best candidate from unconverged population [101, 52]. A promising solution to this problem is endowing the searching mechanism with differentiability or resorting to an approximately differentiable algorithm. This is due to the fact that differentiability has the potential to make the searching stage efficient. Actually, differentiability has facilitated a couple of machine learning approaches and the typical one among them is NAS. Early works



(a) FLOPs compression ratio. (b) Parameter compression ratio.

Figure 3.1: **Top-1 error *vs.* FLOPs and parameter compression ratio on MobileNets.** The original model with different width multipliers α is set as the baseline. The DHP operating points near 100% FLOPs ratio is obtained by pruning the two networks with $\alpha = 2$. The DHP models outperforms the original at all the operating points.

on NAS have insatiable demand for computing resources, consuming tens of thousands of GPU hours for a satisfactory convergence [182, 183]. Differentiable Architecture Search (DARTS) reduces the insatiable consumption to tens of GPU hours, which has boosted the development of NAS during the past year [100].

Another noteworthy direction for automatic pruning is brought by MetaPruning [101] which introduces hypernetworks [45] into network compression. The output of the so-called hypernetwork is used as the parameters of the backbone network. During training, the gradients are also back-propagated to the hypernetworks. This method falls in the paradigm of meta learning since the parameters in the hypernetwork act as the meta-data of the parameters in the backbone network. But the problem of this method is that the hypernetworks can only output fixed-size weights, which cannot serve as a layer-wise configuration searching mechanism. Thus, a searching algorithm such as evolutionary algorithm is necessary for the discovery of a good candidate. Although this is quite a natural choice, there is still one interesting question, namely, whether one can design a hypernetwork whose output size depends on the input (termed as latent vector in this chapter) so that by only dealing with the latent vector, the backbone network can be automatically pruned.

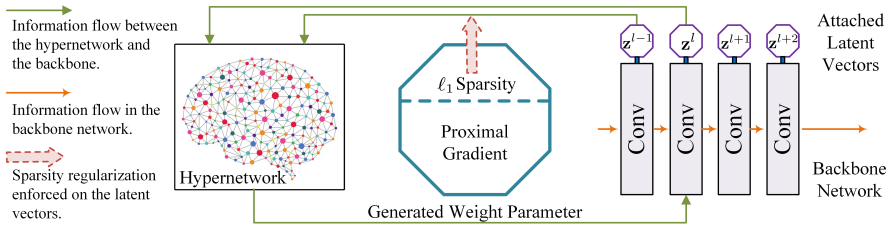


Figure 3.2: **The workflow of the proposed differentiable pruning method.** The latent vectors **z** attached to the convolutional layers act as the handle for network pruning. The hypernetwork takes two latent vectors as input and emits output as the weight of the backbone layer. ℓ_1 sparsity regularization is enforced on the latent vectors. The differentiability comes with the hypernetwork tailored to pruning and the proximal gradient exploited to solve problem. After the pruning stage, sparse latent vectors are obtained which result in pruned weights after being passed through the hypernetwork.

To solve the aforementioned problem, we propose the differentiable meta pruning approach via hypernetworks, *i.e.* Differentiable Hyper Pruning (DHP) shown in Fig. 3.2. A new design of hypernetwork is proposed to adapt to the requirements of differentiability. Each layer is endowed with a latent vector that controls the output channels of this layer. Since the layers in the network are connected, the latent vector also controls the input channel of the next layer. The hypernetwork takes as input the latent vectors of the current layer and previous layer that controls the output and input channels of the current layer respectively. Passing the latent vectors through the hypernetwork leads to outputs which are used as the parameters of the backbone network. To achieve the effect of automatic pruning, ℓ_1 sparsity regularizer is applied to the latent vectors. A pruned model is discovered by updating the latent vectors with proximal gradient. The searching stage stops when the compression ratio drops to the target level. After the searching stage, the latent vectors are sparsified. Accordingly, the outputs of the hypernetworks that are covariant with the latent vector are also compressed. The advantage of the proposed method is that it is only necessary to deal with the latent vectors, which automates network pruning without the other bells and whistles.

With the fast development of efficient network design and NAS, the usefulness of network pruning is frequently challenged. But by analyzing the performance on MobileNetV1 [59] and MobileNetV2 [133] in Fig. 3.1, we conclude that automatic network pruning is of vital importance for further exploring the capacity of efficient networks. Efficient network design and NAS can only result in an overall architecture with building blocks endowed with the same sub-architecture. By automatic network pruning, the efficient networks obtained by either human experts or NAS can be further compressed, leading to layer-wise distinguishing configurations, which can be seen as a fine-grained architecture search.

Thus, the contribution of this chapter is as follows.

I A new architecture of hypernetwork is designed. Different from the classical hypernetwork composed of linear layers, the new design is tailored to automatic network pruning. By only operating on the input of the hypernetwork, the backbone network can be pruned.

II A differentiable automatic networking pruning method is proposed. The differentiability comes with the designed hypernetwork and the utilized proximal gradient. It accelerates the convergence of the pruning algorithm.

III By the experiments on various vision tasks and modern CNNs [53, 62, 59, 133, 169, 131, 76, 96], the potential of automatic network pruning as fine-grained architecture search is revealed.

3.2 RELATED WORKS

3.2.1 *AutoML*

Recently, there is an emerging trend of exploiting AutoML for automatic network compression [55, 101, 52]. The rationality lies in the exploration among the total population of network configurations for a final best candidate. He *et al.* exploited reinforcement learning agents to prune the networks where hand-crafted design is not longer necessary [55]. Hayashi *et al.* utilized genetic algorithm to enumerate candidate in the designed hypergraph for tensor network decomposition [52]. Liu *et al.* trained a hypernetwork to generate weights of the backbone network

and used evolutionary algorithm to search for the best candidate [101]. The problem of these approachs is that the searching algorithms are not differentiable, which does not result in guaranteed convergence.

3.2.2 *Neural Architecture Search*

NAS automatizes the manual task of neural network architecture design. Optimally, searched networks achieve smaller test error, require fewer parameters and need less computations than their manually designed counterparts [182, 127]. But the main drawback of early strategies is their almost insatiable demand for computational resources. To alleviate the computational burden several methods [183, 99, 100] are proposed to search for a basic building block, *i.e.* cell, opposed to an entire network. Then, stacking multiple cells with equivalent structure but different weights defines a full network [119, 12]. Another recent trend in NAS is differentiable search methods such as DARTS [100]. The differentiability allows the fast convergence of the searching algorithm and thus boosts the fast development of NAS during the past year. In this chapter we propose a differentiable counterpart for automatic network pruning. Recent works try to push the frontier of NAS research by either redesigning the search space or proposing a more efficient search method [154, 132, 44, 20].

3.2.3 *Meta learning and hypernetworks*

Meta learning is a broad family of machine learning techniques that deal with the problem of learning to learn. An emerging trend of meta learning uses hypernetworks to predict the weight parameters in the backbone network [45]. Since the introduction of hypernetworks, it has found wide applications in NAS [12], multi-task learning [113], Bayesian neural networks [72], and also network pruning [101]. In this chapter, we propose a new design of hypernetwork which is especially suitable for network pruning and makes differentiability possible for automatic network pruning.

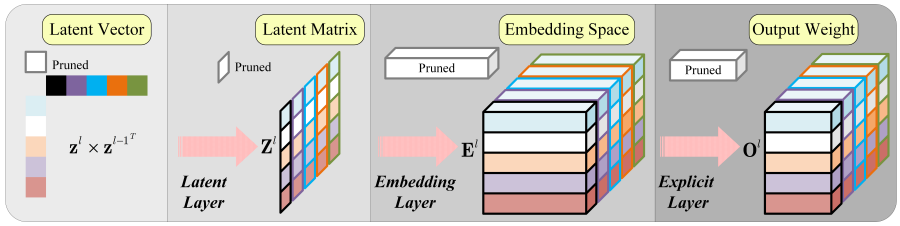


Figure 3.3: **Illustration of the hypernetwork designed for network pruning.** It generates a weight tensor after passing the input latent vector through the latent layer, the embedding layer, and the explicit layer. If one element in $\mathbf{z}^l$ is pruned, the corresponding slice of the output tensor is also pruned. See Subsec. 3.3.1 for details.

3.3 METHODOLOGY

The pipeline of the proposed method is shown in Fig 3.2. The two cores of the whole pipeline are the designed hypernetwork and the optimization algorithm. In the forward pass, the designed hypernetwork takes as input the latent vectors and predicts the weight parameters for the backbone network. In the backward pass, the gradients are back-propagated to the hypernetwork. The ℓ_1 sparsity regularizer is enforced on the latent vectors and proximal gradient is used to solve the problem. The dimension of the output of the hypernetwork is covariant with that of the input. Due to this property, the output weights are pruned along with the sparsified latent vectors after the optimization step. The differentiability comes with the covariance property of the hypernetworks, the ℓ_1 sparsity regularization enforced on the latent vectors, and the proximal gradient used to solve the problem. The automation of pruning is due to the fact that all of the latent vectors are non-discriminatively regularized and that proximal gradient discovers the potential less important elements automatically.

3.3.1 *Hypernetwork design*

Notation: Unless otherwise stated, we use the normal (x), minuscule bold ($\mathbf{z}$), and capital bold ($\mathbf{Z}$) letters to denote scalars, vectors, and matrices/high-dimensional tensors. The elements of a matrix/tensor

is indexed by the subscript as $\mathbf{Z}_{i,j}$ which could be scalars or vectors depending on the the dimension of the indexed subject.

We first introduce the design of the hypernetwork shown in Fig. 3.3. In summary, the hypernetwork consists of three layers. The latent layer takes as input the latent vectors and computes a latent matrix from them. The embedding layer projects the elements of the latent matrix to an embedding space. The last explicit layer converts the embedded vectors to the final output. This design is inspired by fully connected layers in [45, 101] but differs from those designs in that the output dimension is covariant with the input latent vector. This design is applicable to all types of convolutions including the standard convolution, depth-wise convolution, point-wise convolution, and transposed convolution.

Suppose that the given is an L-layer CNN. The dimension of the weight parameter of the l-th convolutional layer is $n \times c \times w \times h$, where n, c, and $w \times h$ denote the output channel, input channel, and kernel size of the layer, respectively. Every layer is endowed with a latent vector $\mathbf{z}^l$. The latent vector has the same size as the output channel of the layer, *i.e.* , $\mathbf{z}^l \in \mathbb{R}^n$. Thus, the previous layer is given a latent vector $\mathbf{z}^{l-1} \in \mathbb{R}^c$. The hypernetwork receives the latent vectors $\mathbf{z}^l$ and $\mathbf{z}^{l-1}$ of the current and the previous layer as input. A latent matrix is first computed from the two latent vectors, namely,

$$\mathbf{Z}^l = \mathbf{z}^l \cdot \mathbf{z}^{l-1^T} + \mathbf{B}_0^l, \tag{3.1}$$

where $[T]$ and $[\cdot]$ denote matrix transpose and multiplication, $\mathbf{Z}^l, \mathbf{B}_0 \in \mathbb{R}^{n \times c}$. Then every element in the latent matrix is projected to an m dimensional embedding space, namely,

$$\mathbf{E}_{i,j}^l = \mathbf{Z}_{i,j}^l \mathbf{w}_1^l + \mathbf{b}_1^l, i = 1, \cdots, n, j = 1, \cdots, c, \tag{3.2}$$

where $\mathbf{E}_{i,j}^l, \mathbf{w}_1^l, \mathbf{b}_1^l \in \mathbb{R}^m$. The vectors $\mathbf{w}_1^l$ and $\mathbf{b}_1^l$ are element-wise unique and for the simplicity of notation, the subscript $_{i,j}$ is omitted. $\mathbf{w}_1^l$, $\mathbf{b}_1^l$, and $\mathbf{E}_{i,j}^l$ can be aggregated as 3D tensors, namely $\mathbf{W}_1^l, \mathbf{B}_1^l, \mathbf{E}^l \in \mathbb{R}^{n \times c \times m}$. After the operation in Eqn. (3.2), the elements of $\mathbf{Z}^l$ are converted to embedded vectors in the embedding space. The final step is to obtain the output that can be explicitly used as the weights of the convolutional layer. To achieve that, every embedded vector $\mathbf{E}_{i,j}^l$ is multiplied by an explicit matrix, that is,

$$\mathbf{O}_{i,j}^l = \mathbf{w}_2^l \cdot \mathbf{E}_{i,j}^l + \mathbf{b}_2^l, i = 1, \cdots, n, j = 1, \cdots, c, \tag{3.3}$$

where $\mathbf{O}^l_{i,j}, \mathbf{b}^l_2 \in \mathbb{R}^{wh}$, $\mathbf{w}^l_2 \in \mathbb{R}^{wh \times m}$. Again, $\mathbf{w}^l_2$ and $\mathbf{b}^l_2$ are unique for every embedded vector and the subscript $_{i,j}$ is omitted. $\mathbf{w}^l_2$, $\mathbf{b}^l_2$, and $\mathbf{O}^l_{i,j}$ can also be aggregated as high-dimensional tensors, *i.e.* $\mathbf{W}^l_2 \in \mathbb{R}^{n \times c \times wh \times m}$ and $\mathbf{B}^l_2, \mathbf{O}^l \in \mathbb{R}^{n \times c \times wh}$. For the sake of simplicity, Eqn. (3.1), (3.2) and (3.3) can be abstracted as

$$\mathbf{O}^l = h(\mathbf{z}^l, \mathbf{z}^{l-1}; \mathbf{W}^l, \mathbf{B}^l), \tag{3.4}$$

where $h(\cdot)$ denotes the functionality of the hypernetwork. The final output $\mathbf{O}^l$ is used as the weight parameter of the l-th layer. The output $\mathbf{O}^l$ is covariant with the input latent vector because pruning an element in the latent vector removes the corresponding slice of the output $\mathbf{O}^l$ (See Fig. 3.3).

When designing the hypernetwork, we tried to add batch normalization and non-linear layers after the linear operation in Eqn. (3.2) and Eqn. (3.3). But it did not lead to clearly better results. Thus, we just kept to the simple design. This is also consistent with the previous designs [45, 101].

3.3.2 *Sparsity regularization and proximal gradient*

The core of differentiability comes with not only the specifically designed hypernetwork but also the mechanism used to search the the potential candidate. To achieve that, we enforce sparsity constraints to the latent vectors. The loss function of the aforementioned L-layer CNN is denoted as

$$\min_{\mathbf{W},\mathbf{B},\mathbf{z}} \mathcal{L}\Big(\mathbf{y}, f\big(\mathbf{x}; h(\mathbf{z}; \mathbf{W}, \mathbf{B})\big)\Big) + \gamma\mathcal{D}(\mathbf{W}) + \gamma\mathcal{D}(\mathbf{B}) + \lambda\mathcal{R}(\mathbf{z}), \tag{3.5}$$

where $\mathcal{L}(\cdot,\cdot)$, $\mathcal{D}(\cdot)$, and $\mathcal{R}(\cdot)$ are the loss function for a specific vision task, the weight decay term, and the sparsity regularization term, γ and λ are the regularization factors. For the simplicity of notation, the superscript l is omitted. The sparsity regularization takes the form of ℓ_1 norm, namely,

$$\mathcal{R}(\mathbf{z}) = \sum_{l=1}^{L} \|\mathbf{z}^l\|_1. \tag{3.6}$$

To solve the problem in Eqn. (3.5), the weights $\mathbf{W}$ and and biases $\mathbf{B}$ of the hypernetworks are updated with SGD. The gradients are back-propagated from the backbone network to the hypernetwork. Thus,

neither the forward nor the backward pass challenges the information flow between them. The latent vectors are updated with proximal gradient algorithm, *i.e.* ,

$$\mathbf{z}[k+1] = \mathbf{prox}_{\lambda\mu\mathcal{R}}\Big(\mathbf{z}[k] - \lambda\mu\nabla\mathcal{L}\big(\mathbf{z}[k]\big)\Big), \tag{3.7}$$

where μ is the step size of proximal gradient and is set as the learning rate of SGD updates. As can be seen in the equation, the proximal gradient update contains a gradient descent step and a proximal operation step. When the regularizer has the form of ℓ_1 norm, the proximal operator has closed-form solution, *i.e.*

$$\mathbf{z}[k+1] = \mathrm{sgn}\Big(\mathbf{z}[k+\Delta]\Big)\Big[\big|\mathbf{z}[k+\Delta]\big| - \lambda\mu\Big]_+, \tag{3.8}$$

where $\mathbf{z}[k+\Delta] = \mathbf{z}[k] - \lambda\mu\nabla\mathcal{L}(\mathbf{z}[k])$ is the intermediate SGD update, the sign operator $\mathrm{sgn}(\cdot)$, the thresholding operator $[\cdot]_+$, and the absolute value operator $|\cdot|$ act element-wise on the vector. Eqn. (3.8) is the soft-thresholding function.

The latent vectors first get SGD updates along with the other parameters $\mathbf{W}$ and $\mathbf{B}$. Then the proximal operator is applied. Due to the use of SGD updates and the fact that the proximal operator has closed-form solution, we recognize the whole solution as approximately differentiable (although the ℓ_1 norm is not differentiable at 0), which guarantees the fast convergence of the algorithm compared with reinforcement learning and evolutionary algorithm. The speed-up of proximal gradient lies in that instead of searching the best candidate among the total population it forces the solution towards the best sparse one.

The automation of pruning follows the way the sparsity applied in Eqn. (3.6) and the proximal gradient solution. First of all, all latent vectors are regularized together without distinguishment between them. During the optimization, information and gradients flows fluently between the backbone network and the hypernetwork. The proximal gradient algorithm forces the potential elements of the latent vectors to approach zero quicker than the others without any human effort and interference in this process. The optimization stops immediately when the target compression ratio is reached. In total, there are only two additional hyper-parameters in the algorithm, *i.e.* the sparsity regularization factor and the mask threshold τ in Subsec. 3.3.3. Thus, running the algorithm is just like turning on the button, which enable the application of the algorithm to all of the CNNs without much interference of domain experts' knowledge.

3.3.3 *Network pruning*

Different from the fully connected layers, the proposed design of hypernetwork can adapt the dimension of the output according to that of the latent vectors. After the searching stage, sparse versions of the latent vectors are derived as $\overline{\mathbf{z}}^{l-1}$ and $\overline{\mathbf{z}}^{l}$. For those vectors, some of their elements are zero or approaching zero. Thus, 1-0 masks can by derived by comparing the sparse latent vectors with a predefined small threshold τ, *i.e.* $\mathbf{m}^{l} = \mathcal{T}\left(\overline{\mathbf{z}}^{l}, \tau\right)$, where the function $\mathcal{T}(\cdot)$ element-wise compares the latent vector with the threshold and returns 1 if the element is not smaller than τ and 0 otherwise. Then latent vector $\overline{\mathbf{z}}^{l}$ is pruned according to the mask $\mathbf{m}^{l}$. See Subsec. 3.3.6 for more analysis.

3.3.4 *Latent vector sharing*

Due to the existence of skip connections in residual networks such as ResNet, MobileNetV2, SRResNet, and EDSR, the residual blocks are interconnected with each other in the way that their input and output dimensions are related. Therefore, the skip connections are notoriously tricky to deal with. But back to the design of the proposed hypernetwork, a quite simple and straightforward solution to this problem is to let the hypernetworks of the correlated layers share the same latent vector. Note that the weight and bias parameters of the hypernetworks are not shared. Thus, sharing latent vectors does not force the the correlated layers to be identical. By automatically pruning the single latent vector, all of the relevant layers are pruned together. Actually, we first tried to use different latent vectors for the correlated layers and applied group sparsity to them. But the experimental results showed that this is not a good choice because this strategy shot lower accuracy than the latent vector sharing strategy.

Basic criteria. In the following, we first describe the general rules for latent vector sharing and then detail the specific rules for special network blocks.

I Every convolutional layer is attached a latent vector.

II The channel that the latent vector controls and the dimension of the latent vector vary with the types of convolutional layers.

a) For standard convolution, point-wise convolution and transposed convolution, the latent vector controls the output channel of the layer and the dimension of the latent vector is the same as the number of output channels.

b) For depth-wise convolution and group convolution, the latent vector controls the input channels per group. The dimension of the latent vector is the same as the number of input channels per group. That is, the latent vector of depth-wise convolution contains only one element.

III The latent vectors are shared among consecutive layers. This is because the output and input channels of consecutive layers are correlated. Thus, the hypernetworks receive the latent vectors of the previous layer and the current layer as input.

IV Not every latent vector needs to be sparsified. The latent vectors free from sparsification are list as follows.

a) The latent vector that controls the input channel of the first convolutional layer. This latent vector has the same dimension with the input image channels, *e.g.* 3 for RGB images and 1 for gray images. Of course, the input images do not need to be pruned.

b) The latent vector attached to depth-wise convolution and group convolution. This latent vector controls the input channels per group. To compress depth-wise and group convolution, the number of groups is reduced, which is controlled by the latent vectors of the previous layer.

Residual block. The residual networks including ResNet, SRResNet, and EDSR are constructed by stacking a number of residual blocks. Depending on the dimension of the feature maps, the residual networks contain several stages with progressively reducing feature map dimension and increasing number of feature maps. (Note that the feature map dimension of EDSR and SRResNet does not change for all of the residual blocks. So there is only one stage for those networks.) For the residual blocks within the same stage, their output channels are correlated due to the existence of the skip connections. In order to prune the second convolution of the residual blocks within the same stage, we use a shared latent vector for them. Thus, by only dealing

with this shared latent vector, all of the second convolutions of the residual blocks can be pruned together. Please refer to Table 3.1 for the ablation study on latent vector sharing and non-sharing strategies.

Dense block. Similar to residual networks, DenseNet also contains several stages with different feature map configurations. But different from residual networks, each dense block concatenates its input and output to form the final output of the block. As a result, each dense block receives as input the outputs of all of the previous dense blocks within the same stage. Thus, the hypernetwork of a dense block also has to receive the latent vectors of the corresponding dense blocks as input.

Inverted residual block. The inverted residual blocks are just a special case of residual blocks. So how the latent vectors are shared across different blocks is the same with the normal residual blocks. Here we specifically address the sharing strategy within the block due to the existence of depth-wise convolution. The inverted residual block has the architecture of "point-wise conv + depth-wise conv + point-wise conv". As explained earlier, the latent vector of depth-wise convolution controls the input channels per group. Thus, the latent vector of the first point-wise convolution controls not only its output channels but also the input channels of the depth-wise convolution and the input channels of the second point-wise convolution. Thus, this latent vector has to be passed to the hypernetworks of the those convolutional layers.

Upsampler of super-resolution networks. The image super-resolution networks are attached with upsampler blocks at the tail of the networks to increase the spatial resolution of the feature map. For the scaling factor of $\times 4$, two upsamplers are attached and each doubles the spatial resolution. Each of the upsampler block contains a standard convolutional layer that increases the number of feature maps by a factor of 4 and a pixel shuffler that shuffles every 4 consecutive feature maps into the spatial dimension. Thus, the output channel of the convolutional layer in the upsmapler is correlated to its input channel. If one input channel is pruned, then four corresponding consecutive output channels should also be pruned. To achieve this control of pruning, a common latent vector is used for the input and output channels. The dimension of this latent vector is the same with the input channel size. This vector is repeated and interleaved to form the one controlling the output channel.

3.3.5 *Discussion on the convergence property*

Compared with reinforcement learning and evolutionary algorithm, proximal gradient may not be the optimal solution for some problems. But as found by previous works [103, 101], automatic network pruning serves as an implicit searching method for the channel configuration of a network. In addition, the network is searched and trained from scratch in this chapter. The important factor is the number of remaining channels of the convolutional layers in the network. Thus, it is relatively not important which filter is pruned as long as the number of pruned channels are the same. This reduces the number of possible candidates by orders of magnitude. In this case, proximal gradient works quite well.

3.3.6 *Implementation consideration*

Compact representation of the hypernetwork. Thanks to the default tensor operations in deep learning toolboxes [115], the operations in the hypernetwork could be represented in a compact form. The embedding operation in Eqn. (3.2) can be written as the following high-dimensional tensor operation

$$\mathbf{E}^l = \mathcal{U}^3\left(\mathbf{Z}^l\right) \circ \mathbf{W}_1^l + \mathbf{B}_1^l, \tag{3.9}$$

where $\mathcal{U}^3(\mathbf{Z}^l) \in \mathbb{R}^{n\times c\times 1}$, $[\circ]$ denotes the broadcastable element-wise tensor multiplication, $\mathcal{U}^3(\cdot)$ inserts a third dimension for $\mathbf{Z}^l$. The operation in Eqn. (3.3) can be easily rewritten as batched matrix multiplication,

$$\mathbf{O}^l = \mathbf{W}_2^l * \mathbf{E}^l + \mathbf{B}_2^l, \tag{3.10}$$

where $[*]$ denotes batched matrix multiplication.

Pruning analysis. Analyzing the three layers of the hypernetworks together with the masked latent vectors leads to a direct impression on how the backbone layers are automatically pruned. That is,

$$\begin{aligned}
\overline{\mathbf{O}}^l &= \mathbf{W}_2^l * \left[\mathcal{U}^3\left((\mathbf{m}^l \circ \overline{\mathbf{z}}^l) \cdot (\mathbf{m}^{l-1} \circ \overline{\mathbf{z}}^{l-1})^T\right) \circ \mathbf{W}_1^l\right] && (3.11)\\
&= \mathbf{W}_2^l * \left[\mathcal{U}^3(\mathbf{m}^l \cdot \mathbf{m}^{l-1^T}) \circ \mathcal{U}^3(\overline{\mathbf{z}}^l \cdot \overline{\mathbf{z}}^{l-1^T}) \circ \mathbf{W}_1^l\right] && (3.12)\\
&= \mathcal{U}^3(\mathbf{m}^l \cdot \mathbf{m}^{l-1^T}) \circ \left[\mathbf{W}_2^l * \left(\mathcal{U}^3(\overline{\mathbf{z}}^l \cdot \overline{\mathbf{z}}^{l-1^T}) \circ \mathbf{W}_1^l\right)\right] && (3.13)
\end{aligned}$$

The equality follows the broadcastability of the the operations $[\circ]$ and $[*]$. As shown in the above equations, applying the masks on the latent vectors has the same effect of applying them on the final output. Note that in the above analysis the bias terms $\mathbf{B}_0^l$, $\mathbf{B}_1^l$, and $\mathbf{B}_2^l$ are omitted since they have a really small influence on the output of the hypernetwork. In conclusion, the final output can be pruned according to the same criterion for the latent vectors.

Initialization of the hypernetwork. All biases are initialized as zero, the latent vector with standard normal distribution, and $\mathbf{W}_1^l$ with Xaiver uniform [39]. The weight of the explicit layer $\mathbf{W}_2^l$ is initialized with Hyperfan-in which guarantees stable backbone network weights and fast convergence [16].

3.4 EXPERIMENTAL RESULTS

To validate the proposed method, extensive experiments were conducted on various CNN architectures including ResNet [53], DenseNet [62] for CIFAR10 [70] image classification, MobileNetV1 [59], MobileNetV2 [133] for Tiny-ImageNet [28] image classification, SRResNet [76], EDSR [96] for single image super-resolution, and DnCNN [169], UNet [131] for gray image denoising. The proposed DHP algorithm starts from a randomly initialized network with the initialization method detailed in Subsec. 3.3.6. After pruning, the training of the pruned network continues with the same training protocol used for the original network. All of the experiments are conducted on NVIDIA TITAN Xp GPUs.

Hyperparameters. The proposed DHP method does not rely on the pretrained model. Thus, all of the networks are trained and pruned from scratch. The hypernetworks are first randomly initialized. The parameter space of hypernetwork increases in proportion to the dimension m of the embedding space. Thus, m should not be too large. m is set to 8 in the experiments. Proximal gradient is used to sparsify the latent vectors. The step size μ of the proximal operator is set as the learning rate of SGD updates. The sparsity regularization factor λ is set by empirical studies. The value is chosen such that the searching epochs constitute arounds 5% – 10% of the whole training epochs. This guarantees acceptable convergence during searching while not introducing too much additional computation. A target FLOPs compression ratio is set for the pruning algorithm. When the difference between the target compression ratio and the actual compression ratio falls below

Share	R	λ	τ	Target FLOPs Ratio (%)	Actual FLOPs Ratio (%)	Actual Params Ratio (%)	Top-1 Err. (%)
Yes	ℓ_1	2^{-4}	5^{-3}	38	39.96	52.49	7.41
No	$\ell_{2,1}$	2^{-4}	5^{-3}	38	39.40	54.24	7.91
Yes	ℓ_1	3^{-4}	5^{-3}	38	39.60	49.00	6.86
No	$\ell_{2,1}$	3^{-4}	5^{-3}	38	39.54	54.04	7.03
Yes	ℓ_1	2^{-4}	5^{-3}	50	51.27	56.84	7.13
No	$\ell_{2,1}$	2^{-4}	5^{-3}	50	50.96	64.05	6.85
Yes	ℓ_1	3^{-4}	5^{-3}	50	51.68	57.74	6.52
No	$\ell_{2,1}$	3^{-4}	5^{-3}	50	50.18	59.15	6.74

Table 3.1: **Ablation study on ResNet56 for CIFAR10 image classification**: exploring latent vector sharing strategy among correlated convolutional layers. "Share" denotes whether the latent vector sharing strategy described in Subsec. 3.3.4 is adopted. The $\ell_{2,1}$ regularizer means that when the latent vectors are not shared among the correlated layers, the group sparsity regularizer $\ell_{2,1}$ is enforced on their latent vectors. Otherwise, the normal ℓ_1 sparsity regularizer is used.

2%, the automatic pruning procedure stops. Then the pruned latent vectors as well as the pruned outputs of the hypernetworks are derived. After that, the outputs of hypernetworks are used as the weight parameters of the backbone network and updated by SGD or Adam algorithm directly. After the pruning procedure, the hypernetworks are removed. The training continues and the training protocol are the same as that used for training the original network. The number of pruning epochs is much smaller than that used for training the original network.

3.5 ABLATION STUDY

The ablation study on the latent vector sharing strategy is shown in Table 3.1. As shown in the table, the latent vector sharing strategy outperforms the non-sharing strategy consistently except for case $\lambda = 2^{-4}$, $\tau = 5^{-3}$ and 50% target FLOPs compression ratio. The inconsistency is largely due to the gap between the actual parameter compression ratio

of different strategies. Due to this fact, various latent vector sharing rules are developed for easier and better automatic network pruning.

ℓ_1 regularizer is mainly used to sparsify the the latent vectors. We tried to replace ℓ_1 regularizer with ℓ_2 regularizer and kept the same pruning strategy. The experiments are conducted on ResNet. The comparison results are shown in Table 3.2. Compared with ℓ_1 regularizer, slightly worse results can be observed for ℓ_2 regularizer. For ResNet-110 and ResNet-164, ℓ_2 regularizer leads to results comparable with ℓ_1 regularizer but at a larger parameter budget. When the regularizer is changed to ℓ_2, the proximal operator becomes

$$\mathbf{z}[k+1] = \left(1 - \frac{\lambda\mu}{\max\{\|\mathbf{z}[k+\Delta]\|, \lambda\mu\}}\right)\mathbf{z}[k+\Delta]. \tag{3.14}$$

By the formulation and experiments, ℓ_1 norm leads to faster convergence. To have a reasonable convergence speed, the λ used for ℓ_2 regularizer is 20 times larger than that for ℓ_1 regularizer.

Layer	Regularizer	Top1 Error (%)	FLOPs (%)	Params (%)
20	ℓ_1	8.46	51.8	56.13
	ℓ_2	8.66	51.59	54.19
110	ℓ_1	5.73	51.62	54.13
	ℓ_2	5.77	51.37	72.37
164	ℓ_1	5.22	51.67	50.97
	ℓ_2	5.18	50.87	60.66

Table 3.2: **Comparison between ℓ_1 norm and ℓ_2 norm regularization.** The experiments are done on ResNet.

3.5.1 *Image classification*

For image classification, experiments are done on two commonly used datasets including CIFAR10 [70] and Tiny-Imagenet [28]. CIFAR10 [70] contains 10 different classes. The training and testing subsets contain 50,000 and 10,000 images with resolution 32×32, respectively. As done by prior works [53, 62], we normalize all images using channel-wise mean and standard deviation of the the training set. Standard data augmentation is also applied. The networks are trained for 300 epochs with SGD optimizer and an initial learning rate of 0.1. The learning rate

Network Top-1 Err. (%)	Compression Method	Top-1 Err. (%)	FLOPs Ratio (%)	Param. Ratio (%)
ResNet-20 7.46	[159]	9.10	52.60	62.78
	DHP-50 (Ours)	8.46	51.80	56.13
	FPGM [54]	9.38	46.00	–
ResNet-56 7.05	Variational [175]	7.74	79.70	79.51
	Pruned-B [81]	6.94	72.40	86.30
	GAL-0.6 [97]	6.62	63.40	88.20
	NISP [163]	6.99	56.39	57.40
	DHP-50 (Ours)	6.42	50.96	58.42
	CaP [110]	6.78	50.20	–
	ENC [67]	7.00	50.00	–
	AMC [55]	8.10	50.00	–
	KSE [95]	6.77	48.00	45.27
	FPGM [54]	6.74	47.70	–
	GAL-0.8 [97]	8.42	39.80	34.10
	DHP-38 (Ours)	7.06	39.07	41.10
ResNet-110 5.31	**DHP-62 (Ours)**	5.37	63.66	63.2
	Variational [175]	7.04	63.56	58.73
	Pruned-B [81]	6.70	61.40	67.60
	GAL-0.5 [97]	7.26	51.50	55.20
	DHP-20 (Ours)	6.61	21.63	22.40
ResNet-164 4.97	Hinge [89]	5.40	53.61	70.34
	SSS [64]	5.78	53.53	84.75
	DHP-50 (Ours)	5.22	51.67	50.97
	Variational [175]	6.84	50.92	43.30
	DHP-20 (Ours)	6.30	21.78	20.46
DenseNet-12-40 5.26	Variational [175]	6.84	55.22	40.33
	DHP-38 (Ours)	6.06	39.80	63.76
	DHP-28 (Ours)	6.51	29.52	26.01
	GAL-0.1 [97]	6.77	28.60	25.00

Table 3.3: **Results for CIFAR10 classification.** The FLOPs ratio and parameter ratio of the pruned networks are reported. DHP outperforms the compared methods under comparable model complexity.

Network Top-1 Err. (%)	Compression Method	Top-1 Err. (%)	FLOPs Ratio (%)	Param. Ratio (%)
MobileNetV1 52.71	**DHP-24-2 (Ours)**	50.75	101.08	43.58
	MobileNetV1-0.75	54.22	57.42	57.64
	MetaPruning [101]	54.48	56.77	88.14
	DHP-50 (Ours)	51.63	51.91	36.95
MobileNetV2 44.75	**DHP-24-2 (Ours)**	43.82	99.09	72.72
	DHP-10 (Ours)	52.43	11.92	6.50
	MetaPruning [101]	56.72	11.00	90.27
	MobileNetV2-0.3	53.99	10.09	11.64

Table 3.4: **Results for Tiny-ImageNet classification.** DHP-24-2 shoots lower error rates than the original model.

is decayed by 10 after 50% and 75% of the epochs. The momentum of SGD is 0.9. Weight decay factor is set to 0.0001. The batch size is 64.

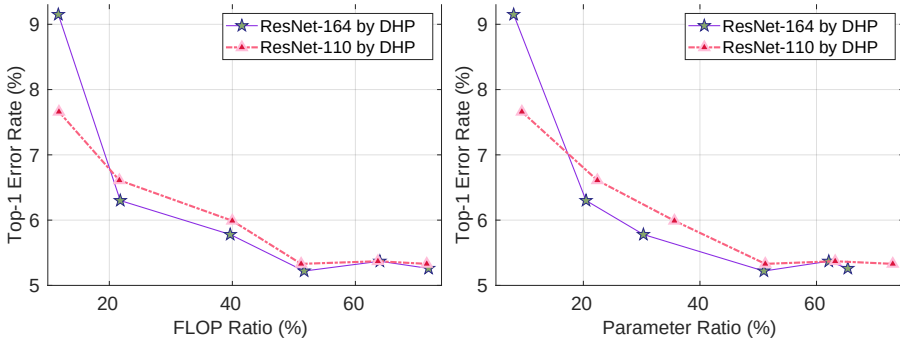
Tiny-Imagenet has 200 classes. Each class has 500 training images and 50 validation images. The resolution of the images is 64×64. The images are normalized with channel-wise mean and standard deviation. Horizontal flip is used to augment the dataset. The networks are trained for 220 epochs with SGD. The initial learning rate is 0.1. The learning rate is decayed by a factor of 10 at Epoch 200, Epoch 205, Epoch 210, and Epoch 215. The momentum of SGD is 0.9. Weight decay factor is set to 0.0001. The batch size is 64.

The compression results on image classification networks are shown in Table 3.3. 'DHP-**' denotes the proposed method with the target FLOPs ratio during pruning stage. As in Fig. 3.1, the operating point DHP-24-2 is derived by compressing the widened mobile networks with $\alpha = 2$ and the target FLOPs ratio 24%. For ResNet-56, the proposed method is compared with 9 different network compression methods and achieves the best performance, *i.e.* 6.42% Top-1 error rate on the most intensively investigated 50% compression level. The compression of DenseNet-12-40 is reasonable compared with the other methods. The accuracy of the operating points DHP-62 of ResNet-110 and DHP-50 of ResNet-164 is quite close to that of the baseline. More results on ResNet-110 and ResNet-164 are shown in the Supplementary. A comparison between ℓ_1 and ℓ_2 regularization in Eqn. (3.6) is done. The

results in the Supplementary shows that ℓ_1 regularization is better than ℓ_2 regularization.

On Tiny-ImageNet, DHP achieves lower Top-1 error rates than MetaPruning [101] under the same FLOPs constraint. DHP results in models that are more accurate than the uniformly scaled networks. The error rate of DHP-10 is 1.56% lower than that of MobileNetV2-0.3 with slightly fewer FLOPs and 5.14% fewer parameters. On MobileNetV1, the accuracy gain of DHP-50 over MobileNetV1-0.75 goes to 2.59% with over 5% fewer FLOPs and 10% fewer parameters. Based on this, we hypothesized that it is possible to derive a model which is more accurate than the original version by pruning the widened mobile networks. And this is confirmed by comparing the accuracy of the operating points DHP-24-2 with the baseline accuracy.

More results on ResNet-110 and ResNet-164 are shown in Fig. 3.4. When the compression ratio is not too severe (above 50%), the accuracy does not drop too much. The extreme compression prunes about 90% FLOPs and parameters of the original network. For ResNet-164, the extreme compression only keeps 8.04% parameters. Thus, the drop in the accuracy is reasonable.



(a) FLOPs compression ratio. (b) Parameter compression ratio.

Figure 3.4: **Top-1 error vs. FLOP and parameter compression ratio on ResNet-164 and ResNet-110.**

3.5.2 *Super-resolution*

For image super-resolution, the networks are trained on DIV2K [2] dataset. It contains 800 training images, 100 validation images, and

Method	PSNR [dB]					FLOPs [G]	Params [M]	Runtime [ms]	Mem [GB]
	Set5	Set14	B100	Urban100	DIV2K				
SRResNet [76]									
Baseline	32.03	28.50	27.52	25.88	28.85	32.83	1.54	34.73	0.6773
Clustering [137]	31.93	28.44	27.47	25.71	28.75	32.83	0.34	31.07	0.8123
Factor-SIC3 [147]	31.86	28.38	27.40	25.58	28.65	20.83	0.81	102.51	1.4957
DHP-60 (Ours)	31.97	28.47	27.48	25.76	28.79	20.29	0.95	27.91	0.5923
Basis-32-32 [90]	31.90	28.42	27.44	25.65	28.69	19.77	0.74	45.73	0.9331
Factor-SIC2 [147]	31.68	28.32	27.37	25.47	28.58	18.38	0.66	74.66	1.1201
Basis-64-14 [90]	31.84	28.38	27.39	25.54	28.63	17.49	0.60	36.75	0.6741
DHP-40 (Ours)	31.90	28.45	27.47	25.72	28.75	13.71	0.64	22.71	0.4907
DHP-20 (Ours)	31.77	28.34	27.40	25.55	28.60	7.77	0.36	14.74	0.3795
EDSR [96]									
Baseline	32.10	28.55	27.55	26.02	28.93	90.37	3.70	49.73	1.3276
Clustering [137]	31.93	28.47	27.48	25.77	28.80	90.37	0.82	50.51	1.2838
Factor-SIC3 [147]	31.96	28.47	27.49	25.81	28.81	65.49	2.19	125.10	1.5007
Basis-128-40 [90]	32.03	28.45	27.50	25.81	28.82	62.65	2.00	48.19	1.3219
Factor-SIC2 [147]	31.82	28.40	27.43	25.63	28.70	60.90	1.90	94.94	1.3209
Basis-128-27 [90]	31.95	28.42	27.46	25.76	28.76	58.28	1.74	45.84	1.3209
DHP-60 (Ours)	31.99	28.52	27.53	25.92	28.88	55.67	2.28	45.11	0.6950
DHP-40 (Ours)	32.01	28.49	27.52	25.86	28.85	37.77	1.53	33.50	0.9650
DHP-20 (Ours)	31.94	28.42	27.47	25.69	28.77	19.40	0.79	22.63	1.1588

Table 3.5: **Results on image super-resolution networks.** The upscaling factor is $\times 4$. Runtime is averaged for Urban100. Maximum GPU memory consumption is reported for Urban100. FLOPs is reported for a 128×128 image patch. DHP achieves significant reduction of runtime.

100 test images. Image patches are extracted from the training images. For EDSR, the patch size of the low-resolution input patch is 48×48 while for SRResNet the patch size is 24×24. The batch size is 16. The networks are optimized with Adam optimizer. The default hyper-parameter is used for Adam optimizer. The weight decay factor is 0.0001. The networks are trained for 300 epochs. The learning rate starts from 0.0001 and decays by 10 after 200 epochs. The networks are tested on Set5 [9], Set14 [166], B100 [108], Urban100 [63], and DIV2K validation set.

In order to speed up the training of EDSR, a simplified version of EDSR is adopted. The original EDSR contains 32 residual blocks and each convolutional layer in the residual blocks has 256 channels. The simplified version has 8 residual blocks and each has two convolutional layers with 128 channels.

The results on image super-resolution networks are shown in Table 3.5. DHP is compared with factorized convolution (Factor) [147], learning filter basis method (Basis) [90], and K-means clustering method (Clustering) [137]. 'SIC*' denotes the number of SIC layers in Factor [147]. The practical FLOPs instead of the theoretical FLOPs is reported for Clustering [137]. To fairly compare the methods and measure the practical compression effectiveness, five metrics are involved including Peak Signal-to-Noise Ratio (PSNR), FLOPs, number of parameters, runtime and GPU memory consumption. Several conclusion can be drawn. **I.** Previous methods mainly focus on the reduction of FLOPs and number of parameter without paying special attention to the actual acceleration. Although Clustering can reduce substantial parameters while maintaining quite good PSNR accuracy, the actual computing resource requirement (GPU memory and runtime) is remained. **II.** Convolution factorization and decomposition methods result in additional CUDA kernel calls, which is not efficient for the actual acceleration. **III.** For the proposed method, the two model complexity metrics, *i.e.* FLOPs and parameters change consistently across different operating points, which leads to consistent reduction of computation resources. **IV.** DHP results in both inference-efficient (DHP-20) and accuracy-preserving (DHP-60) models. The visual results are shown in Fig. 3.5. As can been seen, the visual quality of the images of DHP is almost indistinguishable from that of the baseline.

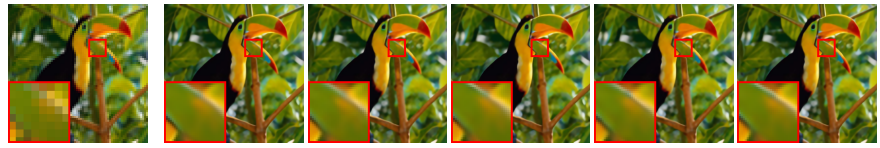


Figure 3.5: **Single image super-resolution visual results.** PSNR and FLOPs measured on the image. Runtime averaged on Set5.

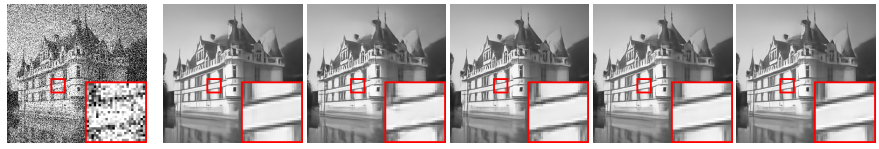
3.5.3 *Denoising*

For image denoising, the networks were trained on the gray version of DIV2K dataset and tested on BSD68 and DIV2K validation set. As done for image super-resolution, image patches are extracted from the training images. For DnCNN, the patch size of the input image is 64×64 and the batch size is 64. For UNet, the patch size is 128×128 and the batch size 16. Gaussian noise is added to degrade the input patches on the fly with noise level $\sigma = 70$. Adam optimizer is used to train the network. The weight decay factor is 0.0001. The networks are trained for 60 epochs and each epoch contains 10,000 iterations. So in total, the training continues for 600k iterations. The learning rate starts with 0.0001 and decays by 10 at Epoch 40.

The results for image denoising networks are shown in Table 3.6. The same metrics as super-resolution are reported for denoising. An additional method, *i.e.* filter group approximation (Group) [117] is included. In addition to the same conclusion in Subsec. 3.5.2, another two conclusions are drawn here. **I.** Group [117] fails to reduce the actual computation resources although with quite good accuracy and satisfactory reduction of FLOPs and number of parameters. This might due to the additional 1×1 convolution and possibly the inefficient implementation of group convolution in current deep learning toolboxes. **II.** For DnCNN, one interesting phenomenon is that Factor [147] is more accurate than the baseline but has larger appetite for other resources. This is due to two facts. Firstly, Factor [147] has skip connections within the SIC layer. The higher accuracy of Factor [147] just validates the effectiveness of skip connections. Secondly, the SIC layer of Factor [147] introduces more convolutional layers. So Factor-SIC3 has five times more convolutioinal layers than the baseline, which definitely slows down the execution. The visual results are shown in Fig. 3.6.

Method	PSNR [dB]		FLOPs [G]	Params [M]	Runtime [ms]	Mem [GB]
	BSD68	DIV2K				
DnCNN [169]						
Baseline	24.93	26.73	9.13	0.56	23.38	0.1534
Clustering [137]	24.90	26.67	9.13	0.12	21.97	0.2973
DHP-60 (Ours)	24.91	26.69	5.65	0.34	18.90	0.1443
DHP-40 (Ours)	24.89	26.65	3.83	0.23	14.62	0.1194
Factor-SIC3 [147]	24.97	26.83	3.54	0.22	125.46	0.5910
Group [117]	24.88	26.64	3.34	0.20	25.69	0.1807
Factor-SIC2 [147]	24.93	26.76	2.38	0.15	84.17	0.4149
DHP-20 (Ours)	24.84	26.58	2.01	0.12	10.72	0.0869
UNet [131]						
Baseline	25.17	27.17	3.41	7.76	8.73	0.1684
Clustering [137]	25.01	26.90	3.41	1.72	10.01	0.6704
DHP-60 (Ours)	25.14	27.11	2.11	4.76	6.86	0.4992
Factor-SIC3 [147]	25.04	26.94	1.56	3.42	39.84	0.1889
Group [117]	25.13	27.08	1.49	2.06	11.20	0.1481
DHP-40 (Ours)	25.12	27.08	1.43	3.24	4.50	0.4992
Factor-SIC2 [147]	25.01	26.90	1.22	2.51	30.16	0.1855
DHP-20 (Ours)	25.04	26.97	0.75	1.61	3.93	0.4992

Table 3.6: **Results on image denoising networks.** The noise level is 70. Runtime and maximum GPU memory are reported for BSD68. FLOPs is reported for a 128×128 image. DHP achieves significant reduction of runtime.



PSNR/FLOPs/Runtime (a) Noisy | 25.60/1.08/7.27 (b) UNet | 25.30/1.08/9.66 (f) Cluster | 25.37/0.49/40.36 (d) Factor | 25.51/0.47/9.00 (e) Group | 25.57/0.45/6.09 (c) DHP

Figure 3.6: **Image denoising visual results.** PSNR and FLOPs measured on the image. Runtime averaged on B100.

3.6 CONCLUSION AND FUTURE WORK

In this chapter, we proposed a differentiable automatic meta pruning method via hypernetwork for network compression. The differentiabil-

ity comes with the specially designed hypernetwork and the proximal gradient used to search the potential candidate network configurations. The automation of pruning lies in the uniformly applied ℓ_1 sparsity on the latent vectors and the proximal gradient that solves the problem. By pruning mobile network with width multiplier $\alpha = 2$, we obtained models with higher accuracy but lower computation complexity than that with $\alpha = 1$. We hypothesize this is due to the per-layer distinguishing configuration resulting from the automatic pruning. Future work might be investigating whether this phenomenon reoccurs for the other networks.

4 GROUP SPARSITY

4.1 INTRODUCTION

In the previous two chapters, we investigated filter decomposition and filter pruning for network compression individually. For filter decomposition, we propose a method that unifies the previous methods in Chapter 2. For filter pruning, a differentiable pruning method is proposed in Chapter 3. In this chapter, we propose a unified analysis of the two techniques.

Despite their success, both the pruning-based and decomposition-based approaches have their respective limitations. Filter pruning can only take effect in pruning output channels of a tensor and equivalently cancelling out inactive filters. This is not feasible under some circumstances. The skip connection in a block is such a case where the output feature map of the block is added to the input. Thus, pruning the output could amount to cancelling a possible important input feature map. This is the reason why many pruning methods fail to deal with the second convolution of the ResNet [53] basic block. As for filter decomposition, it always introduces another 1×1 convolutional layer, which means additional overhead of calling CUDA kernels.

Previously, filter pruning and decomposition were developed separately. In this chapter, we unveil the fact that filter pruning and decomposition are highly related from the viewpoint of compact tensor approximation. Specifically, both filter pruning and filter decomposition seek a compact approximation of the parameter tensors despite their different operation forms to cope with the application scenarios. Consider a vectorized image patch $\mathbf{x} \in \mathbb{R}^{m \times 1}$ and a group of n filters $\mathbf{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_n\} \in \mathbb{R}^{m \times n}$. The pruning methods remove output channels and approximate the original output $\mathbf{x}^T\mathbf{W}$ as $\mathbf{x}^T\mathbf{C}$, where $\mathbf{C} \in \mathbb{R}^{m \times k}$ only has k output channels. Filter decomposition methods approximate $\mathbf{W}$ as two filters $\mathbf{A} \in \mathbb{R}^{m \times k}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$ and $\mathbf{AB}$ is the rank k approximation of $\mathbf{W}$. Thus, both the pruning and decomposition based methods seek a compact approximation to the original network parameters, but adopt different strategies for the approximation.

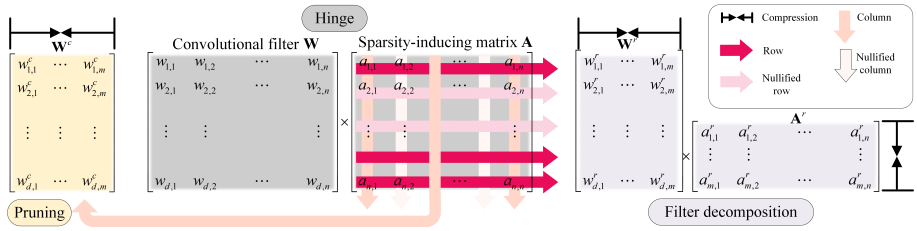


Figure 4.1: **The hinge between filter decomposition and filter pruning.** A sparsity-inducing matrix $\mathbf{A}$ is attached to a normal convolution. The matrix acts as the hinge between filter pruning and decomposition. By enforcing group sparsity to the columns and rows of the matrix, equivalent pruning and decomposition operations can be obtained. For pruning, the product of $\mathbf{W}$ and the column-reduced matrix $\mathbf{A}^c$, *i.e.* $\mathbf{W}^c$ acts as the new convolutional filter. To save computation after filter decomposition the reduced matrices $\mathbf{W}^r$ and $\mathbf{A}^r$ are used as two convolutional filters.

The above observation shows that filter pruning and decomposition constitute complementary components of each other. This fact encourages us to design a unified framework that is able to incorporate the pruning-based and decomposition-based approaches simultaneously. This simple yet effective measure can endow the devised algorithm with the ability of flexibly switching between the two operation modes, *i.e.* filter pruning and decomposition, depending on the layer-wise configurations. This makes it possible to leverage the benefits of both methods.

The hinge point between pruning and decomposition is group sparsity, see Fig. 4.1. Consider a 4D convolutional filter, reshaped into a 2D matrix $\mathbf{W} \in \mathbb{R}^{\text{features} \times \text{outputs}}$. Group sparsity is added by introducing a sparsity-inducing matrix $\mathbf{A}$. By applying group sparsity constraints on the columns of $\mathbf{A}$, the output channel of the sparsity-inducing matrix $\mathbf{A}$ and equivalently of the matrix product $\mathbf{W} \times \mathbf{A}$ can be reduced by solving an optimization problem. This is equivalent to filter pruning. On the other hand, if the group sparsity constraints are applied on the rows of $\mathbf{A}$, then the inner channels of the matrix product $\mathbf{W} \times \mathbf{A}$, namely, the output channel of $\mathbf{W}$ and the input channel of $\mathbf{A}$, can be reduced. To save the computation, the single heavyweight convolution

$\mathbf{W}$ is converted to a lightweight and a 1×1 convolution with respect to the already reduced matrices $\mathbf{W}^r$ and $\mathbf{A}^r$. This breaks down to filter decomposition.

Thus, the contribution of this chapter is four-fold.

I Starting from the perspective of ***compact tensor approximation***, the connection between filter pruning and decomposition is analyzed. Although this perspective is the core of filter decomposition, it is still novel for network pruning. Actually, both of the methods approximate the weight tensor with compact representation that keeps the accuracy of the network.

II Based on the analysis, we propose to use ***sparsity-inducing matrices*** to hinge filter pruning and decomposition and introduce a unified formulation. This square matrix is inspired by filter decomposition and corresponds to a 1×1 convolution. By changing the way how the sparsity regularizer is applied to the matrix, our algorithm can achieve equivalent effect of either filter pruning or decomposition or both. To the best of our knowledge, this is the first work that analyzes the two methods under the same umbrella.

III The third contribution is the development of ***binary search, gradient based learning rate adjustment, layer balancing, and annealing methods***. All are important for the success of the proposed algorithm. These details are obtained by observing the influence of the proximal gradient method on the filter during the optimization.

IV The proposed method can be ***applied to various CNNs***. We apply this method to VGG [136], ResNet [53], ResNeXt [152], WRN [165], and DenseNet [62]. The proposed network compression method achieves state-of-the-art performance on these networks.

4.2 RELATED WORK

In this section, we firstly review the closely related work including pruning-based and decomposition-based compression methods.

4.2.1 *Network Pruning with Group Sparsity*

Structural pruning aims at zeroing out structured groups of the convolutional filters [56, 54]. Specifically, group sparsity regularization has been investigated in recent works for the structural pruning of network parameters [178, 150, 4]. Wen *et al.* [150] and Alvarez *et al.* [4] proposed to impose group sparsity regularization on network parameters to reduce the number of feature map channels in each layer. The success of this method triggered the studies of group sparsity based network pruning. Subsequent works improved group sparsity based approaches in different ways. One branch of works combined the group sparsity regularizer with other regularizers for network pruning. A low-rank regularizer [3] as well as an exclusive sparsity regularizer [161] were adopted for improving the pruning performance. Another branch of research investigated a better group-sparsity regularizer for parameter pruning including group ordered weighted ℓ_1 regularizer [167], out-in-channel sparsity regularization [82] and guided attention for sparsity learning [143]. In addition, some works also attempted to achieve group-sparse parameters in an indirect manner. In [102] and [64], scaling factors were introduced to scale the outputs of specific structures or feature map channels to structurally prune network parameters.

4.2.2 *Filter Decomposition and Group Sparsity*

Another category of works compresses network parameters through tensor decomposition [29, 74, 65, 173, 90]. A brief review of the filter decomposition methods is given in Subsec. 2.2.3. Here we point out that filter decomposition methods rely on the low-rankness of the convolutional filters. Specifically, in the simplest form, the original filter can be decomposed into basis filters and a linear combination matrix by singular value decomposition. Then based on the low-rankness assumption, filters with smaller singular values are removed. When the singular values are multiplied with the linear combination matrix, this can be regarded as applying group sparsity on the matrix.

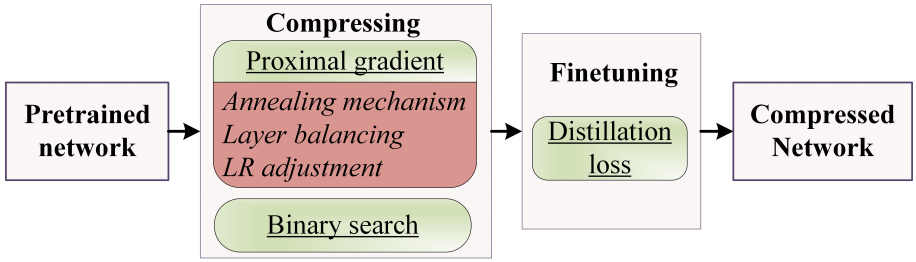


Figure 4.2: **The flowchart of the proposed algorithm.**

4.3 THE PROPOSED METHOD

This section explains the proposed method (Fig. 4.2). Specifically, it describes how group sparsity can hinge filter pruning and decomposition. The pair $\{\mathbf{x}, \mathbf{y}\}$ denotes the input and target of the network. Without loss of clarity, we also use $\mathbf{x}$ to denote the input feature map of a layer. The output feature map of a layer is denoted by $\mathbf{z}$. The filters of a convolutional layer are denoted by $\mathbf{W}$ while the introduced group sparsity matrix is denoted by $\mathbf{A}$. The rows and columns of $\mathbf{A}$ are denoted by $\mathbf{A}_i$, and $\mathbf{A}_j$, respectively. The general structured groups of $\mathbf{A}$ are denoted by $\mathbf{A}_g$.

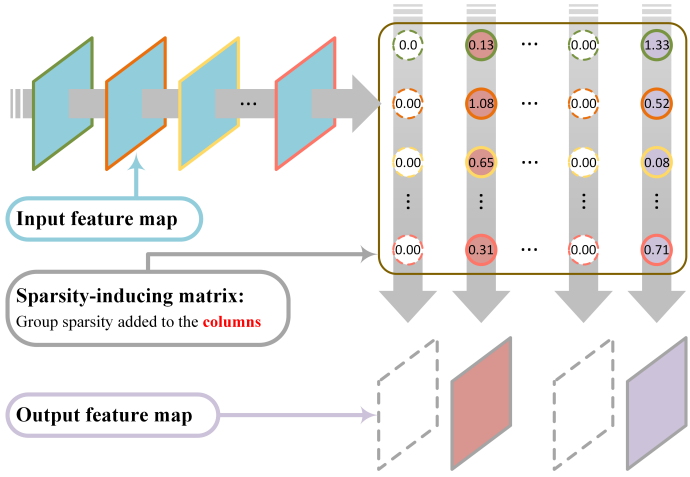
4.3.1 *Group sparsity*

The convolution between the input feature map $\mathbf{x}$ and the filters can be converted to a matrix multiplication, *i.e.* ,

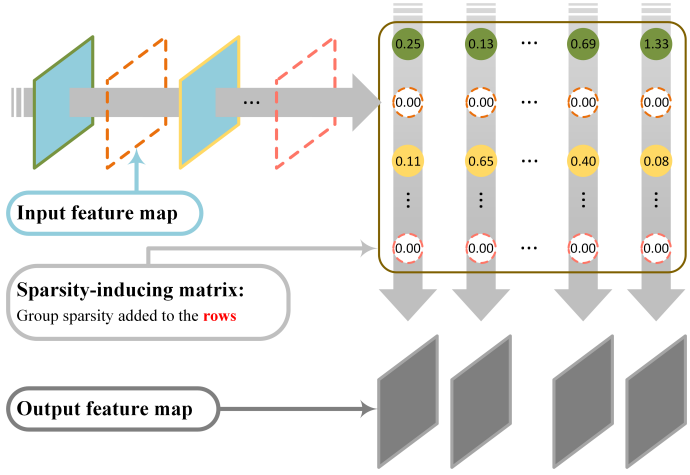
$$\mathbf{Z} = \mathbf{X} \times \mathbf{W}, \tag{4.1}$$

where $\mathbf{X} \in \mathbb{R}^{N \times cwh}$, $\mathbf{W} \in \mathbb{R}^{cwh \times n}$, and $\mathbf{Z} \in \mathbb{R}^{N \times n}$ are the reshaped input feature map, output feature map, and convolutional filter, c, n, $w \times h$, and N denotes the input channel, number of filters, filter size, and number of reshaped features, respectively. For the sake of brevity, the bias term is omitted here. The weight parameters $\mathbf{W}$ are usually trained with some regularization such as weight decay to avoid overfitting the network. To get structured pruning of the filter, structured sparsity regularization is used to constrain the filter, *i.e.*

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{y}, f(\mathbf{x}; \mathbf{W})) + \mu \mathcal{D}(\mathbf{W}) + \lambda \mathcal{R}(\mathbf{W}), \tag{4.2}$$



(a) Group sparsity enforced on column.



(b) Group sparsity enforced on row.

Figure 4.3: **Group-sparsity regularization enforcement.** (a) The columns of the sparsity-inducing matrix are regularized. This results in nullified filters and the corresponding output feature maps are removed. (b) The rows are regularized and some are zeroed out. The filters of the previous layer and also the feature maps are removed.

where $\mathcal{D}(\cdot)$ and $\mathcal{R}(\cdot)$ are the weight decay and sparsity regularization, μ and λ are the regularization factors.

Different from other group sparsity methods that directly regularize the matrix $\mathbf{W}$ [161, 82], we enforce group sparsity constraints by

incorporating a sparsity-inducing matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, which can be converted to the filter of a 1×1 convolutional layer after the original layer. Then the original convolution in Eqn. (4.1) becomes $\mathbf{Z} = \mathbf{X} \times (\mathbf{W} \times \mathbf{A})$. To obtain a structured sparse matrix, group sparsity regularization is enforced on $\mathbf{A}$. Thus, the loss function becomes

$$\min_{\mathbf{W},\mathbf{A}} \mathcal{L}(\mathbf{y}, f(\mathbf{x}; \mathbf{W}, \mathbf{A})) + \mu \mathcal{D}(\mathbf{W}) + \lambda \mathcal{R}(\mathbf{A}). \tag{4.3}$$

Solving the problem in Eqn. (4.3) results in structured group sparsity in matrix $\mathbf{A}$. By considering matrix $\mathbf{W}$ and $\mathbf{A}$ together, the actual effect is that the original convolutional filter is compressed.

In comparison with the filter selection method [102, 64], the proposed method not only selects the filters in a layer, but also makes linear combinations of the filters to minimize the error between the original and the compact filter. On the other hand, different from other group sparsity constraints [161, 82], there is no need to change the original filters $\mathbf{W}$ of the network too much during optimization of the sparsity problem. In our experiments, we set a much smaller learning rate for the pretrained weight matrix $\mathbf{W}$.

4.3.2 *The hinge*

The group sparsity term in Eqn. (4.3) controls how the network is compressed. This term has the form

$$\mathcal{R}(\mathbf{A}) = \Phi(\|\mathbf{A}_g\|_2), \tag{4.4}$$

where $\mathbf{A}_g$ denotes the different groups of $\mathbf{A}$, $\|\mathbf{A}_g\|_2$ is the ℓ_2 norm of the group, and $\Phi(\cdot)$ is a function of the group ℓ_2 norms.

If group sparsity regularization is added to the columns of $\mathbf{A}$ as in Fig. 4.3a, *i.e.* , $\mathcal{R}(\mathbf{A}) = \Phi(\|\mathbf{A}_j\|_2)$, a column pruned version $\mathbf{A}^c$ is obtained and the output channels of the corresponding 1×1 convolution are pruned. In this case, we can multiply $\mathbf{W}$ and $\mathbf{A}^c$ and use the result as the filter of the convolutional layer. This is equivalent to pruning the output channels of the convolutional layer with the filter $\mathbf{W}$.

On the other hand, group sparsity can be also applied to the rows of $\mathbf{A}$, *i.e.* $\mathcal{R}(\mathbf{A}) = \Phi(\|\mathbf{A}_i\|_2)$. In this case, a row-sparse matrix $\mathbf{A}^r$ is derived and the input channels of the 1×1 convolution can be pruned (See Fig. 4.3b). Accordingly, the corresponding output channels of the former convolution with filter $\mathbf{W}$ can be also pruned. However,

Data: training dataset
Result: the compressed network
initialization: the current compression ratio $\gamma_c = 1$; the target compression ratio γ^*, the nullifying threshold of the group ℓ_2 norm $\mathcal{T}$;
while $\gamma_c - \gamma^* <= \alpha$ **do**
    start a a new epoch;
    for *batch* $\in$ *training dataset* **do**
        $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_s \nabla \mathcal{G}(\mathbf{W}_t)$;
        $\mathbf{A}_{t+\Delta} = \mathbf{A}_t - \eta \nabla \mathcal{H}(\mathbf{A}_t)$;
        $\mathbf{A}_{t+1} = \mathbf{prox}_{\lambda\eta\mathcal{R}}(\mathbf{A}_{t+\Delta})$;
    end
    compress the network with the threshold $\mathcal{T}$;
    compute the compression ratio γ_c
end

Algorithm 1: The optimization algorithm used to solve the problem defined in Eqn. (4.3).

since the number of output channel of the later convolution is not changed, multiplying out the two compression matrices does not save any computation. So a better choice is to leave them as two separate convolutional layers. This tensor manipulation method is equivalent to filter decomposition where a single convolution is decomposed into a lightweight one and a linear combination. In conclusion, by enforcing group sparsity to the columns and rows of the introduced matrix $\mathbf{A}$, we can derive two tensor manipulation methods that are equivalent to the operation of filter pruning and decomposition, respectively. This provides a degree of freedom to choose the tensor manipulation method depending on the specifics of the underlying network.

4.3.3 *Proximal gradient solver*

To solve the problem defined by Eqn. (4.3), the parameter $\mathbf{W}$ can be updated with SGD but with a small learning rate, *i.e.* $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_s \nabla \mathcal{G}(\mathbf{W}_t)$, where $\mathcal{G}(\mathbf{W}_t) = \mathcal{L}(\cdot, f(\cdot; \mathbf{W}_t, \cdot)) + \mu \mathcal{D}(\mathbf{W}_t)$. This is because it is undesirable to modify the pretrained parameters too much during the optimization phase. The focus should be on the sparsity matrix $\mathbf{A}$.

Result: the nullifying threshold $\mathcal{T}^* = g^{-1}(\gamma^*)$
initialization: the target compression ratio γ^*, the initial step s, the stop criterion $\mathcal{C}$, and $\mathcal{T} = \mathcal{T}_0$;
while $|\gamma_n - \gamma^*| > \mathcal{C}$ **do**
    compress the network with the threshold $\mathcal{T}$;
    calculate the current compression ratio γ_n;
    if $(\gamma_{n-1} >= \gamma^*) == (\gamma_n < \gamma^*)$ **then** $s \leftarrow s/2$;
    if $\gamma_n > \gamma_t$ **then**
        $\mathcal{T} \leftarrow \mathcal{T} + s$;
    else
        $\mathcal{T} \leftarrow \mathcal{T} - s$;
    end
end

Algorithm 2: Binary search of the threshold $\mathcal{T}$.

The proximal gradient algorithm [114] is used to optimize the matrix $\mathbf{A}$ in Eqn. (4.3). It consists of two steps, *i.e.* gradient descent and proximal operator. The parameters in $\mathbf{A}$ are first updated by SGD with the gradient of the loss function $\mathcal{H}(\mathbf{A}) = \mathcal{L}(\mathbf{y}, f(\mathbf{x}; \mathbf{W}, \mathbf{A}))$, namely,

$$\mathbf{A}_{t+\Delta} = \mathbf{A}_t - \eta \nabla \mathcal{H}(\mathbf{A}_t), \tag{4.5}$$

where η is the learning rate and $\eta >> \eta_s$. Then the proximal operator chooses a neighborhood point of $A_{t+\Delta}$ that minimizes the group sparsity regularization, *i.e.*

$$\begin{aligned} \mathbf{A}_{t+1} &= \mathbf{prox}_{\lambda\eta\mathcal{R}}(\mathbf{A}_{t+\Delta}) \\ &= \underset{\mathbf{A}}{\operatorname{argmin}} \left\{ \mathcal{R}(\mathbf{A}_{t+\Delta}) + \frac{1}{2\lambda\eta} \|\mathbf{A} - \mathbf{A}_{t+\Delta}\|_2^2 \right\}. \end{aligned} \tag{4.6}$$

The sparsity regularizer $\Phi(\cdot)$ can have different forms, *e.g.* , ℓ_1 norm [114], $\ell_{1/2}$ norm [155], ℓ_{1-2} norm [158], or logsum [42]. All of them try to approximate the ℓ_0 norm. In this chapter, we mainly use $\{\ell_p : p = 1, 1/2\}$ regularizers while we also include the ℓ_{1-2} and logsum regularizers in the ablation studies. The proximal operators of the four regularizers have a closed-form solution. Briefly, the solution is the soft-thresholding operator [11] for $p = 1$ and the half-thresholding operator for $p = 1/2$ [155]. The solutions are given in the next section. The gradient step and proximal step are interleaved in the optimization

phase of the regularized loss until some predefined stopping criterion is achieved. After each epoch, groups with ℓ_2 norms smaller than a predefined threshold $\mathcal{T}$ are nullified. And the compression ratio in terms of FLOPs is calculated. When the difference between the current and the target compression ratio γ_c and γ^* is lower than the stopping criterion α, the compression phase stops. The detailed compression algorithm that utilizes the proximal gradient is shown in Algorithm 1.

4.3.4 *Binary search of the nullifying threshold*

After the compression phase stops, the resulting compression ratio is not exactly the same as the target compression ratio. To fit the target compression ratio, we use a binary search algorithm to determine the nullifying threshold $\mathcal{T}$. The compression ratio γ is actually a monotonous function of the threshold $\mathcal{T}$, *i.e.* $\gamma = g(\mathcal{T})$. However, the explicit expression of the function $g(\cdot)$ is not known. Given a target compression threshold γ^*, we want to derive the threshold needed to nullify the sparse groups, *i.e.* $\mathcal{T}^* = g^{-1}(\gamma^*)$, where $g^{-1}(\cdot)$ is the inverse function of $g(\cdot)$. The binary search approach shown in Algorithm 2 starts with an initial threshold $\mathcal{T}_0$ and a step s. It adjusts the threshold $\mathcal{T}$ according to the values of the current and target compression ratio. The step s is halved if the target compression ratio sits between the previous one γ_{n-1} and the current one γ_n. The searching procedure stops as soon as the final compression ratio γ_n is close enough to the target, *i.e.* , $|\gamma_n - \gamma^*| \leq \mathcal{C}$.

4.3.5 *Gradient based adjustment of learning rate*

In the ResNet basic block, both of the two 3×3 convolutional layers are attached a sparsity-inducing matrix $\mathbf{A}^1$ and $\mathbf{A}^2$, namely, 1×1 convolutional layers. We empirically find that the gradient of the first sparsity-inducing matrix is larger than that of the second. Thus, it is easier for the first matrix to jump to a point with larger average group ℓ_2 norms. This results in unbalanced compression of the two sparsity-inducing matrices since the same nullifying threshold is used for all of the layers. Thus, much more channels of $\mathbf{A}^2$ are compressed than channels of $\mathbf{A}^1$. This is an undesirable compression approach

since both of the two layers are equally important. Hence, a balanced compression between them is preferable.

To solve this problem, we adjust the learning rate of the first and second sparsity-inducing matrices according to their gradients. Let the ratio of the average group ℓ_2 norm between the gradients of the matrices be

$$\rho = \sum_g \left(\nabla \mathbf{A}^1\right)_g / \sum_g \left(\nabla \mathbf{A}^2\right)_g . \tag{4.7}$$

Then the learning rate of the first convolution is divided by ρ^m. We empirically set $m = 1.35$.

4.3.6 *Group ℓ_2 norm based layer balancing*

The proximal gradient method depends highly on the group sparsity term. That is, if the initial ℓ_2 norm of a group is small, then it is highly likely that this group will be nullified. The problem is that the distribution of the group ℓ_2 norm across different layers can be very diverse, which can result in unbalanced compression of the layers. In this case, a narrow bottleneck could appear in the compressed network that would hamper the performance. To solve this problem, we use the mean of the group ℓ_2 norm of a layer to recalibrate the regularization factor of the layer. That is,

$$\lambda^l = \lambda \frac{1}{G} \sum_{g=1}^{G} \|\mathbf{A}_g\|_2, \tag{4.8}$$

where λ^l is the regularization factor of the l-th layer. In this way, the layers with larger average group ℓ_2 norm obtain a larger penalty.

4.3.7 *Regularization factor annealing*

The compression procedure starts with a fixed regularization factor. However, towards the end of the compression phase, the fixed regularization factor may be so large that more than the desired groups are nullified in one epoch. Thus, to solve the problem, we anneal the regularization factor when the average group ℓ_2 norm shrinks below some threshold. The annealing also impacts the proximal step but has less influence while the gradient step plays a more active role in finding the local minimum.

4.3.8 *Distillation loss in the finetuning phase*

In Eqn. (4.3), the prediction loss and the groups sparsity regularization are used to solve the compression problem. After the compression phase, the derived model is further finetuned. During this phase, a distillation loss is exploited to force similar logit outputs of the original network and the pruned one. The vanilla distillation loss is used, *i.e.*

$$\begin{aligned}\mathcal{L} = &(1-\alpha)\mathcal{L}_{ce}(\mathbf{y}, \sigma(\mathbf{z}_c)) \\ &+ 2\alpha T^2 \mathcal{L}_{ce}\left(\sigma\left(\frac{\mathbf{z}_c}{T}\right), \sigma\left(\frac{\mathbf{z}_o}{T}\right)\right),\end{aligned} \tag{4.9}$$

where $\mathcal{L}_{ce}(\cdot)$ denotes the cross-entropy loss, $\sigma(\cdot)$ is the softmax function, $\mathbf{z}_c$ and $\mathbf{z}_o$ are the logit outputs of the compressed and the original network. For the sake of simplicity, the network parameters are omitted. We use a fixed balancing factor $\alpha = 0.4$ and temperature $T = 4$.

4.4 CLOSED-FORM SOLUTIONS TO THE PROXIMAL OPERATORS

In this section, the closed-form solutions to the proximal operators in Eqn. (4.6) is presented. The proximal operator of a given function $f(\cdot)$ is defined by

$$\mathbf{prox}_{\lambda f} = \underset{v}{\operatorname{argmin}} \left\{ f(v) - \frac{1}{\lambda}\|x - v\|_2^2 \right\} \tag{4.10}$$

This operator has closed-form solution when the function $f(\cdot)$ has the form of ℓ_1, $\ell_{1/2}$, ℓ_{1-2}, and logsum regularization. For ℓ_1, the solution is the soft-thresholding function and for $\ell_{1/2}$ it is the so-called half-thresholding function. The soft-thresholding function is defined as

$$\mathcal{S}_\lambda(x) = \operatorname{sgn}(x)[|x| - \lambda]_+, \tag{4.11}$$

where $\operatorname{sgn}(\cdot)$ is the sign function and $[\cdot]_+$ calculates the maximum of the argument and 0. The hard-thresholding function is given by

$$\mathcal{H}_\lambda(x) = \begin{cases} \frac{2}{3}x(1 + \cos(\frac{2\pi}{3} - \frac{2}{3}\phi_\lambda(x))), & |x| > \frac{\sqrt[3]{54}}{4}(\lambda)^{\frac{2}{3}}, \\ 0, & \text{otherwise}, \end{cases} \tag{4.12}$$

where $\phi_\lambda(x) = \arccos(\frac{\lambda}{8}(\frac{|x|}{3})^{-\frac{3}{2}})$.

The $\ell_{2,1}$ group sparsity regularizer is defined as

$$\mathcal{R}(\mathbf{A}) = \Phi(\|\mathbf{A}_g\|_2) = \sum_g \|\mathbf{A}_g\|_2^p, \tag{4.13}$$

where $\Phi(\cdot)$ is the function of the group ℓ_2 norms $\|\mathbf{A}_g\|_2$ and has the form of ℓ_1 norm here. The proximal operator of the sparsity-inducing matrix $\mathbf{A}$ defined in the main chapter is

$$\mathbf{A}_{t+1} = \mathbf{prox}_{\lambda\eta\mathcal{R}}(\mathbf{A}_{t+\Delta}) = \underset{\mathbf{A}}{\operatorname{argmin}} \left\{ \mathcal{R}(\mathbf{A}_{t+\Delta}) + \frac{1}{2\lambda\eta}\|\mathbf{A} - \mathbf{A}_{t+\Delta}\|_F^2 \right\}, \tag{4.14}$$

where the function $\mathcal{R}(\cdot)$ replaces $f(\cdot)$ in Eqn. (4.10). The closed-form solution of the proximal operator in Eqn. (4.14) can be derived from the solutions to Eqn. (4.10) according to the following theorem [8].

Theorem 4.1. *Let $f : \mathbb{E} \to \mathbb{R}$ be a function given by $f(\mathbf{x}) = g(\|\mathbf{x}\|)$, where $g : \mathbb{R} \to (-\infty, \infty]$ is a proper closed and convex function satisfying* $\mathrm{dom}(g) \subseteq [0, \infty)$. *Then,*

$$\mathbf{prox}_{\lambda f}(\mathbf{x}) = \begin{cases} \mathbf{prox}_{\lambda g}(\|\mathbf{x}\|_2)\frac{\mathbf{x}}{\|\mathbf{x}\|_2}, & \mathbf{x} \neq \mathbf{0}, \\ \{\mathbf{u} \in \mathbb{E} : \|\mathbf{u}\|_2 = \mathbf{prox}_{\lambda g}(\mathbf{0})\}, & \mathbf{x} = \mathbf{0}. \end{cases} \tag{4.15}$$

Thus, with a little bit variable substitution, when $\Phi(\cdot)$ is ℓ_1 regularizer, the solution to Eqn. (4.14) is given by

$$\mathbf{A}_{t+1} = \left[1 - \frac{\lambda\eta}{\|\mathbf{A}_g\|_2}\right]_+ \mathbf{A}_{g,i}, \tag{4.16}$$

where $\mathbf{A}_{g,i}$ is the i-th element in the g-th group of the sparsity-inducing matrix $\mathbf{A}$, and for the sake of simplicity, the subscript $_{t+\Delta}$ is omitted.

When the function $\Phi(\cdot)$ has the form of $\ell_{1/2}$, ℓ_{1-2}, and logsum, it is non-convex. However, we still use the variable substitution in Theorem 4.1 experimentally and the corresponding results in the main chapter are also very competitive. For $\ell_{1/2}$ regularizer, the solution is given by

$$\mathbf{A}_{t+1} = \begin{cases} \frac{2}{3}\left(1 + \cos\left(\frac{2\pi}{3} - \frac{2}{3}\phi_{\lambda\eta}\left(\|\mathbf{A}_g\|_2\right)\right)\right)\mathbf{A}_{g,i}, & \|\mathbf{A}_g\|_2 > \frac{\sqrt[3]{54}}{4}(\lambda\eta)^{\frac{2}{3}}, \\ 0, & \text{otherwise}, \end{cases} \tag{4.17}$$

ℓ_1	$\mathbf{A}_{t+1} = \left[1 - \frac{\lambda\eta}{\|\mathbf{A}_g\|_2}\right]_+ \mathbf{A}_{g,i}$
$\ell_{1/2}$	$\mathbf{A}_{t+1} = \begin{cases} \frac{2}{3}\left(1 + \cos\left(\frac{2\pi}{3} - \frac{2}{3}\phi_{\lambda\eta}\left(\|\mathbf{A}_g\|_2\right)\right)\right)\mathbf{A}_{g,i}, & \|\mathbf{A}_g\|_2 > \beta, \\ 0, & \text{otherwise}, \end{cases}$ $\beta = \frac{\sqrt[3]{54}}{4}(\lambda\eta)^{\frac{2}{3}}$, $\phi_{\lambda\eta}(\|\mathbf{A}_g\|_2) = \arccos(\frac{\lambda\eta}{8}(\frac{\|\mathbf{A}_g\|_2}{3})^{-\frac{3}{2}})$
ℓ_{1-2}	$\mathbf{A}_{t+1} = (1 + \frac{\lambda\eta}{\|\mathbf{c}\|_2})[1 - \frac{\lambda\eta}{\|\mathbf{A}_g\|_2}]_+\mathbf{A}_{g,i}$ $\mathbf{c}_g = [\|\mathbf{A}_g\|_2 - \lambda\eta]_+$
logsum	$\mathbf{A}_{t+1} = \begin{cases} \frac{c_1+\sqrt{c_2}}{2}\frac{\mathbf{A}_{g,i}}{\|\mathbf{A}_g\|_2}, & c_2 > 0, \\ 0, & c_2 \leqslant 0, \end{cases}$ $\lambda > 0$, $0 < \epsilon < \sqrt{\lambda\eta}$, $c_1 = \|\mathbf{A}_g\|_2 - \epsilon$, $c_2 = c_1^2 - 4(\lambda\eta - \epsilon\|\mathbf{A}_g\|_2)$

Table 4.1: **The solution to the proximal operator for ℓ_1, ℓ_{1-2}, $\ell_{1/2}$, and** logsum **regularizers.**

where $\phi_{\lambda\eta}(\|\mathbf{A}_g\|_2) = \arccos(\frac{\lambda\eta}{8}(\frac{\|\mathbf{A}_g\|_2}{3})^{-\frac{3}{2}})$. Similarly, the solution to the logsum regularizer is given by

$$\mathbf{A}_{t+1} = \begin{cases} \frac{c_1+\sqrt{c_2}}{2}\frac{\mathbf{A}_{g,i}}{\|\mathbf{A}_g\|_2}, & c_2 > 0, \\ 0, & c_2 \leqslant 0, \end{cases} \tag{4.18}$$

where $\lambda > 0$, $0 < \epsilon < \sqrt{\lambda\eta}$, $c_1 = \|\mathbf{A}_g\|_2 - \epsilon$, and $c_2 = c_1^2 - 4(\lambda\eta - \epsilon\|\mathbf{A}_g\|_2)$. When the regularizer is ℓ_{1-2} regularizer, then the solution is given by

$$\mathbf{A}_{t+1} = (1 + \frac{\lambda\eta}{\|\mathbf{c}\|_2})[1 - \frac{\lambda\eta}{\|\mathbf{A}_g\|_2}]_+\mathbf{A}_{g,i} \tag{4.19}$$

where $\mathbf{c}_g = [\|\mathbf{A}_g\|_2 - \lambda\eta]_+$. Note that the case where all of the group ℓ_2 norms $\mathbf{A}_g$ equal 0 is not considered [158] because it never happens during the optimization of our algorithm. The solutions are summarized in Table 4.1.

4.5 IMPLEMENTATION CONSIDERATIONS

4.5.1 *Sparsity-inducing matrix in network blocks*

In the analysis of Sec. 4.3, a 1×1 convolutional layer with the sparsity-inducing matrix is appended after the uncompressed layer. When it

comes to different network blocks, we tweak it a little bit. As stated, both of the 3×3 convolutions in the ResNet [53] basic block are appended with a 1×1 convolution. For the first sparsity-inducing matrix, group sparsity regularization can be enforced on either the columns or the rows of the matrix. As for the second matrix, group sparsity is enforced on its rows due to the existence of the skip connection.

The ResNet [53] and ResNeXt [152] bottleneck block has the structure of $1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$ convolutions. Here, the natural choice of sparsity-inducing matrices are the leading and the ending convolutions. For the ResNet bottleneck block, the two matrices select the input and output channels of the middle 3×3 convolution, respectively. Things become a little bit different for the ResNeXt bottleneck since the middle 3×3 convolution is a group convolution. So the aim becomes enforcing sparsity on the already existing groups of the group convolution. In order to do that, the parameters related to the groups in the two sparsity-inducing matrices are concatenated. Then group sparsity is enforced on the new matrix. After the compression phase, a whole group can be nullified.

4.5.2 *Initialization of* $\mathbf{W}$ *and* $\mathbf{A}$

For the ResNet and ResNeXt bottleneck block, 1×1 convolutions are already there. So the original network parameters are used directly. However, it is necessary to initialize the newly added sparsity-inducing matrix $\mathbf{A}$. Two initialization methods are tried. The first one initializes $\mathbf{W}$ and $\mathbf{A}$ with the pretrained parameters and identity matrix, respectively. The second method first calculates the singular value decomposition of $\mathbf{W}$, *i.e.* $\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. Then the left eigenvector $\mathbf{U}$ and the matrix $\mathbf{S}\mathbf{V}^T$ are used to initialize $\mathbf{W}$ and $\mathbf{A}$. Note that the singular values are annexed by the right eigenvector. Thus, the columns of $\mathbf{W}$, *i.e.* the filters of the convolutional layer lie on the surface of the unit sphere in the high-dimensional space.

4.6 EXPERIMENTAL RESULTS

In this section, the proposed method is validated on three image classification datasets including CIFAR10 and CIFAR100 [70]. The network compression method is applied to ResNet [53], ResNeXt [152],

Regularizer	ℓ_1	ℓ_{1-2}	$\ell_{1/2}$	logsum
Regularization factor λ	$2e^{-4}$	$2e^{-4}$	$4e^{-4}$	$9e^{-5}$

Table 4.2: **The regularization factor for ℓ_1, ℓ_{1-2}, $\ell_{1/2}$, and** logsum**.**

VGG [136], and DenseNet [62] on CIFAR10 and CIFAR100, WRN [165] on CIFAR100. For ResNet20 and ResNet56 on CIFAR dataset, the residual block is the basic ResBlock with two 3×3 convolutional layers. For ResNet164 on CIFAR, the residual block is a bottleneck block. The investigated models of ResNeXt are ResNeXt20 and ResNeXt164 with carlinality 32, and bottleneck width 1. WRN has 16 convolutional layers with widening factor 10.

The training protocol of the original network is as follows. The networks are trained for 300 epochs with SGD on CIFAR dataset. The momentum is 0.9 and the weight decay factor is 10^{-4}. Batch size is 64. The learning rate starts with 0.1 and decays by 10 at Epoch 150 and 225. The ResNet50 model is loaded from the pretrained PyTorch model [115]. The models are trained with Nvidia Titan Xp GPUs. The proposed network compression method is implemented by PyTorch.

We fix the hyper parameters of the proposed method by empirical studies. The stop criterion α in Algorithm 1 is set to 0.1. The threshold $\mathcal{T}$ is set to 0.005. Unless otherwise stated, ℓ_1 regularizer is used. The regularization factors for different regularizers are listed in Table 4.2. As already mentioned, during the compression step, we set different learning rates for $\mathbf{W}$ and $\mathbf{A}$. For CIFAR10 and CIFAR100 datasets, the learning rate η of the sparsity-inducing matrix $\mathbf{A}$ during compression optimization is set to 0.1. The ratio between the learning rate of $\mathbf{W}$ and $\mathbf{A}$ is set to 0.01. That is, the learning rate η_s of $\mathbf{W}$ during compression optimization is 0.001.

4.6.1 *Results on CIFAR10*

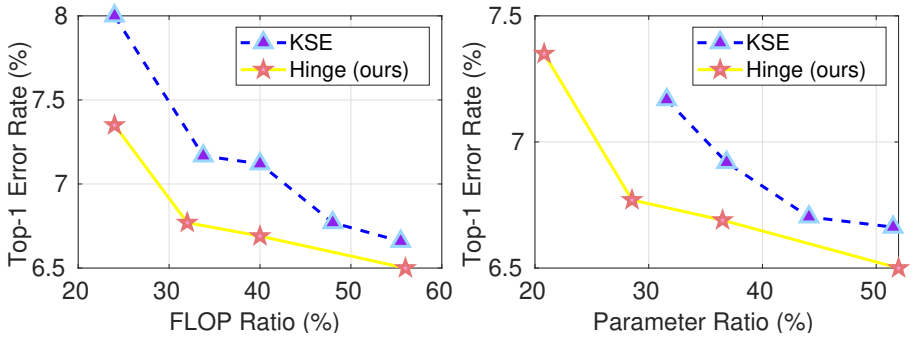
The ablation study on ResNet56 is shown in Table 4.3. Different combinations of the hyper parameters $\mathcal{T}$ and α are investigated. There are only slight changes in the results for different combinations. Anyway, when $\mathcal{T} = 0.005$ and $\alpha = 0.01$, our method achieves the lowest error rate. And we use this combination for the other experiments. As for the

Regularizer	Threshold $\mathcal{T}$	α	Top-1 error (%)
ℓ_1	0.001	0.05	6.54
ℓ_1	0.005	0.1	6.53
ℓ_1	0.001	0.05	6.66
ℓ_1	0.005	0.01	6.37
logsum	0.005	0.01	6.53
$\ell_{1/2}$	0.005	0.01	6.31
ℓ_{1-2}	0.005	0.01	6.56

Table 4.3: **Ablation study.** The proposed compression method is applied to ResNet56 and tested on CIFAR10. The compression ratio is fixed to 50%. Different regularizers and hyper parameters $\mathcal{T}$ and α are examined.

Model	Method	Top-1 / BL (%)	FLOPs (%)	Params (%)
ResNet-56	[175]	7.74/6.96	79.70	79.51
	GAL-0.6 [97]	6.62/7.64	63.40	88.20
	[81]	6.94/6.96	62.40	86.30
	NISP [163]	6.99/6.96	56.39	57.40
	CaP [110]	6.78 / 6.49	50.20	–
	ENC [67]	7.00 / 6.90	50.00	–
	AMC [55]	8.10 / 7.20	50.00	–
	KSE [95]	6.77 / 6.97	48.00	45.27
	FPGM [54]	6.74 / 6.41	47.70	–
	Hinge (ours)	**6.31** / 7.05	50.00	48.73
	KSE [95]	8.00 / 6.97	24.00	–
	Hinge (ours)	**7.35** / 7.05	24.00	20.80
ResNeXt-164	SSS [64]	5.42 / 6.41	44.38	64.38
	Hinge (ours)	**5.13** / 4.82	44.42	50.53
VGG16	[175]	6.82 / 6.75	60.90	26.66
	GAL-0.1 [97]	6.58 / 6.04	54.80	17.80
	Hinge (ours)	**6.41** / 5.98	60.93	19.95
DenseNet-12-40	GAL-0.01 [97]	5.39 / 5.19	64.70	64.40
	[175]	6.84 / 5.89	55.22	40.33
	Hinge (ours)	**5.33** / 5.26	55.60	72.46

Table 4.4: **Comparison of CIFAR10 compression results.**



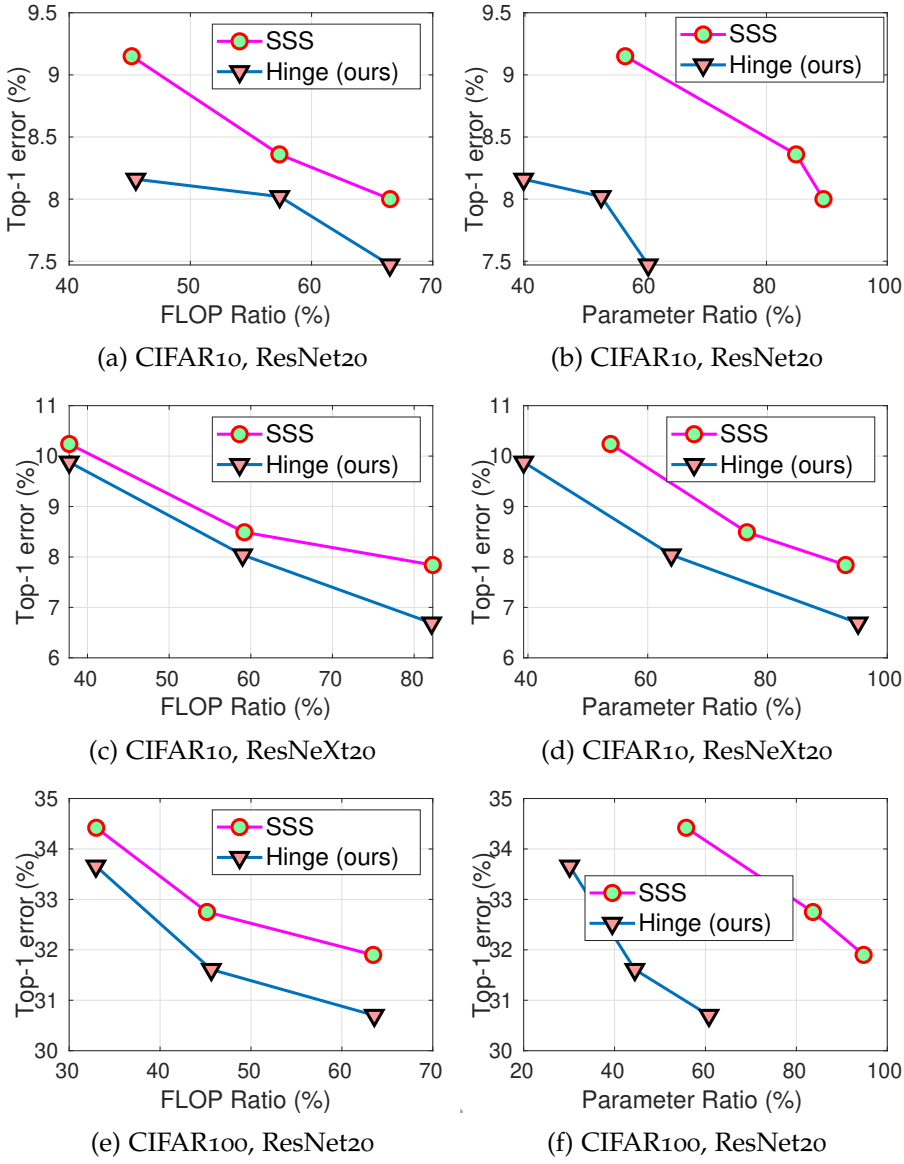
(a) FLOP ratio comparison. (b) Parameter ratio comparison.

Figure 4.4: **Comparison between KSE [95] and Hinge** under different compression ratio on CIFAR10 for ResNet56.

different regularizers, ℓ_1 and $\ell_{1/2}$ regularization are clearly better than ℓ_{1-2} and logsum. Due to the simplicity of the ℓ_1 proximal operator in contrast to the $\ell_{1/2}$, we use ℓ_1 instead of $\ell_{1/2}$ in the other experiments.

The experimental results on CIFAR10 are shown in Table. 4.4. The Top-1 error rate, the percentage of the remaining FLOPs and parameters of the compressed models are listed in the table. For ResNet56, two operating points are reported. The operating point of 50% FLOP compression is investigated by a bunch of state-of-the-art compression methods. Our proposed method achieves the best performance under this constraint. At the compression ratio of 24%, our approach is clearly better than KSE [95]. For ResNet and ResNeXt with 20 and 164 layers, our method shoots a lower error rate than SSS. For VGG and DenseNet, the proposed method reduces the Top-1 error rate by 0.41 % and 1.51 % compared with [175]. In Fig. 4.4, we compare the FLOPs and number of parameters of the compressed model by KSE and the proposed method under different compression ratios. As shown in the figure, our compression method outperforms KSE easily. Fig. 4.5a through Fig. 4.5d compare the FLOPs and number of parameters of ResNet20 and ResNext20 compressed by SSS and our method on CIFAR10. Our method establishes a lower bound for SSS. The layer-wise compression ratio of the compressed network is shown in Fig. 4.6. The overall FLOPs compression ratio is 50%. The proposed method results in a sawtooth architecture. That is, for the convolutions with the same feature dimension (*i.e.* Layer 1 to Layer 18, Layer 19 to Layer 36, and

Layer 37 to Layer54), the middle layers generally have a severer degree of compression.

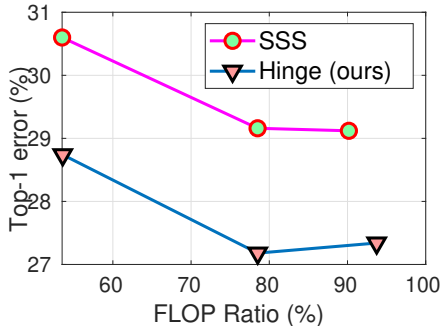


(a) CIFAR10, ResNet20 (b) CIFAR10, ResNet20

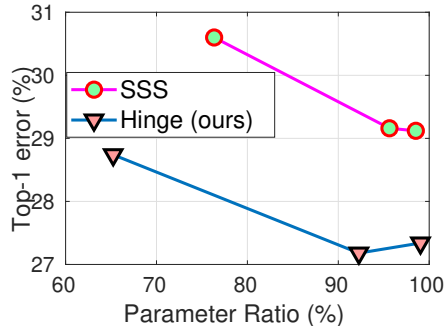
(c) CIFAR10, ResNeXt20 (d) CIFAR10, ResNeXt20

(e) CIFAR100, ResNet20 (f) CIFAR100, ResNet20

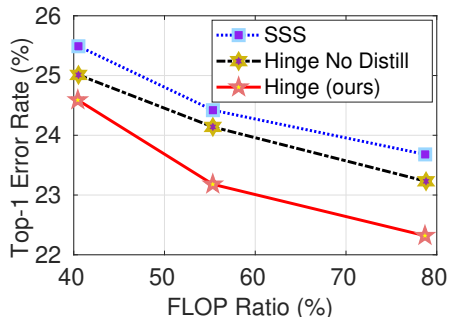
Figure 4.5: **Comparison between SSS [64] and the proposed method.**



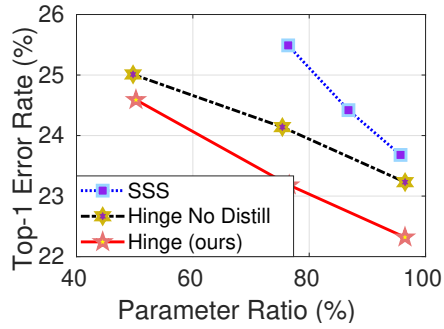
(g) CIFAR100, ResNeXt20



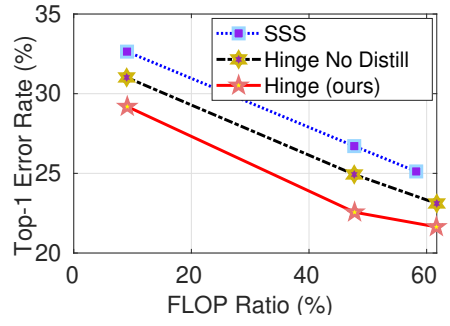
(h) CIFAR100, ResNeXt20



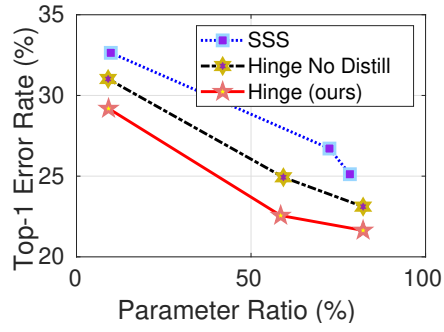
(i) CIFAR100, ResNet164



(j) CIFAR100, ResNet164



(k) CIFAR100, ResNeXt164



(l) CIFAR100, ResNeXt164

Figure 4.5: **Comparison between SSS [64] and the proposed method (continued).**

4.6.2 *Results on CIFAR100*

Table 4.5 shows the compression results on CIFAR100. For the compression of WRN, we analyze the influence of regularization factor

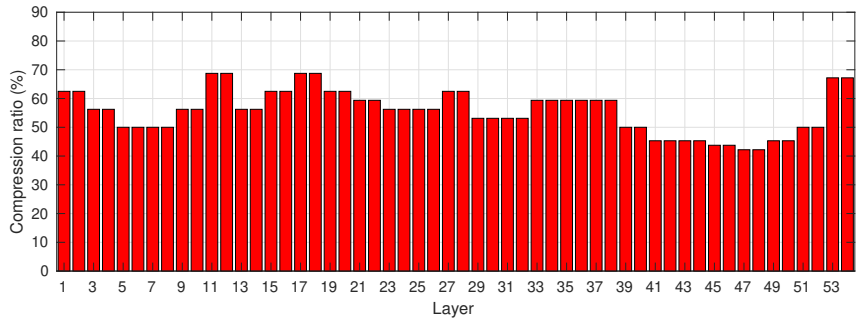


Figure 4.6: **Layer-wise compression ratio of ResNet56 on CIFAR10.**

Model	Method	Top-1 / BL (%)	FLOPs (%)	Params (%)
WRN	CGES [161]	21.97 / 21.62	75.56	–
	Hinge-NA	23.61 / 21.58	75.59	84.31
	Hinge (ours)	**21.79** / 21.58	75.61	83.29
	CGES [161]	22.75 / 21.62	57.31	–
	Hinge-NA	23.13 / 21.58	57.41	68.72
	Hinge (ours)	**22.06** / 21.58	57.39	67.80
ResNet20	SSS [64]	34.42 / 30.91	32.98	54.42
	Hinge (ours)	**33.66** / 31.17	32.94	33.64
ResNet164	SSS [64]	24.42 / 23.31	55.33	86.75
	Hinge (ours)	23.12 / 23.22	55.32	76.57
ResNeXt20	SSS [64]	30.60 / 28.00	53.51	76.34
	Hinge (ours)	**28.74** / 28.05	53.59	65.24
ResNeXt164	SSS [64]	26.71 / 23.18	47.69	72.47
	Hinge (ours)	**22.56** / 23.13	47.75	58.49

Table 4.5: **Comparison of CIFAR100 compression results.** For a fair comparison, the model size from different methods is kept to the same level. Hinge-NA stands for our hinge method without regularization factor annealing during the compression phase.

annealing during the compression phase. It is clear that with the annealing mechanism, the proposed method achieves much better performance. This is because towards the end of the compression phase, the proximal gradient solver has found quite a good neighbor of the

local minimum. In this case, the regularization factor should diminish in order for a better exploration around the local minimum. Compared with the previous group sparsity method CGES [161], our hinge method with the annealing mechanism results in better performance.

Fig. 4.5i through Fig. 4.5l compare the compression performance of SSS and our method on the 164-layer ResNet and ResNext. Even without the distillation loss, our method is already better than SSS. When the distillation loss is utilized, the proposed method brings the Top-1 error rate to an even lower level. The corresponding results for the 20-layer networks are shown in Fig. 4.5e through 4.5g.

4.7 CONCLUSION

In this chapter, we propose to hinge filter pruning and decomposition via group sparsity. By enforcing group sparsity regularization on the different structured groups, *i.e.* , columns and rows of the sparsity-inducing matrix, the manipulation of the tensor breaks down to filter pruning and decomposition, respectively. The unified formulation enables the devised algorithm to flexibly switch between the two modes of network compression, depending on the specific circumstances in the network. Proximal gradient method with gradient based learning rate adjustment, layer balancing, and regularization factor annealing are used to solve the optimization problem. Distillation loss is used in the finetuning phase. The experimental results validate the proposed method.

Part II

NEURAL ARCHITECTURE OPTIMIZATION

5 THE HETEROGENEITY HYPOTHESIS

5.1 INTRODUCTION

Different from the previous chapters on network compression, the focus of this chapter is fine-grained neural architecture optimization. Recently, neural network design has also evolved from manual design [136, 53, 62] to NAS [100, 140] and semi-automation [157, 58, 122]. State-of-the-art network designs focus on discovering the overall network architecture with regularly repeated convolutional layers. This has been the golden standard of current CNN designs. For example, Ma *et al.* mentioned that a network should have equal channel width [106]. But their analysis is limited to minimizing the memory access cost given the FLOPs for a single pointwise convolution.

The motivation of this chapter kind of contradicts the previous design heuristics. It investigates a design space that is usually overlooked and thus not fully explored, namely adjusting the layer-wise channel configurations. The channel configuration of a network is defined as the vector that summarizes the output channels of the convolutional layers. We try to answer three questions: 1) whether there exists a LW-DNA that can outperform the original one; 2) if so, how to identify it efficiently; and 3) why it can beat the regular configuration.

Question 1: The existence of LW-DNA. To answer the first question, we formally articulate the following hypothesis. ***The Heterogeneity Hypothesis:*** *For a CNN, when trained with exactly the same training protocol* (e.g. *number of epochs, batch size, learning rate schedule), there exists a layer-wise differentiated network architecture (LW-DNA) that can outperform the original network with regular layer-wise channel configurations but with a lower model complexity in term of FLOPs and parameters.*

To be specific, we aim at adjusting the numbers of channels of the convolutional layers in predefined CNNs. The other layer configurations such as kernel size and stride are not changed. Formally, consider an L-layer CNN $f(\mathbf{X}; \Theta, \mathbf{c})$, where $\mathbf{c} = (c_1, c_2, \cdots, c_L)$ is the channel configuration of all of the convolutional layers, Θ denotes the parameters in the network, and $\mathbf{X}$ is the input of the network. The heterogeneity

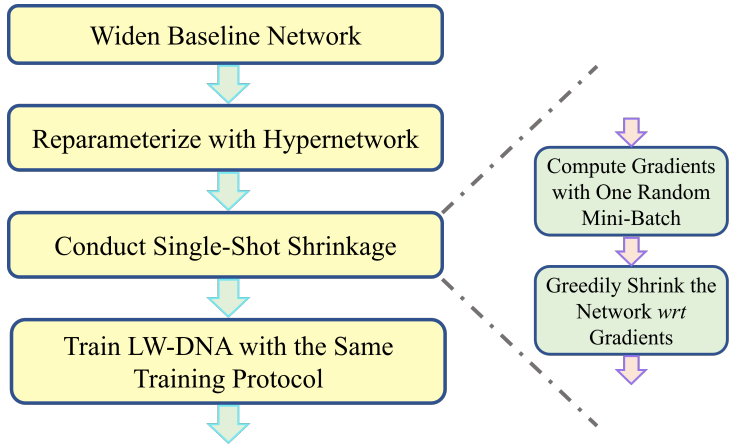


Figure 5.1: **Pipeline for identifying LW-DNA models.** Note that the single-shot shrinkage method only needs to run one random mini-batch. Then the network is shrunk after the single pass. Thus, almost no additional computational cost is introduced. This allows for fair comparison between the baseline model and the LW-DNA model.

hypothesis implies that there should exist a new channel configuration $\mathbf{c}' = (c'_1, c'_2, \cdots, c'_L)$ such that the new architecture $f'(\mathbf{X}; \Theta', \mathbf{c}')$ performs no worse than the original one. After the adjustment, the channel configurations c'_l could be either larger or smaller than the original c_l. We try to answer this question by empirical experiments.

Question 2: How to identify an LW-DNA efficiently? Note that the focus of this chapter is solely the network architectures. The influence of factors other than network architecture such as the training protocol are excluded. This choice allows for controlled experiments and a fair comparison between the possibly existing LW-DNA models and the baseline models. But we are in turn faced with the following problem. ***Problem Statement:*** *If the heterogeneity hypothesis is valid, how can we efficiently and reliably find an LW-DNA model for a CNN without additional computational cost and training time?*

To solve this problem, we are inspired by recent developments in network compression [79, 78, 91]. The pipeline of identifying LW-DNA models is shown in Fig. 5.1. In short, the LW-DNA models are identified by the single-shot shrinking of a widened and reparameterized version of the baseline network. The details are given in Sec. 5.4

Question 3: How to explain the benefits of LW-DNA? As a matter of examples, we identify LW-DNA versions of various state-of-the-art networks for three vision tasks, incl. image classification [53, 62, 59, 133, 58, 140, 141, 122], image restoration [76, 96, 169, 131], and visual tracking [10]. Interestingly, the identified LW-DNA models consistently outperform the baselines even with lower model complexities in terms of FLOPs and number of parameters. We try to explain this phenomenon from several perspectives.

1. CNNs are redundant. So it is possible to find a layer-wise specific channel configuration comparable with the baseline under lower model complexity.

2. As shown in Fig. 5.4, some layers of the LW-DNA models have more channels than the baseline. Indeed, the lower layers tend to be strengthened with more channels. It might be those layers that play the essential role in improving the network accuracy.

3. The accuracy gain of the LW-DNA models might be related to overfitting by the baseline models. We derive this conjecture from several observations. **I.** By comparing the training and testing curves of an LW-DNA model and its baseline in Fig. 5.3d, we find that towards the end of the training, the identified LW-DNA model shows a higher training error but a lower testing error, *i.e.* improved generalization. This phenomenon is consistent across different datasets. This also matches the observations from the pioneering unstructured pruning, like a brain surgeon trying to boost network generalization after brain damage [75, 51]. **II.** The accuracy gain of an LW-DNA model is larger for smaller datasets (*i.e.* Tiny-ImageNet) that are easier to get overfitted to, compared with larger datasets (*i.e.* ImageNet). **III.** On the same dataset (*i.e.* ImageNet), it is easier to identify an LW-DNA model version for larger networks (*i.e.* ResNet50) than for smaller networks (*i.e.* MobileNetV3).

The contributions of this chapter can be summarized as follows. **First**, it demonstrates the possibility of identifying a superior version of a network by only adjusting the channel configuration of the network. This could be used as a post-searching mechanism complementary to semi- or fully automated neural architecture search. **Secondly**, a method that can identify LW-DNA models almost without additional

computational cost and training time is proposed. This method only needs the computation of one random batch. **Thirdly**, the possible reason for the improved performance of an LW-DNA is explained by observing the experimental results.

5.2 RELATED WORK

In this section, we review the works about the lottery ticket hypothesis. Other related works about NAS, hypernetworks, and network pruning could be found in Subsec. 3.2.2, Subsec. 3.2.3, and SubSec. 2.2.1. In this chapter, we try to adjust the channel configuration of the network, which can be regarded as a method complementary to NAS.

5.2.1 *The lottery ticket hypothesis*

The heterogeneity hypothesis is reminiscent of the Lottery Ticket Hypothesis (LTH) [34], which addresses the existence of sparse subnetworks that can match the test accuracy of randomly-initialized dense networks. The winning ticket is identified by greedily pruning single elements of weight parameters with smallest magnitude. Following works try to extend [123], theoretically prove [107], understand [179], and improve the training process [130] of Lottery Ticket Hypothesis (LTH). The unstructured pruning breaks the dynamical isometry in the network [78]. The core problem is the trainability of the sparse subnetworks and the gradient flow in the subnetworks [78]. In contrast, the heterogeneity hypothesis focuses on adjusting the channel configuration of the network. Since the weight elements of an entire channel are pruned together, there is no irregular kernel in the pruned network. Gradient flow is no longer a problem in this scenario.

5.3 PRELIMINARIES

5.3.1 *Hints from network compression*

Recent network compression methods shed light on the existence of advantageous layer-wise specific networks [101, 91, 30]. Those methods can result in shrunk networks with layer-wise specific channel configurations. Some works [101] report accuracy gains of the pruned

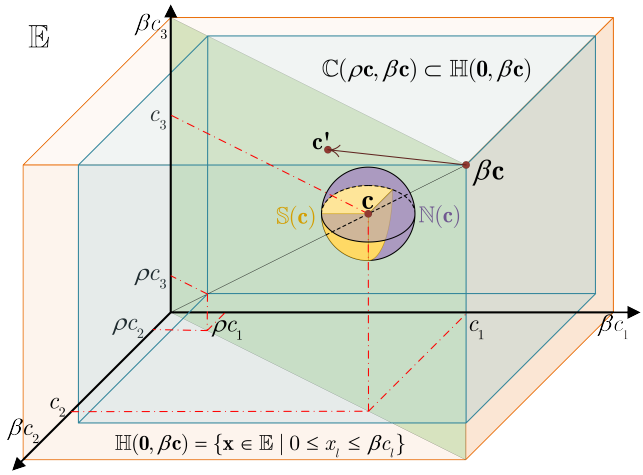


Figure 5.2: **Illustration of the configuration space.** The proposed method identifies layer-specific channel configurations within the enlarged and constrained subspace $\mathbb{C}(\rho\mathbf{c}, \beta\mathbf{c})$. Compared with searching within the constrained neighborhood $\mathbb{S}(\mathbf{c})$ of $\mathbf{c}$, the enlarged configuration space makes it possible to develop a straightforward shrinkage criterion.

network over the width-scaled versions of ResNet and MobileNets [53, 59, 133]. Yet, since the advantageous networks are identified in a network compression sense, thus with an accuracy drop compared with the uncompressed network, it still remains unknown whether there exists a layer-wise specific network that can compete with the original one. A recent work [91] reports an accuracy gain over uncompressed MobileNets on Tiny-ImageNet. Yet, further investigations on larger datasets are not conducted. Moreover, the compact networks are usually derived with training protocols different from those used for the baseline network, *e.g.* additional searching stage, larger batch size, or prolonged fine-tuning stage. It remains unknown how the layer-wise specific channel configurations benefit the network.

5.3.2 *Notations and definitions*

Notation. In this chapter, bold lowercase letters such as $\mathbf{c}$, $\mathbf{x}$, and $\mathbf{z}$ are used to denote vectors while bold capital letters such as $\mathbf{O}$, $\mathbf{Z}$, $\mathbf{W}$ are used to denote matrices and higher dimensional tensors. The vectors, matrices, and higher dimensional tensors are indexed by subscripts.

Greek letters such as α, β denote constant scalars. The configuration vector and configuration space are formally defined as follows.

Definition 1 (Channel configuration vector). Consider an L-layer CNN. The channel configuration vector of the network is defined as an L-dimensional vector that summarizes the number of output channels of the network, *i.e.*

$$\mathbf{c} = (c_1, c_2, \cdots, c_L), \tag{5.1}$$

where c_l denotes the number of output channels in the l-th layer.

Definition 2 (Configuration space). The configuration space $\mathbb{E}$ is a subspace of Euclidean space that contains the allowable channel configuration vectors. (See Fig. 5.2 for one example of the configuration space.)

The dimension of the configuration vectors depends on the number of convolutional layers in the network. Take VGG11 for example. The configuration vector is an 8-dimensional vector, *i.e.* ,

$$\mathbf{c}_{vgg} = (64, 128, 256, 256, 512, 512, 512, 512)\,. \tag{5.2}$$

As in this example, the configuration vector is regular and its elements are dependent on each other in the sense that most of them are repeated. For image classification networks, the golden standard is to repeat building blocks with the same configuration up to the point where the spatial dimension of the feature map gets reduced. Some efficient designs for mobile devices introduce a width multiplier α to adapt to constrained resource requirements, which results in a scaled configuration vector, *i.e.* ,

$$\mathbf{b} = (\alpha c_1, \alpha c_2, \cdots, \alpha c_L),\ \alpha < 1\,. \tag{5.3}$$

5.3.3 *Problem formulation and recast*

Since the configuration vector is manually fixed, it is not guaranteed to be optimal. In this chapter, we explore the corresponding configuration design space. The aim is to demonstrate that there is an irregular configuration vector $\mathbf{c}'$ that can compete with the original, while offering reduced model complexity. To achieve that, we propose an algorithm which can adjust (increase or decrease) the elements of the configuration vector $\mathbf{c}$ while controlling the model complexity. As shown in Fig. 5.2,

such an adjustment procedure best searches in the neighborhood of the vector, *i.e.*

$$\mathbb{N}(\mathbf{c}) \subset \mathbb{E}\,. \tag{5.4}$$

After the adjustment, an element of the configuration vector $\mathbf{c}$ can be either increased or decreased, which corresponds to growing or shrinking the l-th layer of the network. Shrinkage criteria can be defined on the existing network and network shrinkage algorithm could applied. The limitation of a shrinkage algorithm on the original network is that it can only explore a subspace of the neighborhood, *i.e.*

$$\mathbb{S}(\mathbf{c}) = \{\mathbf{x} \in \mathbb{N}(\mathbf{c}) | x_l \leq c_l\} \subset \mathbb{N}(\mathbf{c})\,. \tag{5.5}$$

But we do not want to be restricted to shrinkage only. Instead, it is desirable to do both network shrinkage and growth at the same time for the configuration vector adjustment.

We circumvent this problem by recasting it as a shrinkage problem in a larger configuration space which is obtained by widening the width of the network with a width multiplier $\beta > 1$. The new searching space $\mathbb{H}$ is a hyperrectangle delimited by the zero vector $\mathbf{0}$ and the up-scaled configuration vector $\beta\mathbf{c}$ in the high-dimensional space, *i.e.*

$$\mathbb{H}(\mathbf{0}, \beta\mathbf{c}) = \{\mathbf{x} \in \mathbb{E} | 0 \leq x_l \leq \beta c_l\} \subset \mathbb{E}\,. \tag{5.6}$$

The searching algorithm then starts from the up-scaled vector $\beta\mathbf{c}$ and reduces the value of its l-th element greedily according to the significance of the channels in the corresponding convolutional layer.

5.4 METHODOLOGY

After introducing the preliminaries and the designing considerations in the last section, the algorithm used to identify LW-DNA models is explained in this section. The pipeline is already shown in Fig. 5.1. The identifying procedure proceeds as follows. 1) Reparameterize the widened baseline network with hypernetworks. The outputs of the hypernetworks act as the weight parameters of the baseline network. The inputs of the hypernetwork serve as the handle to shrink the network. 2) Compute the gradients of the hypernetwork input, *i.e.* the latent vectors, with one random batch. 3) Sparsify the latent vectors greedily according to the magnitude of their gradients. 4) Compute the weight parameters with the sparsified latent vectors. 5) Train the

resultant network from scratch with the same training protocol as the baseline network. And in the following, we explain some of the key steps in detail.

5.4.1 *Reparameterizing with hypernetworks*

The network shrinkage method is explained in this section. Instead of directly shrinking the baseline network, we first widen it and reparameterize it with hypernetworks [101, 91]. The reparameterization is adopted based on the following considerations. The hypernetworks bring the shrinkage problem into a latent space. Removing a channel is equivalent to deleting a single element of the latent vector, which converts the problem of dealing with elements in the whole channel to an easier one of dealing with a single element in the latent vector. In addition, it provides a straightforward extension of single-shot shrinkage [79] to channel pruning (See Subsec 5.4.2). And single-shot shrinkage is the core of avoiding additional computational cost when identifying LW-DNA models. The latent vector sharing mechanism in the hypernetworks also makes it possible to deal with various state-of-the-art networks.

Consider the L-layer CNN that is brought to the larger configuration space $\mathbb{H}(\mathbf{0}, \beta\mathbf{c})$ as in Eqn. (5.6). The weight parameter of the l-th convolutional layer of the CNN has the dimension of $\beta c_l \times \beta c_{l-1} \times w \times h$, where βc_l, βc_{l-1}, and $w \times h$ denotes the output channel, input channel, and kernel size of the layer. For the simplicity of notation, let $n = \beta c_l$ and $c = \beta c_{l-1}$. We use the hypernetwork proposed in Subsec. 3.3.1 to reparameterize the convolutional layers of the widened network.

5.4.2 *Single-shot shrinkage*

After reparameterizing the network with hypernetworks, the parameters in the network are first randomly initialized [16]. Then the single-shot shrinkage method is used to adjust the width of the network.

Consider a single mini-batch $\{\mathbf{X}_i, \mathbf{Y}_i\}$ from the dataset. The output of the network is computed as

$$\hat{\mathbf{Y}}_i = f(\mathbf{X}_i; \Theta, \mathbf{z}), \tag{5.7}$$

where $\mathbf{z}$ denotes the latent vectors and Θ is the parameter set that contains $\mathbf{W}_1$ and $\mathbf{W}_2$. The loss is computed as

$$\mathcal{L} = \mathcal{L}\left(\mathbf{Y}_i, f(\mathbf{X}_i; \Theta, \mathbf{z})\right) . \tag{5.8}$$

Then the gradients of the loss function with respect to the latent vectors are computed as

$$\nabla\mathcal{L} = \frac{\partial\mathcal{L}\left(\mathbf{Y}_i, f(\mathbf{X}_i; \Theta, \mathbf{z})\right)}{\partial\mathbf{z}} . \tag{5.9}$$

The magnitude of the gradients is used as the criterion to sparsify the latent vectors. The elements whose gradient magnitude is smaller than a threshold are removed. The threshold is determined by a binary search algorithm, which allows the resultant network to reach a predefined FLOP target. The resultant network is the final LW-DNA model and is trained from scratch with the same training protocol as the baseline model.

The single-shot shrinkage method is inspired by single-shot pruning of weight elements [79]. But the original method is single element oriented. It removes single weight parameters in the network and results in unstructured kernels. It remains to be explored how to transform the single-shot method to network shrinkage. The hypernetworks provide such a connection. By resorting to hypernetworks, the shrinkage is conducted on the latent space whose elements correspond to channels in the network and serve as the agent for shrinkage. Deleting an element of the latent vector is equivalent to remove a channel in the network. Thus, sparsifying the latent vectors according to their gradients is a natural transferring of the single-shot method in [79].

5.4.3 *Knowledge distillation*

For image classification, besides the cross-entropy loss function, a distillation term is also used, *i.e.*

$$\mathcal{L} = (1-\alpha)\mathcal{L}_{ce}(\mathbf{y}, \sigma(\mathbf{z}_s)) + 2\alpha T^2 \mathcal{L}_{ce}\left(\sigma\left(\mathbf{z}_s/T\right), \sigma\left(\mathbf{z}_t/T\right)\right), \tag{5.10}$$

where $\mathcal{L}_{ce}(\cdot)$ is the cross-entropy loss function, $\sigma(\cdot)$ is the softmax function, $\mathbf{y}$ is the class label, $\mathbf{z}_s$ and $\mathbf{z}_t$ are the logit outputs of LW-DNA model and the teacher [57]. We use fixed parameters $\alpha = 0.4$ and $T = 4$. The teacher is the pretrained widened version of the baseline network. Knowledge distillation is not used for experiments

on ImageNet because the execution of the teacher network in this case also consumes considerable time and GPU resources.

5.4.4 *Constraining model complexity*

Model complexity is measured in terms of FLOP and parameter count. The target is to find a model that has both fewer FLOPs and parameters while achieving improved accuracy. Yet, the two metrics are not always consistent with each other. For example, when the FLOPs target is set, a parameter over-pruned model might be observed in some of the experiments, which could lead to inferior performance. Thus, a new hyper-parameter ρ is introduced which controls the minimum percentage of remaining channels in convolutional layers. In this way, the search space $\mathbb{C}(\rho\mathbf{c}, \beta\mathbf{c})$ is a confined subspace of the original search space $\mathbb{H}(\mathbf{0}, \beta\mathbf{c})$, *i.e.*

$$\mathbb{C}(\rho\mathbf{c}, \beta\mathbf{c}) = \{\mathbf{x} \in \mathbb{E} | \rho c_l \leq x_l \leq \beta c_l\} \subset \mathbb{H}(\mathbf{0}, \beta\mathbf{c}) \,. \tag{5.11}$$

A similar hyper-parameter τ is introduced for the final linear layers of image classification networks. The hyper-parameters ρ and τ are termed convolutional percentage and linear percentage in this thesis, respectively. During the pruning, the FLOP budget is fixed. By tuning the hyper-parameters ρ and τ, the algorithm is able to find networks with the same FLOPs but varying parameter budgets.

5.5 EXPERIMENTAL RESULTS

The experimental results are shown in this section. We try to identify LW-DNA for various state-of-the-art networks including ResNet [53], RegNet [122], MobileNets [59, 133, 58], EfficientNet [141], MnasNet [140], DenseNet [62], SRResNet [76], EDSR [96], DnCNN [169], and U-Net [131]. The identified LW-DNA model and the baseline network are trained with exactly the same training protocol. The details of the training protocol for different tasks are given in the supplementary. Knowledge distillation [57] is used for image classification on CIFAR [70] and Tiny-ImageNet[28] (Baseline KD, DHP KD [91], and LW-DNA model). The balancing hyperparameter and temperature are set to 0.4 and 4, respectively. The teacher is the pretrained widened version of the baseline network. Knowledge distillation is not used for experiments on

ρ	τ	Top-1	FLOPs [G]	Params [M]
0.1	0.4	47.02	0.046	0.948
0.1	0.45	46.66	0.046	0.986
0.2	0.4	46.94	0.0459	1.210
0.2	0.45	46.44	0.046	1.265

Table 5.1: **Ablation study of the hyper-parameters ρ and τ on MobileNetV1.**

ImageNet because the execution of the teacher network in this case also consumes considerable time and GPU resources.

5.5.1 *Image Classification*

Ablation study on Tiny-ImageNet. The hyper-parameters ρ and τ are essential to control the model complexity and performance of the optimized architectures. Thus, an ablation study of the hyper-parameters ρ and τ is shown in Table 5.1. The experiments are conducted for MobileNetV1 on Tiny-ImageNet. The FLOPs budget is fixed for the experiments. Two conclusions can be drawn from the result. **I.** By increasing the hyper-parameters ρ and τ, the model complexity is also increased. And the accuracy of the network is also improved. **II.** All of the results in Table 5.1 are better than Baseline KD in Table 5.2, which shows the robustness of ρ and τ. Based on the experience on Tiny-ImageNet, we set $\rho = 0.4$ and $\tau = 0.45$ for ImageNet experiments. Quite surprising, this combination works well across the three investigated networks (ResNet50, RegNet, and MobileNetV3).

Result. The results of image classification networks are compared in Table 5.2 for ImageNet and Tiny-ImageNet and Table 5.3 for CIFAR, respectively. We have several key observations. **I.** The identified LW-DNA models outperform the original network (denoted as Baseline or Baseline KD when knowledge distillation is used) with lower model complexity in terms of both FLOPs and number of parameters. This is a direct support for the Heterogeneity Hypothesis. **II.** The accuracy of the baseline network can be improved by knowledge distillation. Yet, the improved baseline still performs worse than LW-DNA. This shows the robustness of LW-DNA, *i.e.* not affected by a specific training technique. **III.** The improvement of LW-DNA scales up to large-scale datasets,

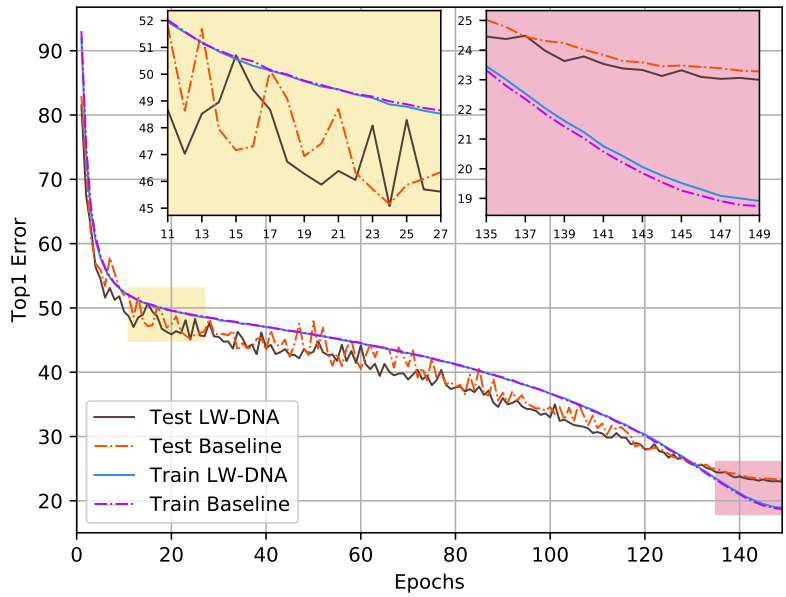
Network	Method	Top-1 Error (%)	FLOPs [G] / Ratio (%)	Params [M] / Ratio (%)
ImageNet [28]				
ResNet50 [53]	Baseline	23.28	4.1177 / 100.0	25.557 / 100.0
	MutualNet [156]	21.40	4.1177 / 100.0	25.557 / 100.0
	LW-DNA	23.00	3.7307 / 90.60	23.741 / 92.90
	MetaPruning [101]	23.80	3.0000 / 72.86	–
	AutoSlim [162]	24.00	3.0000 / 72.86	23.100 / 90.39
RegNet [122] X-4.0GF	Baseline	23.05	4.0005 / 100.0	22.118 / 100.0
	LW-DNA	22.74	3.8199 / 95.49	15.285 / 69.10
MobileNetV3 small [58]	Baseline	34.91	0.0612 / 100.0	3.108 / 100.0
	LW-DNA	34.84	0.0605 / 98.86	3.049 / 98.11
Tiny-ImageNet				
MobileNetV1 [59]	Baseline	51.87	0.0478 / 100.0	3.412 / 100.0
	Baseline KD	48.00	0.0478 / 100.0	3.412 / 100.0
	DHP KD	46.70	0.0474 / 99.16	2.267 / 66.43
	LW-DNA	46.44	0.0460 / 96.23	1.265 / 37.08
MobileNetV2 [133]	Baseline	44.38	0.0930 / 100.0	2.480 / 100.0
	Baseline KD	41.25	0.0930 / 100.0	2.480 / 100.0
	DHP KD	41.06	0.0896 / 96.34	2.662 / 107.34
	LW-DNA	40.74	0.0872 / 93.76	2.230 / 89.90
MobileNetV3 large [58]	Baseline	45.53	0.0860 / 100.0	4.121 / 100.0
	Baseline KD	38.21	0.0860 / 100.0	4.121 / 100.0
	DHP KD	38.14	0.0856 / 99.53	3.561 / 86.42
	LW-DNA	37.45	0.0797 / 92.67	3.561 / 86.43
MobileNetV3 small [58]	Baseline	47.55	0.0207 / 100.0	2.083 / 100.0
	Baseline KD	41.52	0.0207 / 100.0	2.083 / 100.0
	DHP KD	41.46	0.0192 / 92.75	1.078 / 51.76
	LW-DNA	41.35	0.0178 / 85.99	1.799 / 86.36
MnasNet [140]	Baseline	51.79	0.0271 / 100.0	3.359 / 100.0
	Baseline KD	48.17	0.0271 / 100.0	3.359 / 100.0
	DHP KD	48.10	0.0264 / 97.42	2.512 / 74.79
	LW-DNA	46.85	0.0250 / 92.25	1.258 / 37.45

Table 5.2: **Image classification results on ImageNet and Tiny-ImageNet.** Baseline and Baseline KD denote the original network trained without and with knowledge distillation respectively. DHP-KD is the DHP version trained with knowledge distillation. For ImageNet classification, knowledge distillation is not used to train the network while for Tiny-ImageNet classification, knowledge distillation is used.

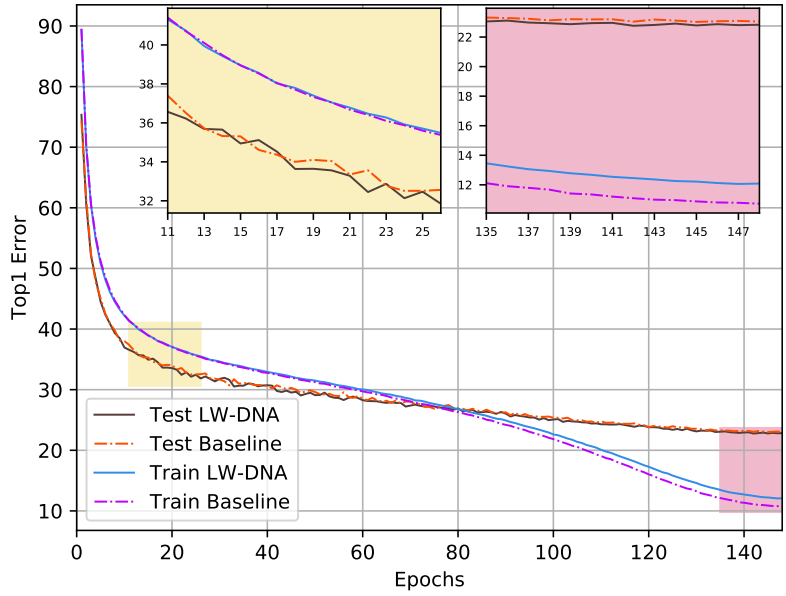
Network	Method	Top-1 Error (%)	FLOPs [G] / Ratio (%)	Params [M] / Ratio (%)
CIFAR100				
RegNet [122] Y-200MF	Baseline	21.94	0.2259 / 100.0	2.831 / 100.0
	Baseline KD	19.87	0.2259 / 100.0	2.831 / 100.0
	LW-DNA	19.87	0.2095 / 92.74	1.524 / 53.85
RegNet [122] Y-400MF	Baseline	21.65	0.4585 / 100.0	3.947 / 100.0
	Baseline KD	18.71	0.4585 / 100.0	3.947 / 100.0
	LW-DNA	18.65	0.4468 / 97.45	2.466 / 62.48
RegNet [122] X-200MF	Baseline	23.62	0.2255 / 100.0	2.353 / 100.0
	Baseline KD	21.38	0.2255 / 100.0	2.353 / 100.0
	LW-DNA	21.19	0.2075 / 92.02	1.239 / 52.68
RegNet [122] X-400MF	Baseline	21.75	0.4698 / 100.0	4.810 / 100.0
	Baseline KD	19.06	0.4698 / 100.0	4.810 / 100.0
	LW-DNA	18.81	0.4610 / 98.13	4.404 / 91.56
EfficientNet [141]	Baseline	20.74	0.4161 / 100.0	4.136 / 100.0
	Baseline KD	19.73	0.4161 / 100.0	4.136 / 100.0
	LW-DNA	19.54	0.3850 / 92.53	2.121 / 51.28
DenseNet40 [62]	Baseline	26.00	0.2901 / 100.0	1.100 / 100.0
	Baseline KD	22.84	0.2901 / 100.0	1.100 / 100.0
	LW-DNA	22.46	0.2638 / 90.93	1.016 / 92.35
CIFAR10				
DenseNet40 [62]	Baseline	5.50	0.2901 / 100.0	1.059 / 100.0
	Baseline KD	4.88	0.2901 / 100.0	1.059 / 100.0
	LW-DNA	4.87	0.2632 / 90.73	0.963 / 90.87
ResNet56 [53]	Baseline	5.74	0.1274 / 100.0	0.856 / 100.0
	Baseline KD	5.73	0.1274 / 100.0	0.856 / 100.0
	LW-DNA	5.49	0.1262 / 99.06	0.536 / 62.62

Table 5.3: **Image classification results on CIFAR.** Baseline and Baseline KD denote the original network trained without and with knowledge distillation, respectively. DHP-KD is the DHP version trained with knowledge distillation. Knowledge distillation is used to train LW-DNA models.

i.e. ImageNet. For the ImageNet experiment, we set $\rho = 0.4$ and $\tau = 0.45$ by the ablation study on Tiny-ImageNet. This hyper-parameter combination works well across the three investigated networks. The success on ImageNet and the robustness of the hyper-parameters imply the wide existence of LW-DNA models and the ease of finding them. **IV.** MutualNet is a training scheme when applied to a specific network, which could be combined with our work.

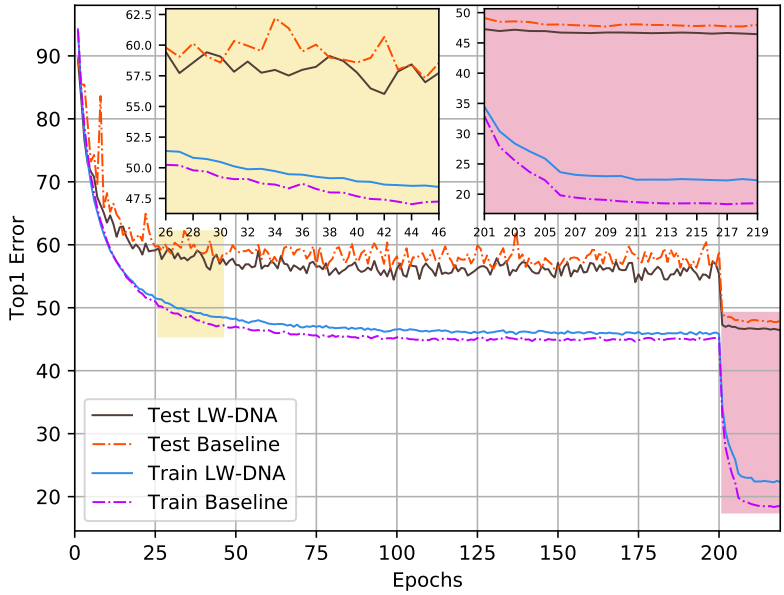


(a) ResNet50, ImageNet.

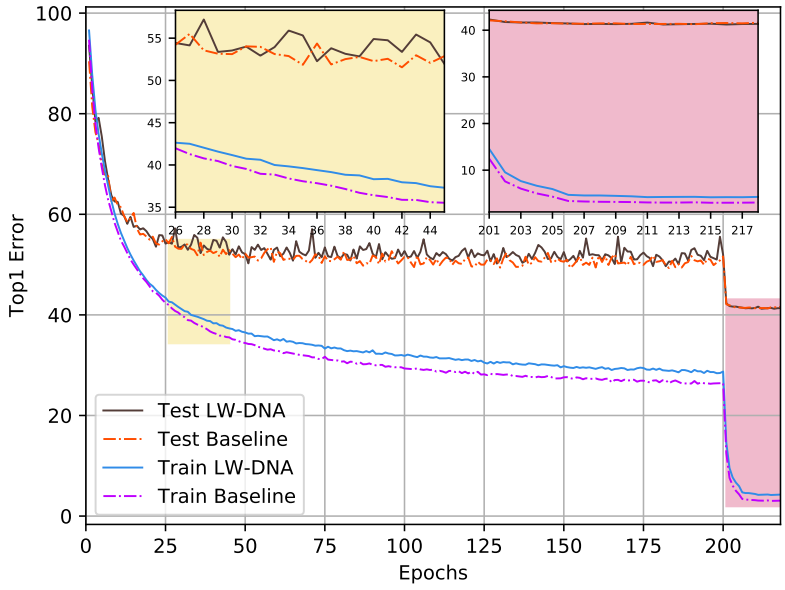


(b) RegNet-4GF, ImageNet.

Figure 5.3: **Training and testing log of LW-DNA and baseline networks.**



(c) MobileNetV1, Tiny-ImageNet.

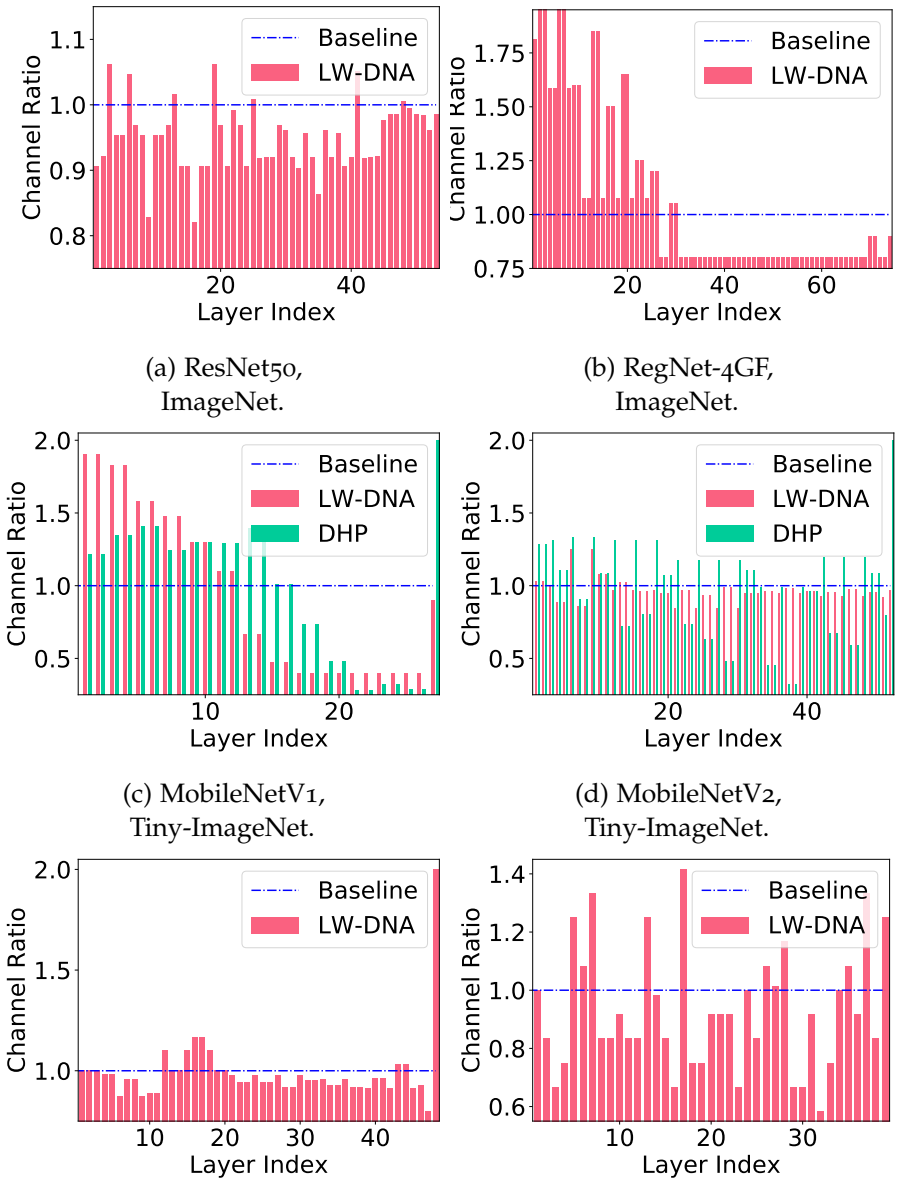


(d) MobileNetV3 small, Tiny-ImageNet.

Figure 5.3: **Training and testing log of LW-DNA and baseline networks (continued).**

***Proximal gradient descent* vs. *single-shot shrinkage*.** Besides single-shot shrinkage, there are also other candidate methods to prune networks, *e.g.* PGD. The choice of single-shot shrinkage is based on the following considerations. First, it is extremely computation-efficient. Only one random batch is used to identify the LW-DNA models. This meets the design requirements of introducing no computational cost. This consistence makes it possible to identify the importance of the architecture of LW-DNA models while controlling the other factors. Secondly, by analyzing the closed-form solution to the proximal operator with ℓ_1 regularization, *i.e.* the soft-thresholding operator, we find that PGD tends to diminish the elements of the latent vectors with the approximately consistent speed. As a results, the final magnitude of the elements has some kind of relationship with the initial magnitude. Therefore, if the initialization of an element is large, it is likely that the final magnitude is still relatively large. The distribution of the latent vector in Layer 6 of MobileNetV2 during the PGD optimization is shown in Fig. 5.5. The final distribution is related to the initialization. Thus, it becomes reasonable to shrink the latent vectors at initialization.

The benefits of LW-DNA models are analyzed by several observations of the experimental results. **I.** The percentage of remaining channels is shown in Fig. 5.4. Some layers of the LW-DNA networks are strengthened. This might contribute to the improved performance of LW-DNA. **II.** As shown in Fig. 5.3d, towards the end of the training, the LW-DNA models shoot a lower test error with increased training error. The improved generalization on the test set comes with reduced model complexity and lower training accuracy. This phenomenon is consistent with the pioneering unstructured pruning methods [75, 51] that try to balance model complexity and overfitting. The same phenomenon on both unstructured pruning and structured pruning points to a common underlying factor. **III.** The accuracy gain of LW-DNA on Tiny-ImageNet is larger than ImageNet. As known, smaller datasets are easier to be overfitted to. **IV.** On ImageNet, it is easier to identify LW-DNA models for ResNet50 and RegNet than MobileNetV3. Since the larger models ResNet50 and RegNet contain more redundancy, it is easier for them to overfit the dataset. Based on the above observations, we conjecture that the improvement of LW-DNA model might be related to model overfitting.



(a) ResNet50, ImageNet.

(b) RegNet-4GF, ImageNet.

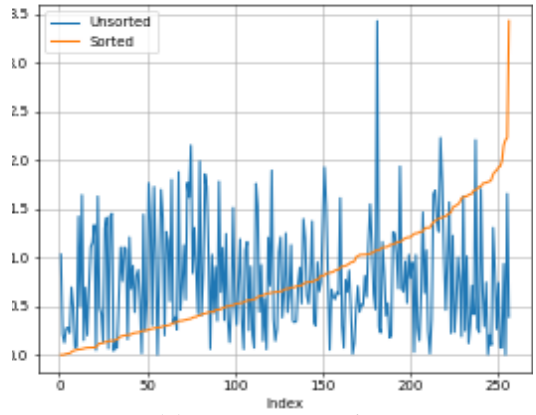
(c) MobileNetV1, Tiny-ImageNet.

(d) MobileNetV2, Tiny-ImageNet.

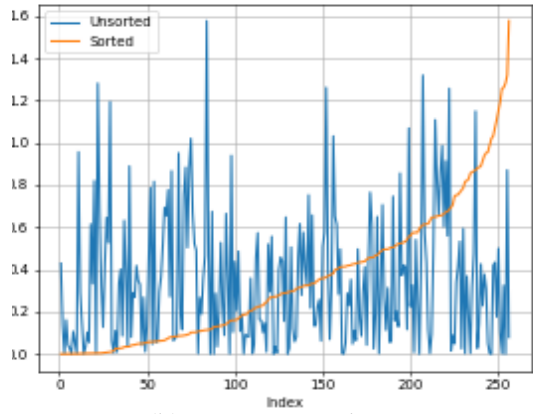
(e) MobileNetV3-Large, Tiny-ImageNet.

(f) DenseNet, CIFAR100.

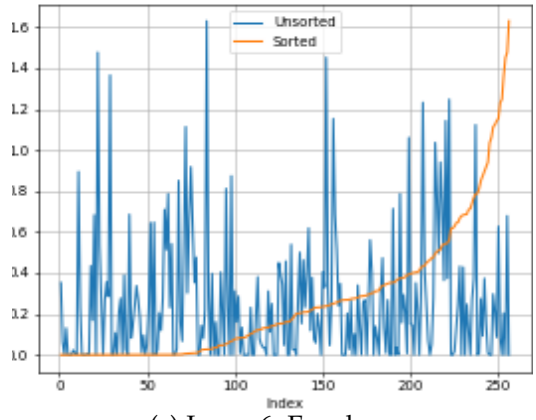
Figure 5.4: **Percentage of remaining output channels of LW-DNA models** over the baseline network.



(a) Layer 6, Epoch 1.



(b) Layer 6, Epoch 10.



(c) Layer 6, Epoch 17.

Figure 5.5: **The distribution of the latent vectors in MobileNetV2** during the PGD optimization of DHP.

Metric	DiMP-Baseline	DiMP-LW-DNA
TrackingNet [112]		
Precision	68.06	68.27
Norm. Prec. (%)	79.70	79.64
Success (AUC) (%)	73.77	73.83
LaSOT [33]		
Precision	54.97	57.30
Norm. Prec. (%)	63.70	65.82
Success (AUC) (%)	55.87	57.43

Table 5.4: **Tracking test results.** DiMP-LW-DNA and DiMP-Baseline use the identified LW-DNA and baseline version of ResNet50, respectively.

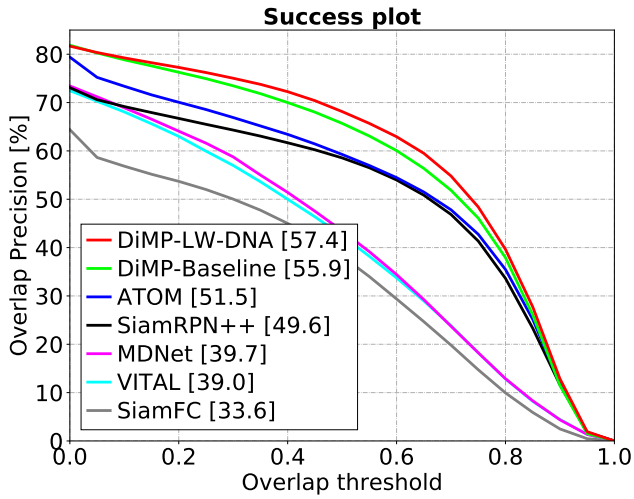


Figure 5.6: **Success plot on the LaSOT dataset for visual tracking.**

5.5.2 *Visual Tracking*

To validate the generalization ability of the identified LW-DNA, we apply the LW-DNA and baseline version of ResNet50 to visual tracking. State-of-the-art tracking workflow DiMP [10] is used as the test bed. For a fair comparison, the LW-DNA and the baseline are trained with the same protocol. They are first pretrained on ImageNet then finetuned following the DiMP workflow. In Table 5.4, the networks are compared on two datasets, *i.e.* TrackingNet [112] and LaSOT [33]. On the smaller dataset TrackingNet, LW-DNA version slightly beats the baseline while

Metric		SRResNet [76]		EDSR [96]	
		Baseline	LW-DNA	Baseline	LW-DNA
PSNR (dB)	Set5 [9]	32.02	32.07	32.10	32.13
	Set14 [166]	28.50	28.51	28.55	28.61
	B100 [108]	27.52	27.52	27.55	27.59
	Urban100 [63]	25.88	25.88	26.02	26.09
	DIV2K [2]	28.84	28.85	28.93	28.99
FLOPs $[G]$		32.81	28.79	90.37	55.44
Ratio (%)		100.0	87.75	100.0	61.34
Params $[M]$		1.53	1.36	3.70	2.84
Ratio (%)		100.0	88.43	100.0	76.94

Table 5.5: **Results on single image super-resolution networks.** The upscaling factor is $\times 4$.

on the larger dataset LaSOT, LW-DNA outperforms the baseline elegantly. The success plot on LaSOT is shown in Fig. 5.6. As shown there, DiMP-LW-DNA is consistently better than DiMP-Baseline and other state-of-the-art tracking methods across the range of overlap threshold. In conclusion, the results show that **the benefits of LW-DNA can be transferred to other vision tasks**.

5.5.3 *Image Restoration*

Table 5.5 shows the results on super-resolution networks. For EDSR, the LW-DNA models perform better than the baseline but with significant reduction of model complexity. On the large test dataset Urban100 and DIV2K, the LW-DNA model of EDSR leads to nearly 0.1dB PSNR gain over the baseline. For SRResNet, LW-DNA achieves slightly reduction of model complexity without drop of PSNR. More results on image denoising are shown in the supplementary. In conclusion, the results **validate the existence of LW-DNA models for low-level vision networks**.

5.6 CONCLUSION

In this chapter, we state the heterogeneity hypothesis which in essence is the existence of advantageous LW-DNA models for a predefined network architecture. We try to validate the hypothesis by empirical

studies. In order to single out the importance of the network architecture, the training protocol is kept the same for the baseline and the LW-DNA models. This is achieved by converting the problem of identifying LW-DNA to a network shrinkage problem and designing an efficient shrinkage algorithm. The experiments on various network architectures and vision tasks demonstrate the benefits of the identified LW-DNA models. By examining the results, we conjecture that the advantage of the LW-DNA model might be related to model overfitting.

Part III

COMPUTATIONAL PROCEDURE OPTIMIZATION

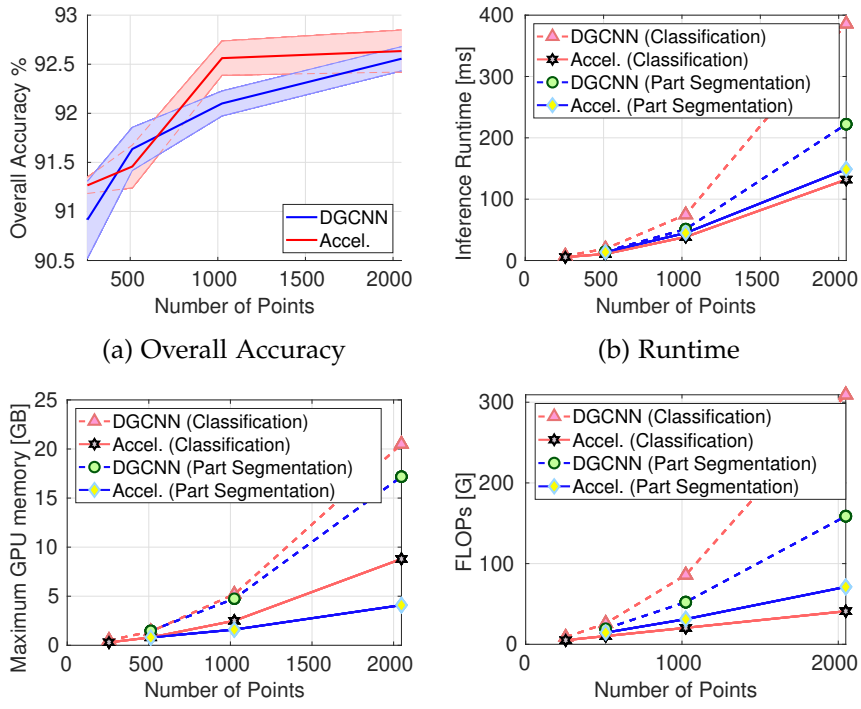
6 TOWARDS EFFICIENT GCN

6.1 INTRODUCTION

Different from the previous chapters where CNNs are investigated, the focus in this chapter is GCNs. Specifically, we try to optimize the computational procedure in GCNs for learning on point clouds. Recently, GCNs [27, 14, 36, 164, 135, 174, 146] have achieved state-of-the-art performances in 3D representation learning on point clouds for classification [120, 121], part segmentation [15], semantic segmentation [148, 61], and surface reconstruction [49]. A typical GCN is composed of a stack of Multilayer Perceptrons (MLPs) that progressively learn a hierarchy of deep features. For a better modelling of the locality on point clouds, neighborhood information gathering modules are placed before MLPs. A certain point gathers information from its neighbors and propagates its information to them. The neighbors can be predefined (*i.e.* , borrowed from an initial mesh in Point2Mesh [49]) or more commonly established by K-Nearest Neighbor (KNN) search on point clouds (static GCN [121, 83]) or on the feature representation (dynamic GCN [148, 171]).

Yet, this design faces several technical challenges. **Firstly**, the computational cost grows quadratically with the number of points [126, 125]. The problem is exacerbated when KNN search is conducted in a high-dimensional feature space. **Secondly**, the graph feature gathering operation expands the dimension of the resultant features. Consider a point cloud with N points and d coordinates. The dimension of the tensor grows from $N \times d$ to $N \times K \times d$ after the K graph feature gathering operation, where K is the number of neighbors. Then the same operation is applied to the expanded tensor with repeated entries, which leads to redundant computations. **Thirdly**, due to the computational complexity and the expanded features, the GPU memory required for GCN computations explodes when the number of processed points increases. The inference speed also slows down drastically.

As shown in Fig. 6.1, each time the number of processed points doubles, the computational complexity, inference time, and consumed



(a) Overall Accuracy (b) Runtime

(c) GPU memory (d) FLOPs

Figure 6.1: **Acceleration performance of a representative GCN.** The performance of DGCNN and the accelated network is shown. (a) The overall accuracy for point cloud classification. Mean and Variance reported for 5 runs. The (b) runtime, (c) GPU memory consumption, and (d) FLOPs of the original GCN explodes with an increasing number of points. By contrast, the optimized network can achieve a significant reduction of computational resources without a drop in accuracy.

GPU memory of the examined GCN almost quadruple. Thus, the aim of this chapter is to analyze the basic operations in GCNs and seek opportunities to build efficient GCNs for learning on point clouds. Compared with the representative GCN in Fig. 6.1, the computationally optimized GCN in this chapter reduces the computational burden and accelerates the inference. This significant improvement relies on the following two findings.

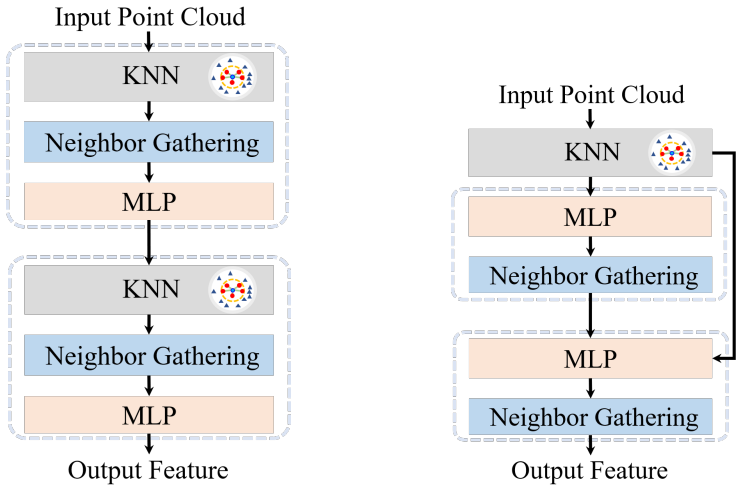
Finding 1. *The local geometric structure information of 3D representations propagates smoothly across the aforementioned multilayer GCN that relies on KNN search for graph feature gathering.*

This finding is supported by the mathematical analysis of the distances between two points before and after one layer of an MLP. In Sec. 6.4.2, we show that the distance between two points after one layer of MLP is upper bounded by the neighborhood distance and lower bounded by the neighborhood centroid distance between the corresponding points before the MLP. This means that across a GCN the distance between two points in the feature space does not abruptly change. Thus, it is not necessary to conduct KNN search every time a neighbor retrieval is needed in MLPs. Instead, a couple of MLPs (referred to as shareholder MLP) can share the results of the same KNN search. Moreover, to ensure a progressively enlarged receptive field across the shareholder MLPs, a larger pool of neighbors can be kept from the first KNN search. Each time neighbor retrieval is needed, the neighbors are sampled from the pool. The shareholder MLPs in the shallower layers can only sample from the near neighbors while the deep shareholders have the chance to sample from far-away neighbors.

Finding 2. *Shuffling the order of the graph feature gathering operation and the MLP used for feature extraction leads to equivalent or similar composite operations for GCNs.*

This finding is also supported by a general analysis in Sec. 6.4.3. As said, in existing GCNs, the graph feature gathering operation happens before the MLP and expand the dimension of the features. By moving the feature extracting MLP before the graph feature gathering operation, the MLP is conducted merely on the non-expanded feature tensors. And this leads to a significant reduction in computations.

The two findings directly lead to the proposed change in computational procedure as shown in Fig. 6.2, which reduces the computational complexity and accelerates the inference of the GCNs. Here, the proposed techniques are applied to four representative GCNs [148, 83, 49, 171]. It is shown that they can improve the efficiency of existing GCNs significantly, indeed. For example, for ModelNet40 point cloud classification with 2048 points, compared with the original DGCNN, the accelerated version is about $\times$**3 times faster**, reduces GPU memory by **57.1%** and computation by **86.7%** without loss of accuracy. More



(a) Conventional GCN (b) Optimized GCN

Figure 6.2: **Optimization of the computation procedure in a conventional GCN.** Instead of calling KNN search for each graph convolution, we enforce several graph convolutions to share the same KNN search with progressively enlarged receptive fields. The shuffling of graph feature gathering and MLP avoids the expansion of features, which leads to accelerated computation in the MLP.

results are shown in Sec. 6.5. Thus, the contributions of this chapter can be summarized as follows.

1. Starting with the analysis of basic operations in representative GCNs, two theorems enabling their acceleration are proved.

2. Based on the proved theorems, two strategies for shuffling operations are proposed to specifically improve the time and memory efficiency of existing GCNs.

3. Extensive experiments on four GCNs for four point cloud learning tasks are carried out, to validate the efficiency of the proposed method. It is demonstrated that both the inference time and memory consumption decreased significantly.

6.2 RELATED WORK

The last years have seen a trend of applying deep neural networks to 3D representations. In this process, computation-efficient network design plays an important role. We briefly summarize closely related contributions.

6.2.1 *Deep Learning for 3D Point Clouds.*

With the easier access to large scale 3D scanned data, convolutional neural networks have been extended from learning 2D features to learning from graph data [160, 35, 77, 46, 18] and 3D point clouds [120, 121, 176, 43]. Existing methods can be roughly categorized into voxel-based, point-based, and voxel-point-mixture methods [134]. Voxel-based methods [109, 118] leverage the architecture of 3D CNNs and apply it to rasterised 3D space. While point-based methods [128, 177, 66, 177] target at an explicit representation and directly operate on graphs. PointNet [120] pioneered the point-based methods by designing a network architecture based on MLP that directly consumes point clouds while respecting the permutation invariance of input data. However, the design of PointNet neglects local structures. Targeting at improving this drawback, PointNet++ [121] introduces a hierachical architecture that recursively calls PointNet on a nested partitioning of input point set. Another way to achieve improvements is via a DGCNN [148], which takes topological information into consideration by defining edge convolution operations.

6.2.2 *Efficient Network Design for 3D data*

Improving computational efficiency enables running well performing neural networks on mobile devices, as well as processing more and more complicated 3D/4D scenes on powerful computers. Targeting at processing 3D/4D scenes with higher performance on the same computational resource, Vote3D [145] and FPNN [84] propose to improve efficiency by dealing with the sparsity problem. Minkowski Engine [21] proposes sparse convolution which uses a hash table for indexing during the convolution process. These methods are designed for improving the efficiency of voxel-based methods. Other efficient

designs for point-based neural networks delve into the basic operations including convolution, pooling, and unpooling [61, 77, 35, 160, 153]. The point sampling based methods like Grid-GCN [153] and RandLA-Net [61] can speed up the network inference, while this direction is parallel to our method.

6.3 NOTATIONS AND PRELIMINARIES

To formally formulate the problem, a couple of concepts are defined in this section. In the following Definition 1 and Definition 2, the neighbors of two points $\mathbf{x}_i$ and $\mathbf{x}_j$ are sorted according to the distance relative to these points, respectively.

Definition 1 (Neighborhood Distance) *Consider two points in a point set $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{S}$. Each of them is equipped with a neighborhood of points derived from KNN search,* i.e. *, $\mathcal{N}_i$, $\mathcal{N}_j$. The neighborhood distance between the two points is the sum of distances between their neighbors,*

$$\mathcal{D}_{\mathcal{N}}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{K} \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2. \tag{6.1}$$

Definition 2 (Neighborhood Centroid Distance) *The neighborhood centroid distance of two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined by the distance between the centroids of their K-nearest neighbors,* i.e. *,*

$$\mathcal{D}_{\mathcal{NC}} = \|\frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_i^k - \frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_j^k\|_2^2, \tag{6.2}$$

where $\frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_i^k$ denotes the centroid of the neighbors.

The Neighborhood Distance indicates the distance between two points. That is, two points with smaller Neighborhood Distance are highly likely to be closer to each other compared to those with larger Neighborhood Distance. Similarly, the Neighborhood Centroid Distance is also a metric that reflects the closeness of two points.

Definition 3 (Graph and Subgraph) *A graph is defined by a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges that defines the connectivity between vertices. A subgraph of a graph $\mathcal{G}$ is defined by the pair $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, where $\mathcal{V}_i \subseteq \mathcal{V}$, and $\mathcal{E}_i \subseteq \mathcal{E}$.* A graph $\mathcal{G}$ can be defined on point clouds and meshes. A subgraph $\mathcal{G}_i$ captures the local connectivity on the 3D representation and is constructed slightly differently for point

clouds and meshes. For a point cloud, the vertices of the subgraph include a point and its K-nearest neighbors and the edge connects the center point and the neighbors. For a mesh, the edge set $\mathcal{E}_i$ contains an edge and its 4 1-ring neighbors [49], and the vertex set $\mathcal{V}_i$ contains the 4 associated vertices.

Definition 4 (Graph Convolution) *Graph convolution is a family of operations that extract higher-level features from the lower-level ones by propagating information between vertices $\mathcal{V}$ or edges $\mathcal{E}$ of the defined graph $\mathcal{G}$. Graph convolution can be defined in terms of the subgraphs,* i.e. *,*

$$\mathbf{x'}_i = g(\mathcal{G}_i; \mathbf{\Theta}) = \sum_{\mathbf{e}_i^k \in \mathcal{E}_i} h(\mathbf{x}_i^k; \mathbf{\Theta}_k), \tag{6.3}$$

where $\mathbf{x}_i^k \in \mathbb{R}^d$ is a vector that encodes the feature on the edge $\mathbf{e}_i^k \in \mathcal{E}_i$, $\mathbf{x'}_i \in \mathbb{R}^M$ is the output feature, $\mathbf{\Theta} = \{\mathbf{\Theta}_k \in \mathbb{R}^{d \times M} | k = 1, 2, \cdots, K\}$ is the ensemble of the trainable parameters and is the same for all of the subgraphs $\mathcal{G}_i$.

The function $h(\cdot)$ transforms a d-dimensional input feature into a M-dimensional vector. It denotes an MLP, which in turn can be implemented as convolution operation. In this specific case, the aggregation function is a summation denoted by $\sum$. Generally, the aggregation function is a symmetric function (*e.g.* , $\sum$ or max) that does not depend on the order of the edges. A stack of graph convolutions and other operations such as pooling constitute a GCN.

Notation. In this chapter, N represents the number of points, d represents the dimensionality of the latent space features, K represents the number of neighbors for each point, and M represents the dimensionality of the intermediate output features.

6.4 METHODOLOGY

In this section, the basic operations in GCNs, *i.e.* , KNN search and MLP in graph convolution, are analyzed. Two theorems about the properties of the two operations are proposed. Building on those, a simplified computational procedure for KNN search and MLP is introduced, which improves the computational efficiency of existing GCNs.

6.4.1 *Computational complexity analysis in GCN*

In state-of-the-art GCNs, KNN search is usually conducted to define the neighborhood, followed by an MLP. The computational complexity of the two operations are analyzed and the simplification methods are presented.

Proposition 1 *The ratio of the computational complexities of KNN search and MLP in a graph convolution is* $\gamma = \frac{N}{KM}$.

Assume that the point cloud is represented by a $N \times d$ matrix $\mathbf{X}$. To compute the K-nearest neighbors of all the points, a pairwise comparison between the points is conducted, *i.e.* , $\mathbf{D} = \mathbf{X}\mathbf{X}^T$. Then for each point, the indices of the K-nearest neighbors are kept and used to extract the graph feature, which results in a 3D tensor with a dimension of $N \times K \times d$. Then an MLP implemented as convolution with kernel size 1×1 and output channel M is conducted on the graph feature. The pairwise comparison and the 1×1 convolution are the computation-intensive parts.

The computational complexity, namely the number of multiplications in the pairwise comparisons is $\mathcal{C}_{nn} = dN^2$. And the computational complexity of the 1×1 convolution is $\mathcal{C}_{conv} = dMKN$. Thus, the ratio between the two complexity terms is

$$\gamma = \frac{\mathcal{C}_{nn}}{\mathcal{C}_{conv}} = \frac{dN^2}{dMKN} = \frac{N}{KM} \tag{6.4}$$

Compared with the number of nearest neighbors K and the output channel dimensionality M, the number of points in a point cloud could vary drastically. When N is small, the pairwise computation load is relatively small and even negligible. But when the point cloud grows huge, the computational load of this pairwise comparison could become dominant. This analysis shows the necessity of simplifying KNN search in GCNs.

6.4.2 *Propagation of point adjacency*

In the following, we investigate how local geometric structure information propagates within the GCN, by analyzing the adjacency property of points before and after graph convolution. This new perspective motivates us to rethink the necessity of frequent KNN callings in GCNs, as already hinted at earlier. It results in a simplification and acceleration

of the adjacency assessment in GCNs. We consider a special case of the graph convolution in Eqn. (6.3) in the following form

$$\mathbf{x}'_i = [\mathbf{x}'_{i1}, \cdots, \mathbf{x}'_{im}, \cdots, \mathbf{x}'_{iM}], \tag{6.5}$$

$$\mathbf{x}'_{im} = \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k >, \tag{6.6}$$

where $\mathbf{x}'_{im}$ denotes the m-th element of the vector $\mathbf{x}'_i$, $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_M\}$ contains the trainable parameters of the MLP with M output channels. For the operation defined above, the following theorem is derived.

Theorem 1 *Given that parameters $\boldsymbol{\theta}_m$ in the network follow an independent Gaussian distribution with o mean and σ^2 variance, the distance of two points in the input space is upper bounded by the neighborhood distance of the corresponding points in the output space up to a scaling factor, and lower bounded by the neighborhood centroid distance of the same points up to a scaling factor,* i.e. ,

$$\begin{aligned} &\sigma^2 K^2 \| \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_i^k - \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_j^k \|_2^2 \\ &\leq \mathbb{E}[\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2] \leq \sigma^2 dKM \sum_{k=1}^{K} \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2. \end{aligned} \tag{6.7}$$

The condition in the theorem is reasonable since the parameters in neural networks are not only often initialized with independent Gaussian distributions. Actually, as shown in Fig. 6.3, also after training, the parameters tend to follow Gaussian-like empirical distributions.

Proof. **Upper bound.** For the simplicity of analysis, inner product and summation are selected as the edge function and the aggregation operation in **Theorem 1**. Thus, the theorem is derived under the assumption that the graph convolution has the following form

$$\mathbf{x}'_i = [\mathbf{x}'_{i1}, \cdots, \mathbf{x}'_{im}, \cdots, \mathbf{x}'_{iM}], \tag{6.8}$$

$$\mathbf{x}'_{im} = \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k >, \tag{6.9}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_M\}$ is the trainable parameters of the MLP with M output channels.

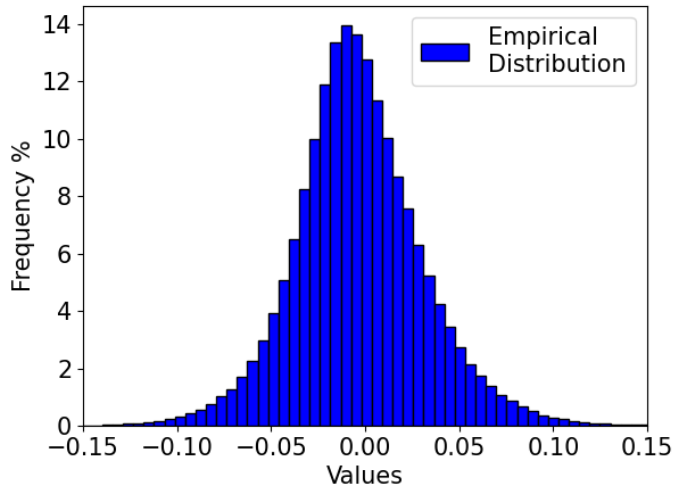


Figure 6.3: **Empirical weight distribution** of a layer in a fully trained dynamic GCN for point cloud classification. The distribution is Gaussian-like.

Then the squared distance between two points $\mathbf{x}'_i$ and $\mathbf{x}'_j$ after the graph convolution is

$$\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2 = \sum_{m=1}^{M} (\mathbf{x}'_{im} - \mathbf{x}'_{jm})^2 \tag{6.10}$$

$$= \sum_{m=1}^{M} (\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k > - \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_j^k >)^2 \tag{6.11}$$

$$= \sum_{m=1}^{M} (\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k >)^2 \tag{6.12}$$

$$\leq \sum_{m=1}^{M} K \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k) >^2 \tag{6.13}$$

$$\leq K \sum_{m=1}^{M} \sum_{k=1}^{K} \|\boldsymbol{\theta}_m\|_2^2 \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2. \tag{6.14}$$

The inequality in Eqn. (6.13) follows that the arithmetic mean is not larger than the quadratic mean while the inequality in Eqn. (6.14) follows Cauchy–Schwarz inequality. Assume that the parameters $\boldsymbol{\theta}_m$ in the network are random variables that follows Gaussian distribution

with 0 mean and σ^2 variance. Then the distance $\|\mathbf{x}'_i - \mathbf{x}'_k\|_2^2$ is also a random variable and the expectation is expressed as,

$$\mathbb{E}[\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2] \leq \mathbb{E}[K \sum_{m=1}^{M} \sum_{k=1}^{K} \|\boldsymbol{\theta}_m\|_2^2 \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2] \tag{6.15}$$

$$= K \sum_{m=1}^{M} \sum_{k=1}^{K} \mathbb{E}[\|\boldsymbol{\theta}_m\|_2^2] \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2 \tag{6.16}$$

$$= \sigma^2 dKM \sum_{k=1}^{K} \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2, \tag{6.17}$$

where the term $\sum_k \|\mathbf{x}_i^k - \mathbf{x}_j^k\|_2^2$ is just the neighborhood distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. □

Proof. **Lower bound.** In Eqn. (6.12), let

$$\mathbf{a}_m = \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k >. \tag{6.18}$$

Thus, Eqn. (6.12) become

$$\sum_{m=1}^{M} (\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k >)^2 = \sum_{m=1}^{M} \mathbf{a}_m^2. \tag{6.19}$$

Using Cauchy-Schwarz inequality

$$\sum_{m=1}^{M} \mathbf{a}_m \mathbf{b}_m \leq \sqrt{\sum_{m=1}^{M} \mathbf{a}_m^2} \sqrt{\sum_{m=1}^{M} \mathbf{b}_m^2} \tag{6.20}$$

and letting $\mathbf{b}_m^2 = 1/M$, then the inequality in Eqn (6.20) becomes

$$\left(\frac{1}{\sqrt{M}} \sum_{m=1}^{M} \mathbf{a}_m \right)^2 \leq \sum_{m=1}^{M} \mathbf{a}_m^2. \tag{6.21}$$

Thus, the lower bound of Eqn (6.12) becomes

$$\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2 = \sum_{m=1}^{M} (\sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k >)^2 \tag{6.22}$$

$$\geq \frac{1}{M} \left(\sum_{m=1}^{M} \sum_{k=1}^{K} < \boldsymbol{\theta}_m, \mathbf{x}_i^k - \mathbf{x}_j^k > \right)^2 \tag{6.23}$$

$$= \frac{1}{M} < \sum_{m=1}^{M} \boldsymbol{\theta}_m, \sum_{k=1}^{K} \mathbf{x}_i^k - \mathbf{x}_j^k >^2 . \tag{6.24}$$

Let $\phi = \sum_{m=1}^{M} \boldsymbol{\theta}_m$ and $\mathbf{z} = \sum_{k=1}^{K} \mathbf{x}_i^k - \mathbf{x}_j^k$. Then

$$\|\mathbf{x}_i' - \mathbf{x}_j'\|_2^2 \geq \frac{1}{M} < \boldsymbol{\phi}, \mathbf{z} >^2 \tag{6.25}$$

$$= \frac{1}{M} (\sum_{l=1}^{d} \boldsymbol{\phi}_l, \mathbf{z}_l)^2 \tag{6.26}$$

$$= \frac{1}{M} \sum_{l=1}^{d} \sum_{n=1}^{d} \boldsymbol{\phi}_l \boldsymbol{\phi}_n \mathbf{z}_l \mathbf{z}_n. \tag{6.27}$$

Then taking the expectation on both sides, the inequality becomes

$$\mathbb{E}[\|\mathbf{x}_i' - \mathbf{x}_j'\|_2^2] \geq \mathbb{E}[\frac{1}{M} \sum_{l=1}^{d} \sum_{n=1}^{d} \boldsymbol{\phi}_l \boldsymbol{\phi}_n \mathbf{z}_l \mathbf{z}_n] \tag{6.28}$$

$$= \frac{1}{M} \sum_{l=1}^{d} \sum_{n=1}^{d} \mathbb{E}[\boldsymbol{\phi}_l \boldsymbol{\phi}_n] \mathbf{z}_l \mathbf{z}_n. \tag{6.29}$$

Note that the elements of $\boldsymbol{\theta}_m$ follow independent Gaussian distribution with o mean and σ^2 variance and $\phi = \sum_{m=1}^{M} \boldsymbol{\theta}_m$. Thus, the elements of ϕ follows independent Gaussian distribution with o mean and $M\sigma^2$ variance. Thus,

$$\mathbb{E}[\boldsymbol{\phi}_l \boldsymbol{\phi}_n] = \begin{cases} 0 & l \neq n \\ M\sigma^2 & l = n \end{cases}. \tag{6.30}$$

Thus, the lower bound becomes

$$\mathbb{E}[\|\mathbf{x}_i' - \mathbf{x}_j'\|_2^2] \geq \frac{1}{M} \sum_{l=1}^{d} M\sigma^2 \mathbf{z}_l^2 \tag{6.31}$$

$$= \sigma^2 \|\mathbf{z}\|_2^2 \tag{6.32}$$

$$= \sigma^2 \| \sum_{k=1}^{K} \mathbf{x}_i^k - \mathbf{x}_j^k \|_2^2 \tag{6.33}$$

$$= \sigma^2 K^2 \| \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_i^k - \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_j^k \|_2^2. \tag{6.34}$$

Thus, the distance between two points after graph convolution is lower bounded by the neighborhood centroid distance of the corresponding points before graph convolution up to a scaling factor.

□

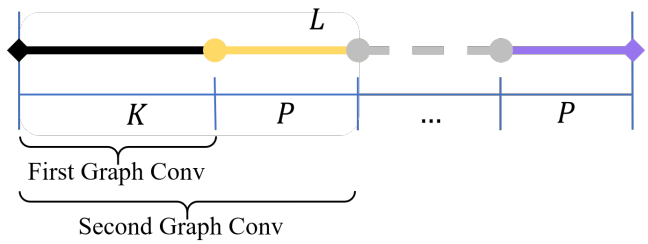


Figure 6.4: **Neighbor sampling** for MLPs sharing the same KNN search.

Since both the neighborhood distance and the neighborhood centroid distance are an indicator of the closeness of two points, Theorem 1 indicates that the adjacency property of points propagates smoothly across the stack of multilayer graph convolutions. This conclusion motivates us to rethink the frequently occurring adjacency assessment of points via KNN in a multilayer GCN. One straightforward bypass is to reduce the number of KNN searches and let several graph convolution modules share the results from one KNN search as shown. Yet, this simple scheme could probably reduce the receptive field for the stack of several graph convolutions.

Thus, we propose to progressively enlarge the receptive field of the graph convolutions that share the same KNN search as shown in Fig 6.4. As shown in Fig. 6.2b, for a stack of n graph convolutions, only one full-fledged KNN search is conducted, which leads to an enlarged pool of $L = K + (n-1)P$ neighbors more than originally needed. Then for the first convolution, the first K-nearest neighbors from the pool are still selected. For the l-th convolutions, the neighbors of a point are identified by randomly sampling the first $K + (l-1)P$ elements of the pool. That is, for each additional graph convolution, the sampling pool is enlarged by a step P. Since the bound in Theorem 1 is not strict, the adjacency between point could still change across the deep layers. This allows the GCN to still be able to capture long-range dependencies between points after the computational simplification (See Fig. 6.7). The above operations reduce multiple KNN searches to one. Across the whole network, KNN search is conducted every several layers. This simplification can accelerate the inference of the network.

6.4.3 *Graph convolution with graph feature gathering*

The graph convolution in Eqn. (6.3) is applied to the subgraph defined by local point proximity. Subgraph features are gathered from the neighbors of a point and brought to the center of the local coordinate system. Then an MLP is applied to the centered features. In summary, the convolution in Eqn. (6.3) is of the following form

$$g(\mathcal{G}_i; \mathbf{\Theta}) = \sum_k f(\mathbf{x}_k - \mathbf{x}_i, \mathbf{\Theta}_k). \tag{6.35}$$

This form of operation is used in a couple of dynamic [148, 171] and non-dynamic [121, 83, 49] GCNs. To conduct the operation, features are first gathered from the neighborhood, *i.e.* , $\mathbf{x}_k - \mathbf{x}_i$, which forms a tensor with dimension $N \times K \times d$. Then the gathered feature is convolved with the MLP. The computational complexity of the convolution operation is $dKMN$. To save computations, we propose to shuffle the order of graph feature gathering and MLP. That is, the computation is conducted as $\sum_k f(\mathbf{x}_k) - f(\mathbf{x}_i)$. In this way, the MLP is first applied to the individual points, after which feature gathering is applied.

To explain the rationale of this shuffling operation, we consider a case widely used in GCNs [148], *i.e.*

$$g(\mathcal{G}_i) = \max_k \{(< [\boldsymbol{\theta}_m, \boldsymbol{\phi}_m], [\mathbf{x}_k - \mathbf{x}_i, \mathbf{x}_i] >)\}, \tag{6.36}$$

where max is used as the aggregation function and the operator $[\cdot]$ concatenates two vectors. It is claimed that the term $\mathbf{x}_k - \mathbf{x}_i$ captures the local information while $\mathbf{x}_i$ keeps the global information. The following theorem states that an equivalent but computationally efficient procedure exists for the special case in Eqn. (6.36).

Theorem 2 *For the graph convolution defined by Eqn. (6.36), shuffling the order of neighbor feature gathering and the MLP leads to an* ***equivalent*** *operation for the GCN.*

Proof. Eqn. (6.36) could be written as

$$\mathbf{x}'_{im} = \max_k \{(< \boldsymbol{\theta}_m, \mathbf{x}_k > + < \boldsymbol{\psi}_m, \mathbf{x}_i >)\}, \tag{6.37}$$

where $\boldsymbol{\psi}_m = \boldsymbol{\phi}_m - \boldsymbol{\theta}_m$. In Eqn. (6.37), the operations are rearranged such that the convolutions w.r.t. the points and their neighbors are perfectly separated. Considering that a point could act as a neighbor of

the other point multiple times and the same parameters $\boldsymbol{\theta}_m$ are used for convolution, the convolution operation in Eqn. (6.37) has an equivalent procedure: 1) applying two different MLPs to the original points, 2) gathering the neighbor features, and 3) summing up the features. The computational complexity is reduced to $2dMN$, which is only $1/K$ of the original. □

Thus, inspired by the equivalent operation for Eqn. (6.37) obtained by shuffling the order of graph feature gathering and MLP, we propose to use the same shuffling procedure for the general case in Eqn. (6.36). The effectiveness of the shuffling operation is validated in the experiments.

6.5 EXPERIMENTS

Dataset. This section validates the effectiveness of the proposed GCN acceleration method on the four popular network architectures DGCNN [148], PointCNN [83], Point2Mesh [49], and LDGCNN [171]. Experiments on four important tasks are included, *i.e.* , point cloud classification, part segmentation, semantic segmentation, and surface reconstruction.

For classification task, the performance is evaluated on the public benchmark the ModelNet40 dataset [151]. ModelNet40 consists of 12,311 meshed CAD models of 40 categories. We follow the experimental setting of PointNet [120, 121] and DGCNN [148]. Inference runtime is measured on a single Titan Xp GPU and the batch size is reduced to 16 for running with 2048 points. For part segmentation task, the ShapeNetPart [15] dataset is used. In ShapeNetPart, there are 16 object categories and 16,881 3D shapes, annotated with 50 parts. 2048 points are sampled from each shape. For semantic segmentation task, Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [5] is used. S3DIS consists of indoor scenes of 272 rooms in six indoor areas, annotated with 13 semantic categories. The experiments follow the standard training, validation, and test split of DGCNN. Part segmentation experiments are run on two Titan Xp GPUs and the batch size is 32. For surface reconstruction, we use the dataset released by [49] and some public 3D models. The visualization of meshes is done on the platform Open3D [180].

Experiment Setup. All of the experiments are rerun for the original and the accelerated networks. The same training protocol is used for fair comparison. For point cloud classification, semantic segmentation,

Network	Method	Points	K	OV Acc.	BL Acc.	Time [ms]	Mem. [GB]	FLOPs [G]	#GPU
PointNet [120]		1024	20	89.4	83.7	4.7	0.5	–	1
PointNet++ [121]		1024	20	90.7	–	113	–	–	1
KPConv [142]		1024	20	92.9	–	108	3.2	–	1
PointCNN [83]	Baseline	1024	20	91.86	87.93	35.3 / 100.%	0.8 / 100.%	2.5 / 100.%	1
	Accel.	1024	20	91.86	87.92	29.1 / 82.4 %	0.6 / 76.7 %	1.9 / 76.2 %	1
DGCNN [148]	Baseline	1024	20	92.10	89.05	74.7 / 100.%	5.2 / 100.%	86.0 / 100.%	1
	Accel.	1024	20	92.56	89.62	38.1 / 51.0%	2.5 / 48.1%	20.4 / 23.7%	1
LDGCNN [171]	Baseline	1024	20	92.54	89.57	95.4 / 100.%	4.9 / 100.%	74.2 / 100.%	1
	Accel.	1024	20	92.50	89.38	24.1 / 25.3%	1.3 / 26.5%	13.5 / 18.3%	1
DGCNN [148]	Baseline	2048	40	92.56	89.90	385.8 / 100.%	20.5 / 100.%	309.2 / 100.%	3
	Accel. S1	2048	40	92.58	89.60	212.7 / 55.1%	20.5 / 100.%	274.8 / 88.9%	3
	Accel. S2	2048	40	92.63	90.16	164.7 / 42.7%	8.7 / 42.4%	75.4 / 24.4%	1
	Accel.	2048	40	92.63	89.82	132.0 / 34.2%	8.8 / 42.9%	41.0 / 13.3%	1

Table 6.1: **Quantitative comparison for point cloud classification on ModelNet40.** All experiments are rerun and the accuracy results are averaged over 5 runs. OV Acc. and BL Acc. denote overall and balanced accuracy, resp.

and part segmentation, the accuracy results are averaged over 5 runs, thus increasing the reliability of the reported numbers. The aim of our experiment is to compare the accuracy, test time, maximum GPU memory of the proposed method with original networks. The training of our accelerated models are all done on a single TITAN XP GPU whereas the original networks require more than one GPU for some of the experiments. Due to the different hardware environments, the runtime might be different from the original papers.

Hyperparameter Setup. Several hyperparameters are involved. We follow the default settings to determine the number of neighbors K in KNN. For classification with 1024 and 2048 points, K is 20 and 40, resp. For part segmentation and semantic segmentation, K is set to 40 and 20, resp. The enlargement step P is chosen empirically for different tasks. We try $P = 1/4K, 1/2K, 3/4K, K$.

Model	Method	Classification	Part segmentation
		Conv / KNN/ Total	Conv / KNN / Total
PointCNN	Baseline	2.4G / 0.118G / 2.5G	8.5G / 1.175G / 9.7G
	Accel.	1.8G / 0.104G / 1.9G	7.0G / 0.588G / 7.6G
DGCNN	Baseline	77.3G / 8.7G / 86.0G	140.8G / 18.0G / 158.8G
	Accel.	20.3G / 0.1G / 20.4G	53.6G / 17.6G / 71.2G
LDGCNN	Baseline	60.9G / 13.3G / 74.2G	149.7G / 53.2G / 202.9G
	Accel.	13.4G / 0.10G / 13.5G	52.3G / 0.4G / 52.7G

Table 6.2: **Breakdown analysis of FLOPs of different networks.**

6.5.1 *Point Cloud Classification.*

Acceleration results. The comparison between the original networks [148, 83, 171] and the accelerated versions for point cloud classification is shown in Table 6.1. With 1024 points available, compared with DGCNN, the accelerated network is about twice faster, reduces the GPU memory and computation by 49% and 76.6% with similar accuracy. On the heavier network LDGCNN [171], the accelerated version is about $\times 4$ times faster. Even for the compact PointCNN, the proposed method could reduce the runtime by 17.6%. When 2048 points are available, the accelerated version is about $\times 3$ faster, reduces GPU memory by 57.1% and computation by 86.7% without loss of accuracy. DGCNN needs

three Titan Xp GPUs for the test with 2048 points, while our method only needs one GPU.

Breakdown analysis. In addition to the overall acceleration performance, the detailed breakdown analysis is given as follows. **I.** Besides our full method, performances with only KNN simplification (Accel. S1) or operation shuffling in Sec. 6.4.3 (Accel. S2) are also ablated. As shown in the table, both of the strategies could improve the efficiency of the network. **II.** The effects of the proposed two methods depend on the redundancy of the baseline networks. For example, PointCNN exploits point subsampling to make the computation highly efficient while DGCNN and LDGCNN keep a high density of points across the network. Applying the same method to a more redundant network could bring more benefits. **III.** The reduction of inference time comes from the reduction of computation. The breakdown analysis of the computation reduction of the two proposed methods is given in Table 6.2. The effect of the two proposed methods on computation reduction depends on the networks and tasks. For LDGCNN, both KNN simplication and operation shuffling contribute a lot to the reduction of computation. For PointCNN on classification and DGCNN on part segmentation, operation shuffling leads to more computation reduction compared with KNN simplification. **IV.** The reduction of GPU memory mainly comes from shuffling of operations because it avoids the expansion of point feature.

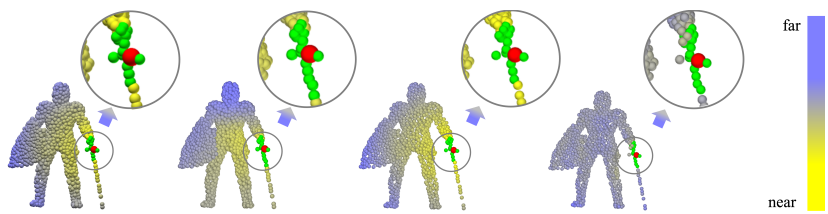


Figure 6.5: **Qualitative result on Modelnet40.** (a), (b) Input space and last-layer feature space rendered as colormap between the red point and the rest points at Epoch 0. The green points are KNN of the red point. (c), (d) follow the same layout with (a), (b) at Epoch 250.

Visualization. In order to validate that the neighborhood geometric features are preserved after all the operations and acceleration strategies,

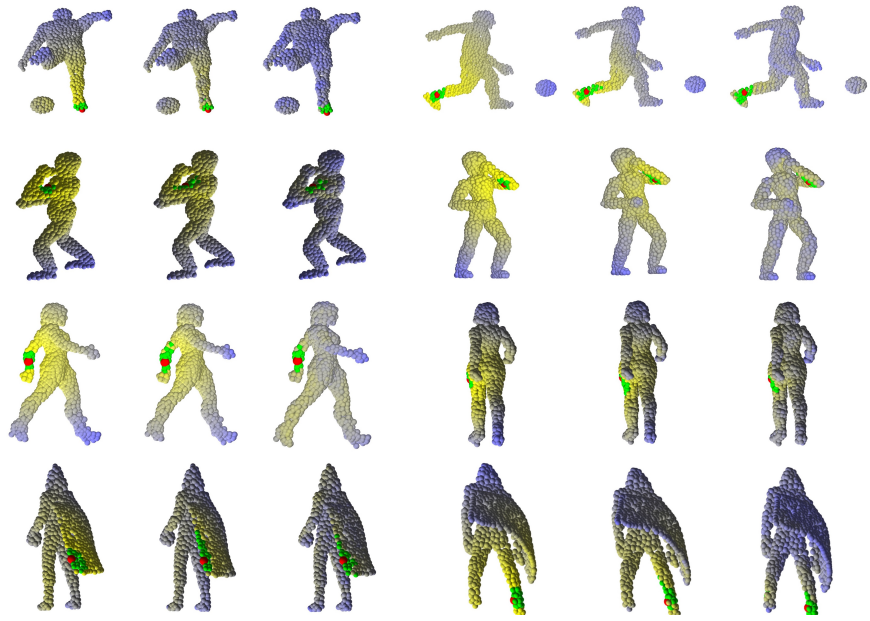


Figure 6.6: **Renderings of input space and feature space** as colormap between the red point and the rest of the points on ModelNet40 dataset. The green points represent KNN of the red point. (a) represents the input space. (b) represents the feature space extracted from the second layer of the network. (c) represents the feature space extracted from the last layer of the network. (d), (e),and (f) follows the same layout with (a), (b), and (c), respectively.

experiments are designed by extracting and visualizing the feature map of the accelerated networks as a distance color map rendered on the 3D point cloud. The visualized feature maps are shown from two perspective including feature map evolution during training and feature map enriching across the network. **I.** The evolution of feature space *w.r.t.* the number of epochs is shown in Fig. 6.5. By comparing Fig. 6.5b and Fig. 6.5d, we can see that there is a clear contraction of near neighbors during the training. At Epoch 250 when the loss of the classification neural network converges, the yellowish neighbor features also converge into a very small region which is smaller than

the KNN represented by the green points. Thus, the local structures of the point clouds are well preserved by the network during training despite the simplification of the operations. **II.** Fig. 6.6 shows more results of feature space for point cloud classification on ModelNet40. As the network goes deeper, the yellow points gradually contract to a small region near the red point. This proves the effectiveness of the local feature extraction is kept by the accelerated network.

Method	mIoU	Runtime [ms]	GPU mem. [GB]	FLOPs [G]	#GPU
PointCNN [83]					
Baseline	83.34	123.0 / 100.%	3.3 / 100.%	9.7 / 100.%	1
Accel.	83.21	111.9 / 91.0%	2.7 / 82.7%	7.6 / 78.8%	1
DGCNN [148]					
Baseline	84.95	116.1 / 100.%	17.2 / 100.%	158.8 / 100.%	2
Accel.	84.78	81.8 / 70.5%	4.1 / 23.8%	71.2 / 44.8%	1
LDGCNN [171]					
Baseline	84.13	365.3 / 100.%	9.7/ 100.%	202.9 / 100.%	2
Accel.	84.02	46.2 / 12.6 %	1.9 / 19.5%	52.7 / 26.0%	1

Table 6.3: **Quantitative comparison for part segmentation of point clouds on ShapeNetPart**. Experiments are rerun. Accuracy is averaged over 5 runs.

Method	mIoU	Runtime	GPU mem.	#GPU
Baseline	57.5	172.7 / 100.%	14.6 / 100.%	2
Accel.	57.0	87.0 / 50.4%	6.0 / 40.8%	1

Table 6.4: **Comparison for semantic segmentation of point clouds on S3DIS.** Accuracy reported over 5 runs. The unit of the metrics is the same as that in Table 6.1.

6.5.2 *Point Cloud Segmentation.*

The experimental results for point cloud part segmentation are shown in Table 6.3. The mean IoU metric is used to quantitatively evaluate

the segmentation performance. As shown in Table 6.3, compared with DGCNN, our method greatly reduces the runtime, GPU memory consumption, and computation by 29.5%, 76.2%, and 56.2%, resp. The networks in [83] and [171] are accelerated by 9% and 77.4%, resp. In Fig. 6.7, the distance of points in the input space and the feature space is shown. As the network gets deeper, the accelerated network could still learn the long-range dependencies between points. As for the semantic segmentation task, Table 6.4 shows that our method reduces the runtime and memory consumption by 49.6% and 59.2% compared with DGCNN.

Method	**F-score**		**Runtime [s]**	**#Param. [k]**
	Bunny	**Bird**		
Baseline	69.7	53.3	0.41 / 100.%	735.8 / 100.%
Accel.	73.0	51.6	0.29 / 70.7%	153.7 / 20.9%

Table 6.5: **Quantitative comparison for surface reconstruction.**

6.5.3 *Surface Reconstruction.*

In order to validate the efficiency of our method applied to prior networks, we compare it with Point2Mesh [49] for the surface reconstruction. Similar to Point2Mesh, we use the F-score as the metric to evaluate the quality of the reconstructed meshes. The result is shown in Table 6.5. It can be observed that our method has similar reconstruction quality as Point2Mesh, while speeding up inference by 29%. Note that the number of parameters is reduced by 79.1%. The qualitative results for different shapes are shown in Fig. 6.8. It is obvious that our accelerated method recovers the 3D meshes with a similar quality as Point2Mesh.

6.5.4 *Applicability.*

The two methods proposed in this chapter have different application scenarios. **I.** In the first method, we handle the simplification of neighbor querying. We focus on GCNs with KNN because they generally performs better and KNN is the most expensive method among the

Figure 6.7: **Visualization of the point distance** across the accelerated network for part segmentation task. The distance of points to the red point in the figures is computed. Lighter color means closer distance. (a) The input shape. (b) Distance between points in the raw data. (c)-(e) Distance between points in the feature space from Layer 1, Layer 2, Layer 3 of the accelerated network. (f) Segmentation result. The accelerated network could still capture long-range dependency between points.

Figure 6.8: **Surface reconstruction results.** (a) Input point cloud. (b, c) Surface and normal map reconstructed by Point2Mesh. (d, e) Surface and normal map reconstructed by our method.

available methods. In the analysis of Theorem 1, the core assumption is the ordered neighbors. It does not matter whether the neighbors come from KNN search or other alternatives such as ball querying. Thus, the same theoretical analysis and conclusion hold for those methods. **II.** The second method is not limited to a specific GCN. Instead, it is applicable to any GCN with the computation pattern that feature gathering occurs before MLP (*e.g.* Point2Mesh). Actually, the second method is adopted for all the four investigated networks. In Table 6.2, the reduction of the FLOPs of convolution reflects how the shuffling trick works for different networks. **III.** Some classical GCNs do not use KNN. Yet, they still need to propagate message between neighbors. In this case (GCN2Conv, RGCNConv, Point2Mesh), the adjacency is usually defined on the input data. In some cases (MeshConv, Point2Mesh), feature gathering occurs before MLP and our shuffling method could be used.

6.6 CONCLUSION

In this work we have presented two strategies for improving the time and memory efficiency of dynamic GCNs. The two strategies are based on the analysis of basic operations in GCNs. The modified networks retain their accuracy while significantly shrinking the test time and GPU memory consumption. Experimental results show that our method has a significant performance on multiple important tasks. In the future, we plan to explore how to add flexibility and efficiency to the design of neural network for 3D tasks.

CONCLUSION

7 CONCLUSION AND OUTLOOK

7.1 CONTRIBUTION

In this thesis, we tried to improve the computational procedure of DNNs from three perspectives including neural network compression, neural architecture optimization, and computational procedure optimization.

In this first part of the thesis, two network compression approaches including filter decomposition and filter pruning are investigated and three new network compression methods are proposed. **First**, a filter basis learning method is proposed to reduce the number of parameters of CNNs in Chapter 2. The contribution of this method is that it can balance the distribution of parameters between the basis filters and the combination coefficients. **Second**, a differentiable pruning method is proposed for automatic network pruning in Chapter 3. The major contribution of this method is the designed new family of hypernetworks used to reparameterize the network and the differentiablity of the pruning method. **Third**, group sparsity is used for network compression in Chapter 4. The contribution of this chapter is the unified analysis of filter decomposition and filter pruning is provided from the perspective compact tensor approximation.

The proposed three methods have different effects on network compression. The filter basis learning method is mainly designed to reduce the number of parameters in the network. The differentiable network pruning method can reduce a significant amount of computation in the network and speed up the inference of the compressed network. And the third method provides an opportunity to flexibly select from filter decomposition and filter pruning for network compression

Then in the second part of the thesis, we focus on neural architecture optimization. Specifically, a cost-free fine-grained architecture optimization method is developed in Chapter 5. The architecture of a given network is optimized by searching in a larger network configuration space. The single-shot shrinkage method is used as the agent to identify a better network architecture. Different from NAS, the architecture optimization procedure of the proposed method is highly efficient.

In the third part of the thesis, an acceleration method is designed to improve the computational efficiency of GCNs for learning on point clouds. Two techniques are proposed to achieve that goal. First, by mathematically analyzing the propagation of local geometric structure information, it is found adjacency property of points propagates smoothly across the network. Thus, this motivates the simplification of KNN search in GCNs. Second, it is found that shuffling the order of graph feature gathering and an MLP leads to equivalent or similar composite operations. The shuffling of operation also avoids the redundant computation on repeated features. In short, by utilizing KNN simplification and operation shuffling, the computational efficiency of GCNs could be greatly improved.

7.2 CHALLENGES

Despite the great advances in efficient computation, there are still a lot of challenges in this field.

Network Compression. For network compression especially network pruning, there are already many published works. Yet, due to the different optimization method, the fine-tuning method after compression, the evaluation metrics, the applied networks, tasks, and datasets, it is hard to have a fair comparison between different methods. Thus, there is a strong need for a benchmark standard that helps to form fair comparisons between different methods. Second, for current network compression methods, the compressed network are usually fine-tuned for quite a long time. In this case, the final parameters in the compressed network is less related to the parameters in the original network. And this raises the question whether the pretrained network parameters are important. Actually, there are already a couple of works which states that the network architecture is more important than the pretrained parameters. Thus, the network compression problem becomes the problem of finding the optimal fine-grained architecture. Thus, more efforts should be directed to improving the search efficiency of network compression algorithms.

Architecture Optimization. The general pipeline of neural architecture optimization includes building a supernet, searching sub-networks in the supernet, and retraining the found architecture. The major problem of neural architecture optimization is the huge computation in the search phase. Although some methods such as weight sharing are pro-

posed, the computation cost is still unaffordable in the case of limited computational resources. In addition, after the performance boost in the early phase of the development, NAS seems to meet a saturation point. Recently, transformer architectures have been successfully applied to a lot of computer vision tasks. In this new era, whether NAS could still boost the performance of transformers needs to be answered.

Computational procedure optimization. The methods used to optimize the computational procedure highly depends on the basic operation and architecture in the network. Besides CNNs and GCNs, transformers have also thrived during the past two years. And transformers are well-known for the large-scale computation. Past efforts are directed to improving efficiency of the self-attention in transformers. Yet, it is also interesting how the other techniques such as network compression and architecture optimization could be applied to transformers.

BIBLIOGRAPHY

[1] Martıén Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. „Tensorflow: A system for large-scale machine learning.“ In: *Proceedings of {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283 (cit. on p. 23).

[2] Eirikur Agustsson and Radu Timofte. „NTIRE 2017 challenge on single image super-resolution: Dataset and study.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 126–135 (cit. on pp. 26, 55, 104).

[3] Jose M Alvarez and Mathieu Salzmann. „Compression-aware training of deep networks.“ In: *Advances in Neural Information Processing Systems*. 2017, pp. 856–867 (cit. on p. 64).

[4] Jose M Alvarez and Mathieu Salzmann. „Learning the number of neurons in deep networks.“ In: *Advances in Neural Information Processing Systems*. 2016, pp. 2270–2278 (cit. on pp. 16, 64).

[5] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. „3D semantic parsing of large-scale indoor spaces.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1534–1543 (cit. on p. 123).

[6] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. „N2N learning: Network to network compression via policy gradient reinforcement learning.“ In: *arXiv preprint arXiv:1709.06030* (2017) (cit. on p. 18).

[7] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. „SURF: Speeded up robust features.“ In: *Proceeding of the European Conference on Computer Vision*. Springer. 2006, pp. 404–417 (cit. on p. 3).

[8] Amir Beck. *First-order methods in optimization*. Vol. 25. SIAM, 2017 (cit. on p. 73).

[9] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. „Low-complexity single-image super-resolution based on nonnegative neighbor embedding." In: *Proceedings of the British Machine Vision Conference*. 2012 (cit. on pp. 26, 57, 104).

[10] Goutam Bhat, Martin Danelljan, Luc Van Gool, and Radu Timofte. „Learning discriminative model prediction for tracking." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 6182–6191 (cit. on pp. 87, 103).

[11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. „Distributed optimization and statistical learning via the alternating direction method of multipliers." In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122 (cit. on p. 69).

[12] Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. „SMASH: One-Shot Model Architecture Search through HyperNetworks." In: *Proceedings of International Conference on Learning Representations*. 2018 (cit. on p. 41).

[13] Jiezhang Cao, Yawei Li, Kai Zhang, and Luc Van Gool. „Video Super-Resolution Transformer." In: *arXiv preprint arXiv:2106.06847* (2021) (cit. on p. viii).

[14] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. „Hyperbolic graph convolutional neural networks." In: *Advances in neural information processing systems*. 2019, pp. 4868–4879 (cit. on p. 109).

[15] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. „ShapeNet: An information-rich 3d model repository." In: *arXiv preprint arXiv:1512.03012* (2015) (cit. on pp. 109, 123).

[16] Oscar Chang, Lampros Flokas, and Hod Lipson. „Principled Weight Initialization for Hypernetworks." In: *Proceedings of International Conference on Learning Representations*. 2020 (cit. on pp. 50, 92).

[17] Changan Chen, Frederick Tung, Naveen Vedula, and Greg Mori. „Constraint-aware deep neural network compression." In: *Proceeding of the European Conference on Computer Vision*. 2018, pp. 400–415 (cit. on p. 37).

[18] He Chen, Pengfei Guo, Pengfei Li, Gim Hee Lee, and Gregory Chirikjian. „Multi-person 3D pose estimation in crowded scenes based on multi-view geometry." In: *Proceedings of the European Conference on Computer Vision*. Springer. 2020, pp. 541–557 (cit. on p. 113).

[19] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. „Compressing neural networks with the hashing trick." In: *Proceedings of the International Conference on Machine Learning*. 2015, pp. 2285–2294 (cit. on p. 16).

[20] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. „Progressive differentiable architecture search: Bridging the depth gap between search and evaluation." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1294–1303 (cit. on pp. 5, 41).

[21] Christopher Choy, JunYoung Gwak, and Silvio Savarese. „4D spatio-temporal convnets: Minkowski convolutional neural networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3075–3084 (cit. on p. 113).

[22] James W Cooley and John W Tukey. „An algorithm for the machine calculation of complex Fourier series." In: *Mathematics of computation* 19.90 (1965), pp. 297–301 (cit. on p. 3).

[23] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. „Binaryconnect: Training deep neural networks with binary weights during propagations." In: *Advances in Neural Information Processing Systems*. 2015, pp. 3123–3131 (cit. on p. 17).

[24] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. „Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or-1." In: *arXiv preprint arXiv:1602.02830* (2016) (cit. on pp. 5, 17).

[25] Elliot J Crowley, Gavin Gray, and Amos J Storkey. „Moonshine: Distilling with cheap convolutions." In: *Advances in Neural Information Processing Systems*. 2018, pp. 2888–2898 (cit. on p. 18).

[26] Navneet Dalal and Bill Triggs. „Histograms of oriented gradients for human detection." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 1. IEEE. 2005, pp. 886–893 (cit. on p. 3).

[27] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. „Convolutional neural networks on graphs with fast localized spectral filtering." In: *Advances in neural information processing systems*. 2016, pp. 3844–3852 (cit. on p. 109).

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. „ImageNet: A large-scale hierarchical image database." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 248–255 (cit. on pp. 50, 52, 94, 96).

[29] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. „Exploiting linear structure within convolutional networks for efficient evaluation." In: *Advances in Neural Information Processing Systems*. 2014, pp. 1269–1277 (cit. on pp. 17, 23, 64).

[30] Xiaohan Ding, Tianxiang Hao, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. „Lossless CNN Channel Pruning via Gradient Resetting and Convolutional Re-parameterization." In: *arXiv preprint arXiv:2007.03260* (2020) (cit. on p. 88).

[31] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. „Learning a deep convolutional network for image super-resolution." In: *Proceeding of the European Conference on Computer Vision*. Springer. 2014, pp. 184–199 (cit. on p. 3).

[32] Xin Dong, Shangyu Chen, and Sinno Pan. „Learning to prune deep neural networks via layer-wise optimal brain surgeon." In: *Advances in Neural Information Processing Systems*. 2017, pp. 4857–4867 (cit. on p. 37).

[33] Heng Fan, Liting Lin, Fan Yang, Peng Chu, Ge Deng, Sijia Yu, Hexin Bai, Yong Xu, Chunyuan Liao, and Haibin Ling. „LaSOT: A high-quality benchmark for large-scale single object tracking." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5374–5383 (cit. on p. 103).

[34] Jonathan Frankle and Michael Carbin. „The lottery ticket hypothesis: Finding sparse, trainable neural networks." In: *arXiv preprint arXiv:1803.03635* (2018) (cit. on p. 88).

[35] Hongyang Gao and Shuiwang Ji. „Graph U-nets." In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2019, pp. 2083–2092 (cit. on pp. 113, 114).

[36] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. „Exact combinatorial optimization with graph convolutional neural networks.“ In: *Advances in Neural Information Processing Systems*. 2019, pp. 15580–15592 (cit. on p. 109).

[37] Ross Girshick. „Fast R-CNN.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1440–1448 (cit. on p. 3).

[38] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. „Rich feature hierarchies for accurate object detection and semantic segmentation.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587 (cit. on p. 3).

[39] Xavier Glorot and Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks.“ In: *Proceedings of the International Conference on Artificial Intelligence and Statistics*. 2010, pp. 249–256 (cit. on p. 50).

[40] Rui Gong, Yuhua Chen, Danda Pani Paudel, Yawei Li, Ajad Chhatkuli, Wen Li, Dengxin Dai, and Luc Van Gool. „Cluster, Split, Fuse, and Update: Meta-Learning for Open Compound Domain Adaptive Semantic Segmentation.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2021, pp. 8344–8354 (cit. on p. viii).

[41] Shuhang Gu, Yawei Li, Luc Van Gool, and Radu Timofte. „Self-guided network for fast image denoising.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 2511–2520 (cit. on p. viii).

[42] Shuhang Gu, Qi Xie, Deyu Meng, Wangmeng Zuo, Xiangchu Feng, and Lei Zhang. „Weighted nuclear norm minimization and its applications to low level vision.“ In: *IJCV* 121.2 (2017), pp. 183–208 (cit. on p. 69).

[43] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. „PCT: Point Cloud Transformer.“ In: *arXiv preprint arXiv:2012.09688* (2020) (cit. on p. 113).

[44] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. „Single path one-shot neural architecture search with uniform sampling.“ In: *Proceedings of the European Conference on Computer Vision*. Springer. 2020, pp. 544–560 (cit. on pp. 5, 41).

[45] David Ha, Andrew Dai, and Quoc V Le. „HyperNetworks.“ In: *Proceedings of International Conference on Learning Representations*. 2017 (cit. on pp. 38, 41, 43, 44).

[46] Pim de Haan, Taco Cohen, and Max Welling. „Natural graph networks.“ In: *arXiv preprint arXiv:2007.08349* (2020) (cit. on p. 113).

[47] Song Han, Huizi Mao, and William J Dally. „Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding.“ In: *Proceedings of International Conference on Learning Representations*. 2015 (cit. on pp. 5, 16, 37).

[48] Song Han, Jeff Pool, John Tran, and William Dally. „Learning both weights and connections for efficient neural network.“ In: *Advances in Neural Information Processing Systems*. 2015, pp. 1135–1143 (cit. on p. 16).

[49] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. „Point2Mesh: A Self-Prior for Deformable Meshes.“ In: *arXiv preprint arXiv:2005.11084* (2020) (cit. on pp. 109, 111, 115, 122, 123, 129).

[50] Karen Hao. *MIT Technology Review: 10 Breakthrough Technologies 2020*. 2020. URL: https://www.technologyreview.com/lists/technologies/2020/#tiny-ai (visited on 12/10/2021) (cit. on p. 37).

[51] Babak Hassibi and David G Stork. „Second order derivatives for network pruning: Optimal brain surgeon.“ In: *Advances in Neural Information Processing Systems*. 1993, pp. 164–171 (cit. on pp. 37, 87, 100).

[52] Kohei Hayashi, Taiki Yamaguchi, Yohei Sugawara, and Shin-ichi Maeda. „Einconv: Exploring Unexplored Tensor Decompositions for Convolutional Neural Networks.“ In: *Advances in Neural Information Processing Systems*. 2019, pp. 5553–5563 (cit. on pp. 37, 40).

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. „Deep residual learning for image recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778 (cit. on pp. 3, 4, 8, 13, 23, 25, 26, 40, 50, 52, 61, 63, 75, 85, 87, 89, 94, 96, 97).

[54] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. „Filter pruning via geometric median for deep convolutional neural networks acceleration." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4340–4349 (cit. on pp. 37, 53, 64, 77).

[55] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. „AMC: AutoML for model compression and acceleration on mobile devices." In: *Proceeding of the European Conference on Computer Vision*. 2018, pp. 784–800 (cit. on pp. 32, 37, 40, 53, 77).

[56] Yihui He, Xiangyu Zhang, and Jian Sun. „Channel pruning for accelerating very deep neural networks." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1389–1397 (cit. on pp. 5, 14, 16, 37, 64).

[57] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. „Distilling the knowledge in a neural network." In: *arXiv preprint arXiv:1503.02531* (2015) (cit. on pp. 5, 18, 93, 94).

[58] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. „Searching for MobileNetV3." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1314–1324 (cit. on pp. 4, 5, 85, 87, 94, 96).

[59] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. „MobileNets: Efficient convolutional neural networks for mobile vision applications." In: *arXiv preprint arXiv:1704.04861* (2017) (cit. on pp. 4, 40, 50, 87, 89, 94, 96).

[60] Jie Hu, Li Shen, and Gang Sun. „Squeeze-and-excitation networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7132–7141 (cit. on p. 4).

[61] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. „RandLA-Net: Efficient semantic segmentation of large-scale point clouds." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11108–11117 (cit. on pp. 109, 114).

[62] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. „Densely connected convolutional networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2261–2269 (cit. on pp. 3, 4, 23, 26, 40, 50, 52, 63, 76, 85, 87, 94, 97).

[63] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. „Single image super-resolution from transformed self-exemplars." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 5197–5206 (cit. on pp. 26, 57, 104).

[64] Zehao Huang and Naiyan Wang. „Data-driven sparse structure selection for deep neural networks." In: *Proceeding of the European Conference on Computer Vision*. 2018, pp. 304–320 (cit. on pp. 53, 64, 67, 77, 79–81).

[65] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. „Speeding up Convolutional Neural Networks with Low Rank Expansions." In: *Proceedings of the British Machine Vision Conference*. 2014 (cit. on pp. 5, 14, 15, 17, 18, 23, 64).

[66] Chiyu Jiang, Jingwei Huang, Andrea Tagliasacchi, Leonidas Guibas, et al. „ShapeFlow: Learnable Deformations Among 3D Shapes." In: *arXiv preprint arXiv:2006.07982* (2020) (cit. on p. 113).

[67] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. „Efficient neural network compression." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12569–12577 (cit. on pp. 32, 53, 77).

[68] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. „Accurate image super-resolution using very deep convolutional networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1646–1654 (cit. on p. 3).

[69] Diederik P Kingma and Jimmy Ba. „Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 26).

[70] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009 (cit. on pp. 26, 50, 52, 75, 94).

[71] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105 (cit. on pp. 3, 4, 13).

[72] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. „Bayesian hypernetworks." In: *arXiv preprint arXiv:1710.04759* (2017) (cit. on p. 41).

[73] Mark Labbe. *Energy consumption of AI poses environmental problems*. 2021. URL: https://searchenterpriseai.techtarget.com/feature/Energy-consumption-of-AI-poses-environmental-problems (visited on 12/10/2021) (cit. on p. 4).

[74] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. „Speeding-up convolutional neural networks using fine-tuned CP-decomposition." In: *Proceedings of International Conference on Learning Representations*. 2015 (cit. on pp. 17, 64).

[75] Yann LeCun, John S Denker, and Sara A Solla. „Optimal brain damage." In: *Advances in Neural Information Processing Systems*. 1990, pp. 598–605 (cit. on pp. 37, 87, 100).

[76] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. „Photo-realistic single image super-resolution using a generative adversarial network." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4681–4690 (cit. on pp. 25, 26, 28, 40, 50, 56, 87, 94, 104).

[77] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. „Self-attention graph pooling." In: *Proceedings of the International Conference on Machine Learning*. PMLR. 2019, pp. 3734–3743 (cit. on pp. 113, 114).

[78] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. „A Signal Propagation Perspective for Pruning Neural Networks at Initialization." In: *arXiv preprint arXiv:1906.06307* (2019) (cit. on pp. 86, 88).

[79] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. „SNIP: Single-shot network pruning based on connection sensitivity." In: *arXiv preprint arXiv:1810.02340* (2018) (cit. on pp. 86, 92, 93).

[80] Fengfu Li, Bo Zhang, and Bin Liu. „Ternary weight networks." In: *arXiv preprint arXiv:1605.04711* (2016) (cit. on p. 5).

[81] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. „Pruning filters for efficient convnets." In: *arXiv preprint arXiv:1608.08710* (2016) (cit. on pp. 53, 77).

[82] Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun. „OICSR: Out-In-Channel Sparsity Regularization for Compact Deep Neural Networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7046–7055 (cit. on pp. 64, 66, 67).

[83] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. „PointCNN: Convolution on χ-transformed points." In: *Advances in Neural Information Processing Systems*. 2018, pp. 828–838 (cit. on pp. 109, 111, 122–125, 128, 129).

[84] Yangyan Li, Soeren Pirk, Hao Su, Charles R Qi, and Leonidas J Guibas. „FPNN: Field probing neural networks for 3D data." In: *Advances in Neural Information Processing Systems*. 2016, pp. 307–315 (cit. on p. 113).

[85] Yawei Li, Kamil Adamczewski, Wen Li, Shuhang Gu, Radu Timofte, and Luc Van Gool. „Revisiting Random Channel Pruning for Neural Network Compression." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2022 (cit. on p. viii).

[86] Yawei Li, Eirikur Agustsson, Shuhang Gu, Radu Timofte, and Luc Van Gool. „CARN: convolutional anchored regression network for fast and accurate single image super-resolution." In: *Proceeding of the European Conference on Computer VisionW*. Springer. 2018, pp. 166–181 (cit. on p. vii).

[87] Yawei Li, Babak Ehteshami Bejnordi, Bert Moons, Tijmen Blankevoort, Amirhossein Habibian, Radu Timofte, and Luc Van Gool. „Spatio-Temporal Gated Transformers for Efficient Video Processing." In: *Advances in Neural Information Processing Systems Workshops*. 2021 (cit. on p. viii).

[88] Yawei Li, He Chen, Zhaopeng Cui, Radu Timofte, Marc Pollefeys, Gregory Chirikjian, and Luc Van Gool. „Towards Efficient Graph Convolutional Networks for Point Cloud Handling.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2021, pp. 2144–2153 (cit. on p. vii).

[89] Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. „Group Sparsity: The Hinge Between Filter Pruning and Decomposition for Network Compression.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020 (cit. on pp. vii, 37, 53).

[90] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. „Learning Filter Basis for Convolutional Neural Network Compression.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 5623–5632 (cit. on pp. vii, 56, 57, 64).

[91] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. „DHP: Differentiable meta pruning via hypernetworks.“ In: *Proceeding of the European Conference on Computer Vision*. Springer. 2020, pp. 608–624 (cit. on pp. vii, 86, 88, 89, 92, 94).

[92] Yawei Li, Wen Li, Martin Danelljan, Kai Zhang, Shuhang Gu, Luc Van Gool, and Radu Timofte. „The Heterogeneity Hypothesis: Finding Layer-Wise Differentiated Network Architectures.“ In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2144–2153 (cit. on p. vii).

[93] Yawei Li, Vagia Tsiminaki, Radu Timofte, Marc Pollefeys, and Luc Van Gool. „3D Appearance Super-Resolution with Deep Learning.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019 (cit. on p. vii).

[94] Yawei Li, Kai Zhang, Jiezhang Cao, Radu Timofte, and Luc Van Gool. „LocalViT: Bringing locality to vision transformers.“ In: *arXiv preprint arXiv:2104.05707* (2021) (cit. on p. viii).

[95] Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. „Exploiting Kernel Sparsity and Entropy for Interpretable CNN Compression.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019 (cit. on pp. 31, 32, 53, 77, 78).

[96] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. „Enhanced deep residual networks for single image super-resolution." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 1132–1140 (cit. on pp. 3, 25, 26, 28, 40, 50, 56, 87, 94, 104).

[97] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. „Towards optimal structured cnn pruning via generative adversarial learning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2790–2799 (cit. on pp. 53, 77).

[98] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. „Sparse convolutional neural networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 806–814 (cit. on p. 16).

[99] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. „Progressive neural architecture search." In: *Proceedings of the European Conference on Computer Vision*. 2018, pp. 19–34 (cit. on pp. 5, 41).

[100] Hanxiao Liu, Karen Simonyan, and Yiming Yang. „DARTS: Differentiable architecture search." In: *Proceedings of International Conference on Learning Representations*. 2019 (cit. on pp. 5, 38, 41, 85).

[101] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. „MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019 (cit. on pp. 37, 38, 40, 41, 43, 44, 49, 54, 55, 88, 92, 96).

[102] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. „Learning efficient convolutional networks through network slimming." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2736–2744 (cit. on pp. 32, 64, 67).

[103] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. „Rethinking the value of network pruning." In: *Proceedings of International Conference on Learning Representations*. 2019 (cit. on pp. 37, 49).

[104] Jonathan Long, Evan Shelhamer, and Trevor Darrell. „Fully convolutional networks for semantic segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3431–3440 (cit. on p. 3).

[105] David G Lowe. „Object recognition from local scale-invariant features." In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2. IEEE. 1999, pp. 1150–1157 (cit. on p. 3).

[106] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. „ShuffleNet V2: Practical guidelines for efficient cnn architecture design." In: *Proceedings of the European Conference on Computer Vision*. 2018, pp. 116–131 (cit. on pp. 4, 85).

[107] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. „Proving the lottery ticket hypothesis: Pruning is all you need." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6682–6691 (cit. on p. 88).

[108] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. „A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics." In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2. IEEE. 2001, pp. 416–423 (cit. on pp. 26, 57, 104).

[109] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. „Occupancy networks: Learning 3D reconstruction in function space." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4460–4470 (cit. on p. 113).

[110] Breton Minnehan and Andreas Savakis. „Cascaded projection: End-to-end network compression and acceleration." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10715–10724 (cit. on pp. 32, 53, 77).

[111] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. „Importance estimation for neural network pruning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11264–11272 (cit. on p. 37).

[112] Matthias Muller, Adel Bibi, Silvio Giancola, Salman Alsubaihi, and Bernard Ghanem. „Trackingnet: A large-scale dataset and benchmark for object tracking in the wild." In: *Proceeding of the European Conference on Computer Vision*. 2018, pp. 300–317 (cit. on p. 103).

[113] Zheyi Pan, Yuxuan Liang, Junbo Zhang, Xiuwen Yi, Yong Yu, and Yu Zheng. „HyperST-Net: Hypernetworks for Spatio-Temporal Forecasting." In: *arXiv preprint arXiv:1809.10889* (2018) (cit. on p. 41).

[114] Neal Parikh, Stephen Boyd, et al. „Proximal algorithms." In: *Foundations and Trends® in Optimization* 1.3 (2014), pp. 127–239 (cit. on p. 69).

[115] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. „Automatic differentiation in PyTorch." In: (2017) (cit. on pp. 23, 49, 76).

[116] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. „Carbon emissions and large neural network training." In: *arXiv preprint arXiv:2104.10350* (2021) (cit. on p. 4).

[117] Bo Peng, Wenming Tan, Zheyang Li, Shun Zhang, Di Xie, and Shiliang Pu. „Extreme network compression via filter group approximation." In: *Proceeding of the European Conference on Computer Vision*. 2018, pp. 300–316 (cit. on pp. 14, 17, 27, 29, 31, 32, 58, 59).

[118] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. „Convolutional occupancy networks." In: *arXiv preprint arXiv:2003.04618* (2020) (cit. on p. 113).

[119] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. „Efficient Neural Architecture Search via Parameters Sharing." In: *Proceedings of the International Conference on Machine Learning*. 2018, pp. 4095–4104 (cit. on p. 41).

[120] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. „PointNet: Deep learning on point sets for 3D classification and segmentation.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 652–660 (cit. on pp. 3, 109, 113, 123, 124).

[121] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. „PointNet++: Deep hierarchical feature learning on point sets in a metric space.“ In: *Advances in neural information processing systems*. 2017, pp. 5099–5108 (cit. on pp. 3, 109, 113, 122–124).

[122] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. „Designing Network Design Spaces.“ In: *arXiv preprint arXiv:2003.13678* (2020) (cit. on pp. 85, 87, 94, 96, 97).

[123] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. „What's Hidden in a Randomly Weighted Neural Network?“ In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11893–11902 (cit. on p. 88).

[124] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. „Xnor-net: Imagenet classification using binary convolutional neural networks.“ In: *Proceeding of the European Conference on Computer Vision*. Springer. 2016, pp. 525–542 (cit. on pp. 5, 17).

[125] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. „Accelerating 3D Deep Learning with PyTorch3D.“ In: *arXiv preprint arXiv:2007.08501* (2020) (cit. on p. 109).

[126] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. *Pytorch3D*. 2020 (cit. on p. 109).

[127] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. „Regularized evolution for image classifier architecture search.“ In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4780–4789 (cit. on pp. 4, 41).

[128] Davis Rempe, Tolga Birdal, Yongheng Zhao, Zan Gojcic, Srinath Sridhar, and Leonidas J Guibas. „CaSPR: Learning Canonical Spatiotemporal Point Cloud Representations." In: *Advances in Neural Information Processing Systems* 33 (2020) (cit. on p. 113).

[129] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. „Faster R-CNN: Towards real-time object detection with region proposal networks." In: *Advances in Neural Information Processing Systems*. 2015, pp. 91–99 (cit. on p. 3).

[130] Alex Renda, Jonathan Frankle, and Michael Carbin. „Comparing rewinding and fine-tuning in neural network pruning." In: *arXiv preprint arXiv:2003.02389* (2020) (cit. on p. 88).

[131] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. „U-Net: Convolutional networks for biomedical image segmentation." In: *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241 (cit. on pp. 40, 50, 59, 87, 94).

[132] Binxin Ru, Pedro Esperanca, and Fabio Carlucci. „Neural Architecture Generator Optimization." In: *arXiv preprint arXiv:2004.01395* (2020) (cit. on p. 41).

[133] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. „MobileNetV2: Inverted residuals and linear bottlenecks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520 (cit. on pp. 4, 40, 50, 87, 89, 94, 96).

[134] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. „PV-RCNN: Point-voxel feature set abstraction for 3D object detection." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10529–10538 (cit. on p. 113).

[135] Weijing Shi and Raj Rajkumar. „Point-GNN: Graph neural network for 3D object detection in a point cloud." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1711–1719 (cit. on p. 109).

[136] Karen Simonyan and Andrew Zisserman. „Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on pp. 3, 4, 13, 26, 63, 76, 85).

[137] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. „Clustering convolutional kernels to compress deep neural networks." In: *Proceeding of the European Conference on Computer Vision*. 2018, pp. 216–232 (cit. on pp. 14, 17, 31, 32, 56, 57, 59).

[138] Emma Strubell, Ananya Ganesh, and Andrew McCallum. „Energy and policy considerations for deep learning in NLP." In: *arXiv preprint arXiv:1906.02243* (2019) (cit. on p. 4).

[139] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on p. 37).

[140] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. „Mnasnet: Platform-aware neural architecture search for mobile." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2820–2828 (cit. on pp. 5, 85, 87, 94, 96).

[141] Mingxing Tan and Quoc V Le. „Efficientnet: Rethinking model scaling for convolutional neural networks." In: *arXiv preprint arXiv:1905.11946* (2019) (cit. on pp. 5, 87, 94, 97).

[142] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. „KPConv: Flexible and deformable convolution for point clouds." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 6411–6420 (cit. on p. 124).

[143] Amirsina Torfi, Rouzbeh A Shirvani, Sobhan Soleymani, and Naser M Nasrabadi. „GASL: Guided Attention for Sparsity Learning in Deep Neural Networks." In: *arXiv preprint arXiv:1901.01939* (2019) (cit. on p. 64).

[144] Frederick Tung and Greg Mori. „Similarity-preserving knowledge distillation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1365–1374 (cit. on pp. 5, 18).

[145] Dominic Zeng Wang and Ingmar Posner. „Voting for voting in online point cloud object detection." In: *Robotics: Science and Systems*. Vol. 1. 3. 2015, pp. 10–15607 (cit. on p. 113).

[146] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. „Graph attention convolution for point cloud semantic segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10296–10305 (cit. on p. 109).

[147] Min Wang, Baoyuan Liu, and Hassan Foroosh. „Factorized convolutional neural networks." In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 545–553 (cit. on pp. 14, 15, 17, 18, 24, 27–30, 32, 56–59).

[148] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. „Dynamic graph CNN for learning on point clouds." In: *Acm Transactions On Graphics (tog)* 38.5 (2019), pp. 1–12 (cit. on pp. 109, 111, 113, 122–125, 128).

[149] Yunxuan Wei, Shuhang Gu, Yawei Li, Radu Timofte, Longcun Jin, and Hengjie Song. „Unsupervised real-world image super resolution via domain-distance aware training." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13385–13394 (cit. on p. viii).

[150] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. „Learning structured sparsity in deep neural networks." In: *Advances in Neural Information Processing Systems*. 2016, pp. 2074–2082 (cit. on pp. 16, 64).

[151] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. „3D ShapeNets: A deep representation for volumetric shapes." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1912–1920 (cit. on p. 123).

[152] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. „Aggregated residual transformations for deep neural networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1492–1500 (cit. on pp. 4, 63, 75).

[153] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. „Grid-GCN for fast and scalable point cloud learning." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5661–5670 (cit. on p. 114).

[154] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. „PC-DARTS: Partial channel connections for memory-efficient architecture search." In: *arXiv preprint arXiv:1907.05737* (2019) (cit. on pp. 5, 41).

[155] Zongben Xu, Xiangyu Chang, Fengmin Xu, and Hai Zhang. „$L_{1/2}$ regularization: A thresholding representation theory and a fast solver." In: *TNNLS* 23.7 (2012), pp. 1013–1027 (cit. on p. 69).

[156] Taojiannan Yang, Sijie Zhu, Chen Chen, Shen Yan, Mi Zhang, and Andrew Willis. „MutualNet: Adaptive ConvNet via mutual learning from network width and resolution." In: *Proceedings of the European conference on computer vision*. Springer. 2020, pp. 299–315 (cit. on p. 96).

[157] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. „NetAdapt: Platform-aware neural network adaptation for mobile applications." In: *Proceeding of the European Conference on Computer Vision*. 2018, pp. 285–300 (cit. on p. 85).

[158] Quanming Yao, James T Kwok, and Xiawei Guo. „Fast Learning with Nonconvex L1-2 Regularization." In: *arXiv preprint arXiv:1610.09461* (2016) (cit. on pp. 69, 74).

[159] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. „Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers." In: *Proceedings of International Conference on Learning Representations*. 2018 (cit. on p. 53).

[160] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. „Hierarchical graph representation learning with differentiable pooling." In: *arXiv preprint arXiv:1806.08804* (2018) (cit. on pp. 113, 114).

[161] Jaehong Yoon and Sung Ju Hwang. „Combined group and exclusive sparsity for deep neural networks." In: *Proceedings of the International Conference on Machine Learning*. JMLR. org. 2017, pp. 3958–3966 (cit. on pp. 64, 66, 67, 81, 82).

[162] Jiahui Yu and Thomas Huang. „AutoSlim: Towards one-shot architecture search for channel numbers." In: *arXiv preprint arXiv:1903.11728* (2019) (cit. on p. 96).

[163] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. „NISP: Pruning networks using neuron importance score propagation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9194–9203 (cit. on pp. 53, 77).

[164] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. „Graph transformer networks." In: *Advances in Neural Information Processing Systems*. 2019, pp. 11983–11993 (cit. on p. 109).

[165] Sergey Zagoruyko and Nikos Komodakis. „Wide residual networks." In: 2016 (cit. on pp. 63, 76).

[166] Roman Zeyde, Michael Elad, and Matan Protter. „On single image scale-up using sparse-representations." In: *Proceedings of International Conference on Curves and Surfaces*. Springer. 2010, pp. 711–730 (cit. on pp. 26, 57, 104).

[167] Dejiao Zhang, Haozhu Wang, Mario Figueiredo, and Laura Balzano. „Learning to share: Simultaneous parameter tying and sparsification in deep learning." In: *Proceedings of International Conference on Learning Representations*. 2018 (cit. on p. 64).

[168] Kai Zhang, Yawei Li, Wangmeng Zuo, Lei Zhang, Luc Van Gool, and Radu Timofte. „Plug-and-play image restoration with deep denoiser prior." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021) (cit. on p. viii).

[169] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. „Beyond a Gaussian denoiser: residual learning of deep CNN for image denoising." In: *IEEE Transactions on Image Processing* 26.7 (2017), pp. 3142–3155 (cit. on pp. 3, 40, 50, 59, 87, 94).

[170] Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. „Learning deep CNN denoiser prior for image restoration." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3929–3938 (cit. on p. 3).

[171] Kuangen Zhang, Ming Hao, Jing Wang, Clarence W de Silva, and Chenglong Fu. „Linked dynamic graph CNN: Learning on point cloud via linking hierarchical features." In: *arXiv preprint arXiv:1904.10014* (2019) (cit. on pp. 109, 111, 122–125, 128, 129).

[172] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. „ShuffleNet: An extremely efficient convolutional neural network for mobile devices.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6848–6856 (cit. on p. 4).

[173] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. „Accelerating very deep convolutional networks for classification and detection.“ In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.10 (2015), pp. 1943–1955 (cit. on pp. 5, 14, 15, 17, 18, 64).

[174] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. „ShellNet: Efficient point cloud convolutional neural networks using concentric shells statistics.“ In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1607–1616 (cit. on p. 109).

[175] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. „Variational convolutional neural network pruning.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2780–2789 (cit. on pp. 53, 77, 78).

[176] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. „Point transformer.“ In: *arXiv preprint arXiv:2012.09164* (2020) (cit. on p. 113).

[177] Yongheng Zhao, Tolga Birdal, Jan Eric Lenssen, Emanuele Menegatti, Leonidas Guibas, and Federico Tombari. „Quaternion Equivariant Capsule Networks for 3D Point Clouds.“ In: *arXiv preprint arXiv:1912.12098* (2019) (cit. on p. 113).

[178] Hao Zhou, Jose M Alvarez, and Fatih Porikli. „Less is more: Towards compact CNNs.“ In: *Proceeding of the European Conference on Computer Vision*. Springer. 2016, pp. 662–677 (cit. on pp. 16, 64).

[179] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. „Deconstructing lottery tickets: Zeros, signs, and the supermask.“ In: *NeurIPS*. 2019, pp. 3592–3602 (cit. on p. 88).

[180] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. „Open3D: A Modern Library for 3D Data Processing.“ In: *arXiv:1801.09847* (2018) (cit. on p. 123).

[181] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. „Trained ternary quantization." In: *arXiv preprint arXiv:1612.01064* (2016) (cit. on p. 17).

[182] Barret Zoph and Quoc V Le. „Neural architecture search with reinforcement learning." In: *Proceedings of International Conference on Learning Representations*. 2017 (cit. on pp. 4, 37, 38, 41).

[183] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. „Learning transferable architectures for scalable image recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8697–8710 (cit. on pp. 5, 38, 41).

ACRONYMS

PSNR	Peak Signal-to-Noise Ratio
SGD	Stochastic Gradient Descent
PGD	Proximal Gradient Descent
HOG	Histogram of Oriented Gradients
SIFT	Scale-Invariant Feature Transform
SURF	Speeded Up Robust Features
FFT	Fast Fourier Transform
AutoML	Automated Machine Learning
AI	Artificial Intelligence
DNNs	Deep Neural Networks
CNNs	Convolutional Neural Networks
GCNs	Graph Convolutional Networks
MLP	Multilayer Perceptron
MLPs	Multilayer Perceptrons
NAS	Neural Architecture Search
DARTS	Differentiable Architecture Search
DHP	Differentiable Hyper Pruning
KNN	K-Nearest Neighbor
SR	Super-Resolution
LW-DNA	Layer-Wise Differentiated Network Architecture
LTH	Lottery Ticket Hypothesis
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit

INDEX

ℓ_1 regularization, 72
ℓ_{1-2} regularization, 72
$\ell_{1/2}$ regularization, 72
logsum regularization, 72

automated machine learning, 37

binary search, 63, 69, 70

Canny detector, 3
carbon footprints, 4
carbon neutrality, 4
channel configuration vector, 90
channel split factor, 6, 7, 15, 20
compact tensor approximation, 7, 61, 135
configuration space, 90
convolutional neural networks, 6

deep neural networks, 3, 4
differentiable architecture search, 38
differentiable hyper pruning, 39
differentiable pruning, 39

evolutionary algorithm, 7, 37

fast Fourier transform, 3
filter basis learning, 6, 14, 15, 35, 135
filter decomposition, 5

graph and subgraph, 114
graph convolution, 115
graph convolutional networks, 9, 109
group sparsity, 8, 62

half-thresholding function, 72
Harris detector, 3
heterogeneity hypothesis, 8
histogram of oriented gradients, 3
hypernetworks, 38, 39, 41, 42, 44, 46–51, 88, 91–93

K-nearest neighbor, 9, 109
knowledge distillation, 5

layer-wise differentiated network architecture, 8, 85
lottery ticket hypothesis, 88
low-rank approximation, 7
low-rank decomposition, 7

multilayer perceptron, 9
multilayer perceptrons, 109

neighborhood centroid distance, 114
neighborhood distance, 114
network pruning, 5
network quantization, 5
neural architecture search, 4, 37
non-structured pruning, 16

peak signal-to-noise ratio, 26
proximal gradient descent, 8, 100
proximal operator, 69, 72–74, 78, 100

reinforcement learning, 37
reinforcement learning , 7

scale-invariant feature transform, 3
soft-thresholding function, 45, 72
sparsity inducing matrix, 62, 66, 67, 70, 73–76, 82
sparsity-inducing matrix, 7
speeded up robust features, 3
stochastic gradient descent, 26
structured pruning, 16
super-resolution, 3

COLOPHON

This document was typeset in LaTeX using the typographical look-and-feel `classicthesis`. Most of the graphics in this thesis are generated using `pgfplots` and `pgf/tikz`. The bibliography is typeset using `biblatex`.