Network Compression and Architecture Search in Deep Learning

BAI, Haoli

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
September 2021

Thesis Assessment Committee

Professor CHAN Lai Wan (Chair)

Professor LYU Rung Tsong Michael (Thesis Supervisor)

Professor KING Kuo Chin Irwin (Thesis Supervisor)

Professor Andrej BOGDANOV (Committee Member)

Professor LIN Hsuan Tien (External Examiner)

Abstract of thesis entitled:

Network Compression and Architecture Search in Deep Learning

Submitted by BAI, Haoli

for the degree of Doctor of Philosophy

at The Chinese University of Hong Kong in September 2021

Deep neural networks have achieved remarkable success in various applications such as computer vision and natural language processing. However, the growing size and computation of deep neural nets also pose new challenges for their development on resource-limited edge devices. In this thesis, we explore network compression and neural architecture search to design efficient deep learning models. Specifically, we aim at addressing several common challenges in network compression and architecture search from the following four parts.

Firstly, we study the problem of few-shot network pruning, a common and practical scenario in network compression. Most existing pruning efforts require full access to the training data to fine-tune the compressed models, however, the access to training data can be restricted in practice due to security or privacy issues. We thus explore the trade-off among data security, network efficiency and performance, by pruning the network with only few-shot training instances (i.e., 5 instances per class). However, it is found that the few-shot data lead to high estimation errors for the pruned model, which shall accumulate and propagate layer-wisely. Towards that end, we propose cross distillation, a new layer-wise knowledge distillation approach to mitigate error propagation, which improves the generalization of the pruned model. In the meanwhile, we also provide theoretical analysis for the proposed approach to guarantee the effectiveness.

Secondly, we study post-training quantization, another popular branch in network compression. Aside from the few-shot training data, post-training quantization further considers the training time and memory overhead, all of which are important dimensions for the quantization pipeline. Especially, the recently developed large pre-trained language

models in natural language processing has made the conventional quantization-aware training rather resource-demanding. Post-training quantization, as the other common approach for quantization training, is more efficient for large pre-trained language models with regarding to both the data consumption and computation power. Specifically, we propose to partition the language model into multiple modules, and minimize the module-wise reconstruction error incurred by quantization. The model partition inspires a new model parallel strategy that enables module-wise local training on separate computing devices. We show that the such parallel training can achieve the nearly the theoretical speed-up, which significantly improves the quantization efficiency with reasonable performance on large pre-trained language models.

In the third part, we continue with quantization on pre-trained language models by exploring BERT binarization. As the limit of quantization, binarization brings the most size reduction, but suffers from large performance degradation even with adequate training resources. Different from the normal training of the full precision BERT or the ternary BERT, we find that the optimization of the binary BERT is troubled by the steep loss landscapes. Motivated by such observations, we propose ternary weight splitting, which initializes the binary BERT by equivalently splitting from a trained half-sized ternary BERT. Thus the binary model inherits the knowledge from the ternary one, and can be fine-tuned for further improvement. Meanwhile, our ternary weight splitting can be also conducted adaptively, allowing a variety of sizes for binary BERT on different edge devices.

Finally, we shift gear to neural architecture search (NAS), an orthogonal approach to design efficient deep learning models. Instead of manually designed criteria for model compression above, NAS directly searches good neural structures from a pre-defined search space. However, NAS algorithms usually suffer from high computation overhead, and thus its algorithmic efficiency is of high priority. Towards that end, we provide a systematic study on parameter sharing, a popular technique to develop efficient NAS algorithms. We first unify existing parameter sharing heuristics with a general framework named affine parameter sharing. We find within the framework that while parameter sharing benefits the training of multiple network candidates jointly, it also challenges the discrimination of good network architectures. Given the observation, a new parameter sharing strategy is proposed to enhance architecture discrimination without the sacrifice of searching efficiency.

論文題目：深度學習中的神經網路壓縮和架構搜索

作者　　：柏昊立

學校　　：香港中文大學

學系　　：計算機科學與工程學系

修讀學位：哲學博士

摘要　　：

深度神經網絡在計算機視覺和自然語言處理等各種應用中取得了巨大的成就。然而，隨著不斷增長的模型大小和運算量，深度神經網絡在資源有限的邊緣設備上的部署也面臨新的挑戰。在本論文中，我們將從網絡壓縮和神經架構搜索兩個角度來設計高效的深度神經網絡。具體地，我們將從以下四個部分來解決網絡壓縮和架構搜索中的常見挑戰。

首先，我們研究絡壓縮中一個常見且實用的場景：小樣本網絡剪枝問題。大多數現有的剪枝工作都需要完全訪問訓練數據來微調壓縮後的模型。然而，由於數據安全或隱私問題，在實際情況中可能會限制對訓練數據的訪問。為了更好的探索數據安全性、網絡效率和性能之間的權衡，我們通過僅僅使用少量訓練數據（例如每類5 個樣本）來進行網絡剪枝。我們發現在小樣本數據下，剪枝模型存在較高的誤差，並且該誤差會沿網絡逐層累積和傳播。為了解決這個問題，我們提出了交叉蒸餾，一種新的逐層知識蒸餾方法，以緩解誤差傳播，提高剪枝模型的泛化能力。與此同時，我們還對所提出的方法進行了理論分析，以保證其有效性。

其次，我們研究網絡壓縮中的另一個流行方法：訓練後量化。除了少量的訓練數據外，訓練後量化進一步將訓練時間和顯存消耗納入量化流程的考量指標。特別地，最近自然語言處理中的大型預訓練語言模型使得傳統的量化感知訓練對資源的需求很大。而訓練後量化作為另一種常見的量化訓練方法，在這種大型預訓練語言模型上的數據消耗和計算開銷的效率更高。具體而言，我們提出將語言模型劃分為多個

模塊，並最小化量化引起的重構誤差。此外，我們的劃分方案支持一種新的模型並行策略，它可以在各自的計算設備上進行模塊化本地訓練。我們的並行策略可以在實踐中達到接近理論的加速比，從而大大簡化預訓練語言模型的量化流程。

在第三部分中，我們探索BERT模型的二值化，研究極限量化對於預訓練語言模型的影響。二值化BERT可以最大化減小模型尺寸，但即使有足夠的訓練資源，其性能也會大幅下降。我們發現，與全精度或三值化BERT的訓練不同，二值化BERT的優化受困於陡峭的損失函數平面。受此啓發，我們提出了三值化權重分裂，即通過從訓練好的一半寬度的三值化模型等價分裂成全尺寸的二值化BERT。因此二值化BERT可以繼承之前三值化模型的知識，並且通過微調能夠進一步改進模型效果。同時，三值化權重分裂也支持自適應拆分，這樣就可以為不同邊緣設備提供相應尺寸的二值化模型。

最後，我們轉向神經網絡架構搜索（NAS），另一類設計高效深度學習模型的方法。與之前手動設計的模型壓縮準則不同，架構搜索直接從預定義的搜索空間中搜索模型結構。然而，NAS通常有著巨大的計算開銷，因此其算法效率一直是研究者高度關注的重點。為此，我們對參數共享在高效NAS算法中的作用進行了系統性的研究。我們提出了仿射參數共享框架，用以統一分析現有參數共享的啓發式策略。根據提出的框架，我們發現參數共享可以同時促進多個網絡候選結構的訓練，但是也削弱了辨別良好網絡結構的能力。基於此觀察，我們設計了一種新的參數共享策略，可以在不影響搜索效率的前提下更好地辨別網絡架構。

Acknowledgement

First and foremost, I would like to thank my supervisors, Prof. Michael Rung-Tsong Lyu and Prof Irwin Kuo-Chin King for their kind supervision throughout my PhD study at CUHK. I am greatly lucky to have their supervision. They help sharpen my research taste and build my research expertise, from maturing a research idea to designing implementation details, and from paper writing to conference presentations.

I would like to extend my gratitude to my thesis assessment committee members: Prof. Laiwan Chan, and Prof. Andrej Bogdanov, for their valuable comments and suggestions to this thesis and all my term presentations. Meanwhile, great thanks to Prof. Hsuan-Tien Lin from National Taiwan University who kindly served as the external examiner for this thesis.

I sincerely thank Lu Hou, my mentor during the internship at Huawei Noah's Ark Lab. Her kind advice and encouragement have a great impact on me. It is my honor to have the support from Xin Jiang, Qun Liu, Lifeng Shang and Wei Zhang in the lab. I also thank my friends there: Xiaoqi Jiao, Zhiqi Huang, Mingyang Yi, for their discussions and help.

I deeply appreciate the support from Jiaxiang Wu, my mentor during the internship at Tencent AI Lab. His passion for coding has always encouraged me to stay focus on my work. I would never forget the wonderful days with Jiaxing Wang, Kuo Zhong, Guanlin Li, Quanle Guo, Xupeng Shi, my closest friends there.

I would like to thank Prof. Zenglin Xu, who initially guides me to the research world. I am also thankful to his lab fellows: Bin Liu, Guangxi Li, Xianghong Fang, Liangjian Wen, for their generous encouragement. I am grateful to my old friends Xin Dong, Yuhang Li and Jiajin Li, for their instructive suggestions on my research.

It is wonderful to spend the 4-year PhD study with my group fellows:

Yuxin Su, Shenglin Zhao, Hongyi Zhang, Xixian Chen, Xiaotian Yu, Jichuan Zeng, Cuiyun Gao, Jiani Zhang, Hou Pong Chan, Jian Li, Wang Chen, Shilin He, Pengpeng Liu, Yue Wang, Han Shao, Wenxiang Jiao, Yifan Gao, Jingjing Li, Weibin Wu, Ziqiao Meng, Xinyu Fu, Zhuangbin Chen, Tianyi Yang, Menglin Yang, Wenxuan Wang, Wenchao Gu, Jentse Huang, Jinyang Liu, Yintong Huo. Many thanks for their help.

Last but most important, I dedicate my greatest gratitude to my parents and my girlfriend. It is their support that helps me overcome the challenges throughout my PhD study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The last decades have witnessed the prosperous development of deep learning in various applications of artificial intelligence, such as computer vision [7, 8, 9], natural language processing [10, 11], speech recognition [12, 13] and recommender systems [14, 15, 16]. The remarkable success comes with the growing size and computation overhead of deep neural networks. Recently, the pre-trained language models and their variants even scale to hundreds of billion parameters [17, 18, 19, 20], outperforming human beings on a variety of tasks.

In the meanwhile, the superior performance of deep learning has also ignited the application of AI services on edge devices, such as smart phones [21], embedded devices [22, 23] and autonomous driving [24]. However, the increasing memory and computation of deep learning models bring a couple of new challenges to AI services on the edge. First of all, the cumbersome size makes the model response extremely slow, which can take up to several minutes for a single user query [17, 20]. Secondly, the huge memory consumption makes the model training and deployment rather difficult on resource-limited devices. For instance, a GPT-3 model takes up to thousands of GPUs for distributed training [17], which is far beyond the limit of edge devices. Thirdly and consequently, most existing efforts deploy these gigantic models on the cloud, which may suffer from network latency, privacy, and security issues during the data transmission.

To mitigate the above challenges, **network compression** and **neural architecture search** are two practical solutions to design

efficient deep learning models. In this thesis, we shall focus on these two directions, and provide a set of solutions to explore the trade-off among speed, memory consumption, and task performance. In the following, we first give a brief overview of network compression and neural architecture search and summarize their potential challenges that motivate our contributions in this thesis.

Network Compression. Starting from an over-parameterized, network compression applies various slimming techniques to obtain the light-weight neural networks. Common network compression techniques include pruning [25, 26, 27], quantizaiton [28, 29, 30, 31, 32], knowledge distillation [33, 34] and low-rank factorization [35, 36]. These techniques can also be combined together for higher compression rate [37, 38]. We shortly introduce each of these techniques in the below.

Network pruning removes parameters or connections in a network that are believed to be redundant. The redundancy can be measured by either parameter magnitude [39], norm of gradients [40, 41] or spectrum of Hessian matrices [42, 43]. Pruning can be generally categorized to unstructured pruning [37, 39] and structured pruning [25, 27]. For unstructured pruning, any useless parameters can be removed, which usually enjoys a higher compression rate but can hardly be accelerated on most existing hard-wares. Structured pruning, on the other hand, can only remove a group of parameters (e.g., channels, filters or layers), which directly leads to a slimmed architecture with faster inference speed.

Quantization, on the other hand, does not require modifying the network architecture, but simply replaces the original full-precision parameters and activations into low-bit fixed point representations [29, 30, 31]. The low-bit representation not only reduces the size of the model but also allows practical speed-up with integer operations on modern hardwares [44]. As the limit of network quantization, binarization is also a popular topic in existing literature [28, 45, 46, 47, 48, 49]. By converting parameters to either ± 1, the network can be accelerated up to $58\times$ with "XNOR" operations [45]. To compensate the performance degradation, quantization-aware training (QAT) is usually conducted to fine-tune the network based on the training data. However, for scenarios with limited training resources, post-training quantization (PTQ) is preferred as it can better balance the trade-off among model compactness, training resources and performance.

Knowledge distillation [33, 34] motivates from a different perspective for model compression. Instead of explicitly compressing the network, knowledge distillation transfers the information from a well-trained teacher model to a pre-defined compact student model. Specifically, the knowledge embedded in the teacher network can be distilled by either the output logits [34] or the hidden representation. The dense vectors of output logits are believed to contain the class-wise correlation, which is empirically shown to benefit the training of student models. The hidden representation, on the other hand, contains task specific knowledge at different levels (i.e., layers) of a network. Mean squared error is the most widely used metric to measure the discrepancy of hidden representation between the teacher and student model [33, 50, 51, 6]. Other common criteria include normalized distances [52] or activation boundaries [53].

Tensor factorization [35, 54] is another common approach for network compression. By factorizing the full rank network parameters into multiple low-rank sub-tensors, the model size can be significantly reduced. The widely approaches for factorization include Canonical Polyadic decomposition [55] and Tucker decomposition [36]. There are also other advanced variants such as tensor-train decomposition [54, 56], which allows more flexibility for tensor decomposition.

Neural Architecture Search. Different from network compression, neural architecture search (NAS) directly searches for an efficient network structure from the pre-defined search space. To evaluate the efficiency of a network, one can adopt metrics such as parameter size, floating-point operations per second (FLOPs) [57, 58], or the real-time latency measured on hard-wares [59, 60]. A NAS algorithm usually consists of three parts: a search strategy that picks network architecture given the efficiency constraint; a search space that determines the composition of neural architecture; and a performance estimation strategy that evaluates each candidate architecture and provides supervision signals back to update the search strategy.

The NAS search space include different types of convolution kernels, connections, activation functions [3, 61, 2], or number of channels [62, 63, 64] and layers in the network [58, 64]. There are also recent efforts that build the search space with different compression strategies, such as different layer-wise pruning ratio [65, 66, 67] or quantization bit-width [66, 68, 69]. The search space design plays an important role in NAS algorithms, where a well-designed search space can greatly improve

the searching outcome [70].

The search strategies of NAS algorithms can be generally divided into three classes: gradient based search [2, 71, 59], reinforcement learning (RL) based search [3, 61, 57], and evolutionary search [72, 62, 73]. The gradient-based search first assigns each candidate choice with a learnable parameter, which can be updated with end-to-end backpropagation. After training, one can take the path with the maximum weights as the final architecture choice. However, such approaches may suffer from the underestimation of network performance, which may lead to sub-optimal architectures. The RL-based algorithms, on the other hand, deploys an extra RL controller for architecture selection. Taking directly the task accuracy or model size as the reward, one can apply policy gradient methods [74] for the training of RL controllers [3, 61]. Finally, evolutionary computing can also be used for NAS problems. Each architecture can be first represented as an individual in the population. With proper fitness evaluation, the promising architectures are then selected to generate new architectures through evolutionary operations, such as Genetic Algorithms [75], Genetic Programming [76] or Ant Colony Optimization [77].

As the third component in NAS, performance estimation is another research focus that is closely related to algorithmic efficiency. NAS is well-known for its high computational cost. When NAS is initially proposed [3], each candidate network needs to be trained and evaluated individually from scratch, which takes up to thousands of GPU days. To mitigate the issue, parameter sharing is a popular solution. Specifically, one can first train a supernet to convergence, where each network candidate can be later sampled and evaluated as a sub-path of the supernet. The procedure is also known as one-shot NAS [78], which significantly reduces the time consumption to only a single GPU day [61].

Challenges and Thesis Focus. Despite the great success of network compression and architecture search in designing efficient deep learning models, there are still a number of practical issues unsolved, which are the main focus of this thesis. To better describe the challenges, we first unify both network compression and neural architecture search in a single paradigm for designing efficient deep learning models, as shown in Figure 1.1. The paradigm includes the user side that requires an efficient deep learning model, as well as the service side that is able to provide services of network compression or neural architecture

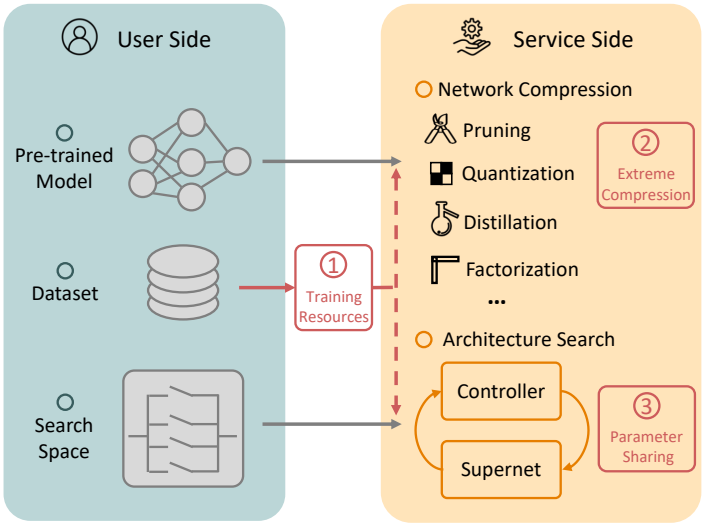


Figure 1.1: The overview of efficient deep learning paradigm. For network compression, users are supposed to provide pre-trained models and training data. For neural architecture search, the pre-defined search space is required aside from the training data. However, the paradigm suffers from three main challenges as shown in red boxes, which are the research focus of this thesis.

search. For network compression services, a pre-trained model should be prepared in advance from the users, which is then passed to the service side. Otherwise, for neural architecture search, the search space needs to be defined based on the desired model types from users. Then various NAS algorithms can be applied within the search space on the service side. For both services, the training dataset is always required to either fine-tune the compressed model or compare the goodness of different trained network architectures. We introduce three challenges associated with such an efficient deep learning paradigm below, which motivate our solutions in the thesis.

- **Challenge 1: Lossless Network Compression with Limited Training Resources.**
  Network compression is no free lunch but also consumes training resources, such as training data and computation overhead. However, either training data or the associated computation resources can be prohibited in practice. First of all, for some domains where data privacy is of high priority (e.g., medical images), it may be risky to expose the entire dataset to third-party organizations. However, the absence of training data may hurt the network performance significantly. In the second place, there are also scenarios that a compressed model is required on the fly when it is prohibited

to conduct the time-consuming fine-tuning over the full training set. Consequently, it is necessary to develop data-efficient and computation-efficient network compression algorithms to meet the demands of the above scenarios in practice.

- **Challenge 2: Extreme Network Compression with Adequate Training Resources.**
  When adequate training resources are available (i.e., training data and computational devices), the network can be thoroughly fine-tuned to recover the degraded performance after model compression. Nevertheless, this can still be challenging when extreme model compression is applied. In network quantization, for instance, while 8-bit quantized models do not suffer from apparent performance drop, binarization (e.g., 1 bit) can significantly hurt the model, e.g., the top-1 accuracy may decrease by more than 20% on ImageNet [45, 47]. Consequently, more advanced quantization algorithms are supposed to be developed to improve the network under such extreme compression.

- **Challenge 3: Efficient Architecture Search with Parameter Sharing.**
  The efficiency of NAS algorithms has been a major concern in the existing literature. As it is computationally intractable to enumerate each candidate model from the search space [3], a common solution is to establish a supernet that embeds different candidate models as its sub-graphs [61, 2, 59, 78] By sharing the parameters of different candidate models, the supernet training is empirically believed to help its sub-networks jointly. However, there are still no rigorous and quantitative studies on the effect of parameter sharing. For instance, what are the potential drawbacks of parameter sharing? How can the controller better identify different network architectures given the shared parameters among various sub-networks?

1.2 Thesis Contributions

Aiming at addressing the aforementioned challenges in network compression and architecture search, the contributions of this thesis are summarized in the following:

- **Pruning with Limited Training Resources.**

  - We study the problem of few-shot network pruning, where only tens of training instances are available. We find that due to the lack of training data, the pruned network suffers from high estimation error that propagates layer-wisely and finally deteriorates the pruned model.
  - We propose cross distillation, a layer-wise knowledge distillation approach for few-shot network pruning. The proposed approach interconnects the teacher and student layer in the forward pass, such that the error propagation can be effectively mitigated.
  - Extensive experiments are conducted on popular network architectures over image classification benchmarks. The empirical results demonstrate the superiority of cross distillation against several existing pruning counterparts.

- **Quantization with Limited Training Resources.**
  - We study post-training quantization of BERT, a widely used pre-trained language model. We seek to improve the quantization performance of BERT given only limited training instances and computational devices.
  - We propose module-wise reconstruction error minimization, an efficient post-training quantization solution that enjoys quick training with limited data and light memory consumption. Additionally, a new model parallel strategy is designed based on the proposed method to further accelerate the training.
  - Extensive experiments are conducted on natural language understanding and reading comprehension tasks. Empirical results show that the proposed approach can accelerate the training by more than 140× with only a minor performance drop.

- **Binarization with Adequate Training Resources.**
  - We study the problem of BERT binarization, which follows quantization-aware training with adequate training resources to boost the performance. However, we find it hard to optimize the binary BERT directly due to its complex loss landscape.
  - We propose ternary weight splitting, an equivalent splitting

approach that initializes the binary BERT from a trained ternary BERT. The initialized binary model is then fine-tuned for further improvement.

  - The proposed ternary weight splitting also supports adaptive splitting, which can flexibly adjust the model size depending on different hardware constraints.

  - We achieve new state-of-the-art BERT quantization results, with 24x size reduction and negligible performance drop.

- **Revisiting Parameter Sharing in Architecture Search.**

  - We revisit the role of parameter sharing in NAS based on the channel number search problem, by establishing a versatile framework that unifies previous hand-crafted parameter sharing heuristics.

  - We quantitatively define parameter sharing in the proposed framework. It is found that parameter sharing indeed promotes training efficiency, but also results in less discrimination of good neural architectures, and vice versa.

  - We thus introduce transitionary affine parameter sharing, a new sharing strategy that balances both searching efficiency and architecture discrimination.

  - Experiments on image classification benchmarks show that transitionary parameter sharing leads to better network architectures, which surpasses a number of competitive channel search counterparts.

1.3 Thesis Organization

An overview of thesis chapters are shown in Figure 1.2. Specifically, the remainder of this thesis is organized as follows:

- **Chapter 2.** In this chapter, we give a systematic background review on network compression and neural architecture search for efficient deep learning. First we introduce common network compression approaches such as network pruning, quantization, knowledge distillation, and matrix/tensor factorization. Then we summarize neural architecture search from three aspects: search space, controller, and performance estimation.

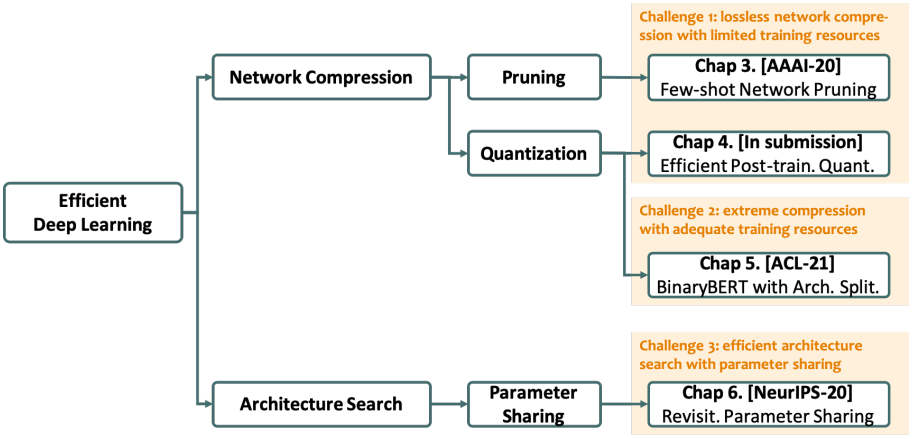


Figure 1.2: The thesis organization.

- **Chapter 3.** In this chapter, we first introduce the problem setting of few-shot network pruning, as well as the underlying challenges. Then we propose cross distillation, which is composed of both correction and imitation for the layer-wise pruning. After that, we show our empirical results on image classification benchmarks and compare them with other few-shot pruning counterparts. Finally, we conclude this work.

- **Chapter 4.** In this chapter, we first compare post-training quantization with quantization-aware training in the context of limited training resources. Then we propose module-wise reconstruction error minimization, an efficient post-training quantization solution for BERT. The approach can be armed with a new model parallel strategy and teacher forcing to further improve both the training efficiency and quantized performance. After that, we provide thorough empirical evaluations with the proposed approach on various tasks of natural language processing across different quantization bit-width. Finally, we conclude this work.

- **Chapter 5.** In this chapter, we first study the challenges of training a binary BERT. Based on the observations, we then propose ternary weight splitting, which equivalently splits from a trained ternary BERT to initialize the binary model. We then show that ternary weight splitting can be performed adaptively depending on the quantization sensitivity and hardware constraints, which can be formulated as a combinatorial optimization problem. In the next, we evaluate our approach on both natural language understanding and reading comprehension, followed by the conclusion of this work.

- **Chapter 6.** In this chapter, we shift the gear to analyze parameter sharing for channel number search problems in neural architecture search. We first establish affine parameter sharing, a unified framework to incorporate previous sharing heuristics. Based on the framework, we then empirically study the advantages and disadvantages of parameter sharing, after which a new transitionary parameter sharing is proposed to balance architecture discrimination and searching efficiency. Finally, we evaluate the proposed transitionary parameter sharing scheme against a number of state-of-the-art baselines on image classification benchmarks.
- **Chapter 7.** The last chapter concludes this thesis and provides some promising research directions for future exploration.

□ **End of chapter.**

Chapter 2

Background

In this chapter, we provide the necessary background review for this thesis. We show that to design efficient deep learning models, there are amount of approaches and categories for both network compression and neural architecture search. An overview of the background taxonomy is presented in Figure 2.1, where red arrows mark the chapter focus in this thesis.

Notation Setup. We first set up the general notations in the background and will stick to them in the remaining chapters unless otherwise specified. We denote a L-layer deep neural network as $\mathcal{F} = f_L \circ f_{L-1} \circ f_1$, where f_l denotes the l-th layer of the network [1]. Common choices for f_l can be convolutional layer in computer vision, or recurrent cells and fully connected layer in natural language processing. Especially, for convolutional layers, the l-th activation $\mathbf{h}_l \in \mathrm{R}^{b \times c_i \times k \times k}$ can be obtained by $\mathbf{h}_l = f_l(\mathbf{h}_{l-1}) = \sigma(\mathbf{w}_l * \mathbf{h}_{l-1})$, where $\sigma(\cdot)$ is some activation function such as $\mathrm{ReLU}(\cdot)$, $*$ denotes the convolutional operation, $\mathbf{w}_l \in \mathrm{R}^{c_o \times c_i \times k \times k}$ is the corresponding 4-D convolutional kernel, and b, c_i, c_o and k are the batch size, input channels, output channels and the kernel size respectively. For fully connected layers, one may simply substitute $*$ with matrix multiplication, and $\mathbf{w}_l \in \mathrm{R}^{c_o \times c_i}$ becomes a 2-D matrix. Moreover, we denote the collection of network parameters as $\mathbf{w} = \{\mathbf{w}_l\}_{l=1}^{L}$.

[1] While there can be more complex network structures, we take such formulation for ease of presentation.

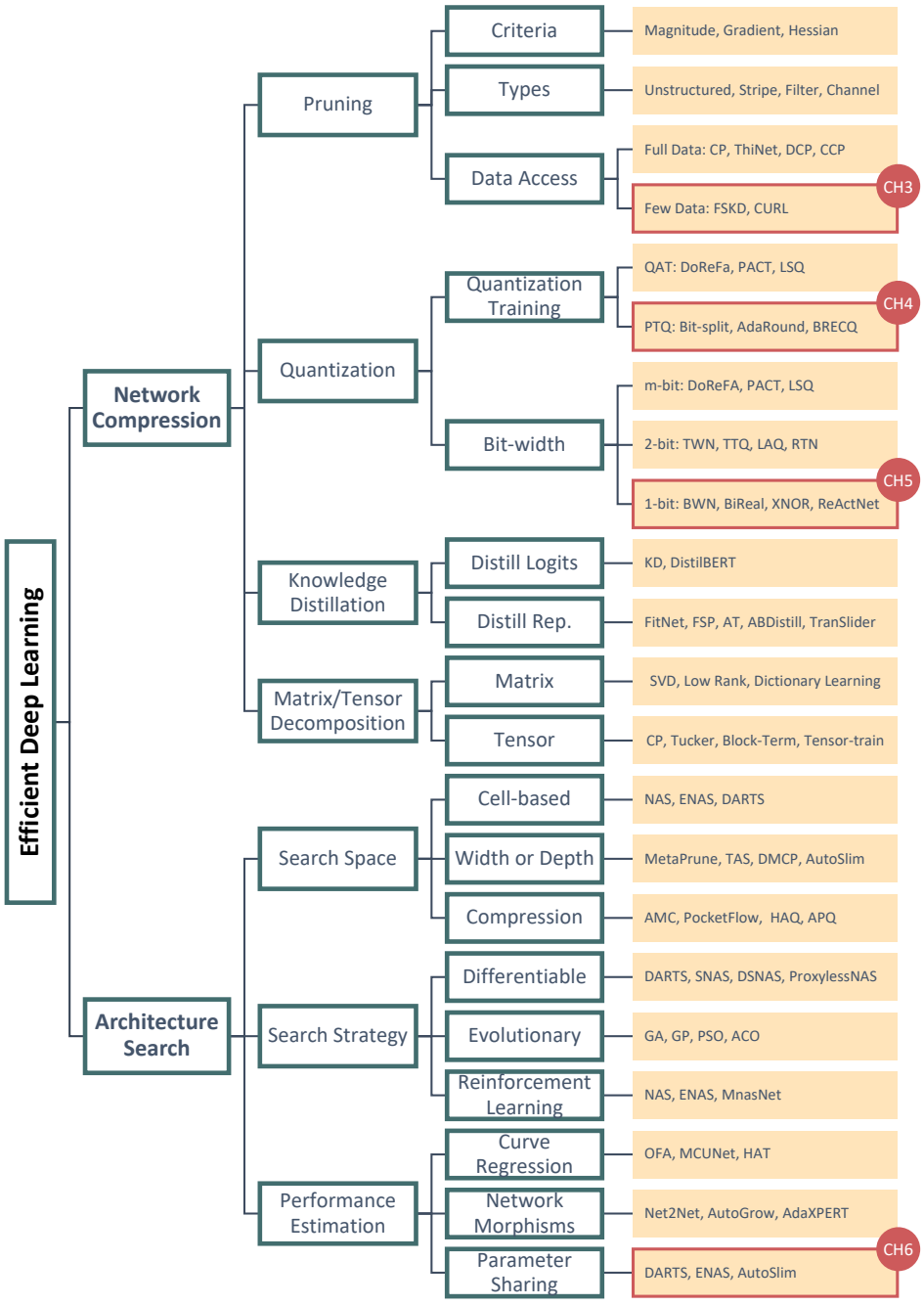


Figure 2.1: The taxonomy of background review.

2.1 Network Compression

2.1.1 Pruning

Deep neural networks are known to be over-parameterized. Motivated by optimal brain damage [79] and optimal brain surgeon [80] back to 1990s, network pruning aims to remove the unimportant connections, filters, or layers in a neural network. This yields a sparse and compact network which can significantly reduce the model size.

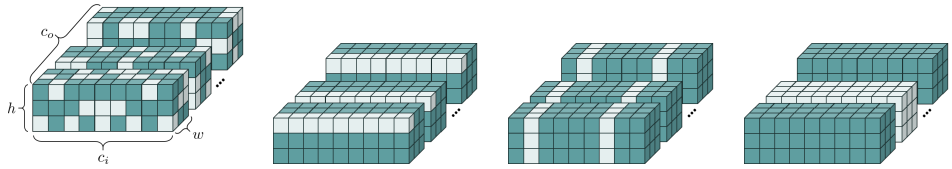
We illustrate network pruning based on convolutional neural networks, while it can be easily extended to other types of networks. Given a convolutional kernel $\mathbf{w} \in \mathrm{R}^{c_o \times c_i \times k \times k}$, network pruning seeks to find a mask $\mathbf{m} \in \{0,1\}^{c_o \times c_i \times k \times k}$ over the network parameter $\tilde{\mathbf{w}} = \mathbf{w} \odot \mathbf{m}$. The masked elements in $\tilde{\mathbf{w}}$ are thus regarded as pruned away. To identify important parameters in the network, there are several pruning criteria as introduced below.

Pruning Criteria. The most commonly used pruning criterion is the parameter magnitude (e.g., ℓ_1 and ℓ_2 norm) [81, 37, 25, 39, 26, 82], based on the assumption that large parameters are more important than the smaller ones. Given the magnitude threshold λ, the mask can be obtained by $\mathbf{m} = \mathbf{1}(|\mathbf{w}| - \lambda > 0)$, where $\mathbf{1}(\cdot)$ is the element-wise indicator function. Aside from the parameter magnitude, the first order (i.e. gradients $\mathbf{g}(\mathbf{w})$) and second order information (i.e. Hessian matrix $\mathbf{H}(\mathbf{w})$) at $\mathbf{w}$ are also well known pruning criteria [42, 83, 43]. Recall that the Taylor expansion of the loss function ℓ at the original parameter $\mathbf{w}$ gives

$$\ell(\tilde{\mathbf{w}}) \approx \ell(\mathbf{w}) + \mathbf{g}(\mathbf{w})^\top(\tilde{\mathbf{w}} - \mathbf{w}) + \frac{1}{2}(\tilde{\mathbf{w}} - \mathbf{w})^\top \mathbf{H}(\mathbf{w})(\tilde{\mathbf{w}} - \mathbf{w}). \tag{2.1}$$

Usually, the first-order term $\mathbf{g}(\mathbf{w})$ dominates the change of loss function [42, 83], as the second-order term is usually negligible. A large loss change indicates the importance of the pruned parameter and vice versa. Nonetheless, to prune a well-trained network at some local minimal with $\mathbf{g}(\mathbf{w}) \approx 0$, the Hessian matrix $\mathbf{H}(\mathbf{w})$ plays the role in determining the pruning mask $\mathbf{m}$ based on the loss curvature [43]. There are also other criteria such as mutual information [84], signal-noise ratio [85] or posterior estimates [86, 87], though they are less popular in practice.

Types of Pruning. Based on the pattern of sparsity in the mask $\mathbf{m}$, pruning can be generally categorized to *unstructured pruning* and



(a) Unstructured Pruning (b) Stripe Pruning (c) Channel Pruning (d) Filter Pruning

Figure 2.2: Illustrations of (a) unstructured pruning, and three kinds of structured pruning: (b) stripe pruning, (c) channel pruning and (d) filter pruning. The light color indicates the pruned parameters.

structured pruning (i.e., stripe pruning, channel pruning, and filter pruning), as shown in Figure 2.2. For unstructured pruning, any unimportant connections of the network parameter can be removed, which allows high flexibility in pruning. A naive way is to sort the magnitudes of all parameters in the ascending order and then prune the top-K elements [39]. However, the irregular pattern of sparsity can be hardly supported for acceleration on modern architectures of CPU or GPU. Structured pruning, on the other hand, removes an entire group of parameters, such as stripe pruning [88] of convolutional kernels in Figure 2.2(b), channel pruning [25, 27] in Figure 2.2(c) or filter pruning [81, 26] in Figure 2.2(d). The regular sparsity in structured pruning can be readily accelerated on general computation hard-wares.

Pruning Pipeline. The most common pipeline for pruning includes three steps: 1) **training from scratch**; 2) **pruning** and 3) **re-training**. Specifically, given a network architecture and randomly initialized parameters, we first train the network to convergence, which is done before pruning. Then we conduct pruning based on the trained network parameters according to the above criteria. Finally, we re-train the pruned network to reboot the performance based on the sparse network.

A popular way is to merge the second step and third step together [39, 81], by minimizing the expected task loss (e.g., the cross entropy loss ℓ_{ce}) over the training set $\mathcal{D}$ with some sparse regularizations as follows:

$$\min_{\mathbf{w}} \mathrm{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\Big[\ell_{ce}(\mathbf{x},\mathbf{y};\mathbf{w}) + \lambda\|\mathbf{w}\|_p\Big], \tag{2.2}$$

where $\|\cdot\|_p$ is the p-norm tuned by λ. For unstructured pruning, ideally one should take $\|\cdot\|_0$ (i.e., $p = 0$). However, in practice $\|\cdot\|_1$ is preferred as a convex relaxation of the sparse learning problem [27]. For structured

pruning, one can take $\|\mathbf{w}\|_{2,1} = \sum_i \|\mathbf{w}_i\|_2$, which encourages group sparsity along filters [81] ($\mathbf{w}_i \in \mathrm{R}^{c_i \times k \times k}$) or channels [25] ($\mathbf{w}_i \in \mathrm{R}^{c_o \times k \times k}$). Equation (2.2) can be directly optimized, which is usually simple and easy to use. However, a drawback is that it can be hard to control the sparsity ratio with the λ. As all network parameters are weighted equally, the resulting sparse network is less adaptive to different layer-wise pruning sensitivity. Existing efforts [39, 25] find that λ need to be annealed at different training epochs, so as to smoothly decrease the network size for better performance. This can also be tricky in practical implementations. Moreover, the objective function in Equation (2.2) or re-training is also time-consuming, i.e., nearly the same amount of time with training from scratch. Therefore, there are more efforts focusing purely on the second step such that more knowledge of the pruned model can be preserved to avoid the intensive re-training.

Towards that end, layer-wise pruning is a common method by minimizing the output discrepancy layer-by-layer in a greedy manner [35, 25, 26, 50]. Specifically, by taking the output $\mathbf{h}_{l+1}$ of the un-pruned l-th layer as the regression target $\mathbf{y}$, the pruning of $\mathbf{w}_l$ can be formulated as a least square problem with norm constraints:

$$\min_{\mathbf{w}_l} \ \|\mathbf{y} - \sigma(\mathbf{w}_l * \tilde{\mathbf{h}}_{l-1})\|_2^2 \quad \text{s.t. } \|\mathbf{w}_l\|_p \leq t, \tag{2.3}$$

where $\tilde{\mathbf{h}}_{l-1}$ is the output of the previously pruned $(l-1)$-th layer, and t is some pre-set pruning target for the p-norm. The formulation also resembles knowledge distillation, as will be introduced in Section 2.1.3. The original model act as the teacher network to provide layer-wise supervision with the pruned model (i.e., the student network). Layer-wise pruning usually outperform Equation (2.2) significantly e.g., on VGG-16, only 1.3% $\downarrow$ in accuracy with 5$\times$ speed-up [25]. Even when armed with fine-tuning, the performance can only be further improved by 1.0%, as the pruned model already maximally preserve the original task knowledge.

Data Accessibility. Training data is crucial for the re-training of pruned networks. However, the data accessibility can be limited in practice, which challenges most existing pruning approaches. As the network compression service is usually provided by some third-party organizations, the customers are required to provide their trained models together with training data. However, data privacy is of high priority for

some domains, such as medical images.

Most existing pruning approaches, such as CP [25], ThiNet [26], DCP [27] and CCP [89] require full training set for sufficient fine-tuning of the pruned model. However, with limited access to training data, these solutions always lead to sub-optimal solutions.

To deal with the scenario without training data, one potential solution is to generate pseudo inputs from random noise. For instance, by optimizing the random noise to match the intermediate statistics of a deep neural network, one can obtain helpful input that is far better than chance [90]. Alternatively, a generative model can be deployed to generate the pseudo training set [91, 92, 93]. However, the generation of high resolution and quality input from both random noise or generative models can be still challenging and expensive, which prohibits their application on large-scale problems.

Challenge and Our Focus. Recently, a new branch of research is explored: few-shot network pruning, which explores the trade-off among network performance, efficiency, and data security. Usually, only a few labeled data are used for few-shot pruning, such as 5 images per class for 5-shot image classification. While there are some preliminary works such as FSKD [50] and CURL [94] in this direction, we will show in Chapter 3 that they still suffer from high estimation errors with the few-shot data. Correspondingly, our solution will be proposed to address the challenge.

2.1.2 Quantization

Network quantization aims to convert the full-precision network parameters or activations into low-bit representations. The quantized networks thus enjoy size reduction and potential inference speed-up with low-bit operations on modern hard-wares. Quantization can be categorized to general multi-bit quantization [29, 30, 31], ternarization (2-bit) [95, 96] and binarization (1 bit) [28, 45, 97, 47]. While the existing quantization literature is vast, here we name a few quantization methods that are related to our research.

Multi-bit Quantization. For a full-precision network parameter $\mathbf{w}$, the general b-bit quantization can be summarized by the function $\mathcal{Q}_b(\cdot)$ as follows:

$$\hat{\mathbf{w}} = \mathcal{Q}_b(\mathbf{w}) = s \cdot \Pi_{\Omega(b)}(\mathbf{w}/s), \tag{2.4}$$

where $\Pi(\cdot)$ is the projection function to the closest integer, $s \in \mathrm{R}^+$ is the step size of quantization, and $\Omega(b)$ is the set of b-bit quantization points.

The multi-bit quantization points $\mathcal{Q}_b$ can be distributed either uniformly or non-uniformly. Uniform quantization is most widely adopted in the quantization literature, where the quantization points are evenly spaced, i.e., $\Omega(b) == \{-2^{b-1}, ..., 0, ..., 2^{b-1} - 1\}$. The design of quantization step-size s has been a major research focus in uniform quantization. A simple way is to set s according to the range of $\mathbf{w}$, i.e., $s = \frac{\max(\mathbf{w}) - \min(\mathbf{w})}{2^b}$. Nonetheless, such a method suffers from outliers in $\mathbf{w}$, especially in convolutional networks when the parameter distribution is highly skewed [29, 98, 99]. To make the quantization distribution more compact, DoReFa [29] re-scales the parameter $\mathbf{w}$ with the following function:

$$\hat{\mathbf{w}} = 2\mathcal{Q}_b(\bar{\mathbf{w}}) - 1, \quad \bar{\mathbf{w}} = \frac{\tanh \mathbf{w}}{2 \max(|\tanh \mathbf{w}|)} + \frac{1}{2}, \tag{2.5}$$

where $\bar{\mathbf{w}}$ squashes the small values in $\mathbf{w}$ and thus leaves the space to represent larger values.

Recent solutions prefer learning-based approaches to narrow down the quantization range. In PACT [30], a parameterized range $[-\alpha, \alpha]$ is set and updated through end-to-end training with the task loss ℓ. To encourage a compact quantization range, L_2 regularization over α is further induced in PACT. Notably, the clipping boundary can be equivalently determined based on the step-size and quantization bit, for instance, $\alpha = s \times 2^{b-1}$ for symmetric quantization. Alternatively, LSQ [31] and LSQ+[100] propose to optimize the quantization step size s instead. Additionally, they find that the step-size gradient $\nabla_s \ell$ should be re-scaled for proper convergence of the parameter. These together are empirically shown to give the state-of-the-art quantization performance so far.

Non-uniform quantization, on the other hand, allows the quantization points to be adjusted to maximize the network performance. Common approaches to design non-uniformly distributed points include k-means clustering [37, 101], logarithm scaling [102, 103, 104], linear combination of codebooks [105, 32] or purely learning-based approaches [38]. While this brings more flexibility for representation, it can be difficult to achieve practical speed-up on most existing hard-wares. Therefore, non-uniform quantization is thus less preferred in practice.

While we mainly focus on parameter quantization in the above paragraphs, similar rules can be applied for activation quantization. However, nonlinear activation functions such as ReLU(·) or GeLU(·) make the activation elements mostly positive, where asymmetric quantization is usually preferred [30, 100]. Meanwhile, to prevent outliers in the activation, a clipping value is also encouraged in the activation function, such as ReLU6(·) [105] or tanh(·) [29].

Ternarization. As a special case of 2-bit quantization, ternarization converts the full-precision parameter into three distinct values, e.g., $\{\pm\alpha, 0\}$, where α is the scaling factor. Ternary Weight Network (TWN) [95] pioneers to ternarize $\mathbf{w}$ element-wisely as

$$\hat{w}_i^t = \mathcal{Q}_2(w_i) = \begin{cases} \alpha \cdot \text{sign}(w_i) & |w_i| \geq \Delta \\ 0 & |w_i| < \Delta \end{cases}, \tag{2.6}$$

where sign(·) is the sign function, $\Delta = \frac{0.7}{n}\|\mathbf{w}\|_1$, and the scaling factor $\alpha = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}|w_i|$ with $\mathcal{I} = \{i \mid \hat{w}_i \neq 0\}$.

To enhance the representation power of ternarized networks, TTQ [96] further assigns different learnable scaling factors for both positive and negative quantized values. While TWN and TTQ determine the scaling factor heuristically, LAQ [104] further consider the loss increase due to ternarization into the scaling factors. Recent ternary network solutions such as RTN [106] follow learning-based approaches to determine the optimal scaling factor from data.

Binarization. By taking only one bit-width, binarization achieves the most size reduction and is widely explored in the quantization literature [28, 45, 97, 47]. As a representative work, Binary-Weight-Network (BWN) [45] aims to minimize the quantization error of binarization as follows:

$$\min_{\alpha,\mathbf{b}} \quad \|\mathbf{w} - \alpha\mathbf{b}\|^2, \tag{2.7}$$

where $\alpha > 0$ is the scaling factor similar to those in TWN [95], and $\mathbf{b} \in \{\pm 1\}^n$ is the binary filter for $\mathbf{w} \in \mathrm{R}^n$. Equation (2.7) can be solved alternatively for $\mathbf{b}$ and α, with the following closed form solutions:

$$\mathbf{b} = \text{sign}(\mathbf{w}), \quad \alpha = \frac{1}{n}\|\mathbf{w}\|_1. \tag{2.8}$$

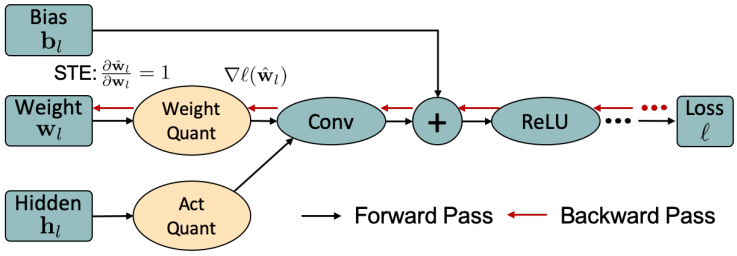


Figure 2.3: The computational graph of a quantized convolutional neural network. The rectangles and circles denote the intermediate tensors and operations respectively. The forward pass first quantizes tensors into the convolutional operation. In the backward pass, the STE operator copies the gradient $\nabla \ell(\hat{\mathbf{w}})$ of quantized $\hat{\mathbf{w}}$ directly to $\mathbf{w}$.

Consequently, the binarized weights can be element-wisely obtained by

$$\hat{w}_i^b = \mathcal{Q}_1(w_i) = \alpha \cdot \text{sign}(w_i). \tag{2.9}$$

Aside from closed-form update of scaling factors. recent efforts on binarized networks prefer learning-based approaches [47, 107, 108], which infer the optimal value of α from data.

Network binarization often suffers from large performance drop. For instance, BWN has nearly the 10% accuracy drop for ResNet-18 [109] on ImageNet dataset. The drop is even more significant with binarized activations. To improve the performance of binary networks, there are orthogonal efforts that design quantization-friendly architectures, such as adding more full-precision short-cut connections [47, 108], designing binarization-friendly activations [49, 110] or increasing channel numbers [49]. Moreover, it is also helpful to combine advanced training techniques such as knowledge distillation [110, 111], sqeeze and excitation (SE) modules [112, 111], soft-to-hard quantization [113, 107].

Quantization Training. Quantization usually degrades the network performance, especially with low bit-width. Therefore, it is necessary for network fine-tuning to recover the performance degradation, similar to network pruning. Figure 2.3 shows the computational graph of a quantized network. Specifically, each forward pass first converts the full-precision weight $\mathbf{w}$ (a.k.a latent weights) to $\hat{\mathbf{w}} = \mathcal{Q}_b(\mathbf{w})$. Then loss $\ell(\hat{\mathbf{w}})$ can be obtained with $\hat{\mathbf{w}}$. In the backward pass, we use $\nabla \ell(\hat{\mathbf{w}})$ to update latent full-precision weights $\mathbf{w}$ due to the non-differentiability of $\mathcal{Q}_b(\cdot)$, which is known as the straight-through estimator (STE) [28]. There are generally two kinds of approaches for

training quantized networks: quantization-aware training (QAT) and post-training quantization (PTQ), as introduced in the following:

Quantization-aware training (QAT) resembles standard fine-tuning, but in the context of quantized networks. Given the training set $\mathcal{D}$, QAT optimizes network latent parameters $\mathbf{w}$ and trainable parameters (e.g., step-sizes $\mathbf{s}$.) in Equation (2.4) by minimizing the training objective ℓ as follows:

$$\min_{\mathbf{w},\mathbf{s}} \mathrm{E}_{\mathbf{x}\sim\mathcal{D}}\ [\ell(\mathbf{x}; \hat{\mathbf{w}}, \mathbf{s})], \quad \text{s.t. } \hat{\mathbf{w}} = \mathcal{Q}_b(\mathbf{w}). \tag{2.10}$$

Typical training objective can be either the cross-entropy loss between the prediction and ground-truth labels for classification tasks [29, 114], or the distillation objective between the quantized model and a full-precision teacher model [38, 6]. QAT usually requires multiple iterations over the full training data $\mathcal{D}$, which can be time-consuming. Furthermore, as discussed in the challenges of network pruning, it may lead to some data security and privacy issues to access the entire training data in practice.

Post-training quantization (PTQ), on the contrary, minimizes the reconstruction error without intensive training over $\mathcal{D}$. One line of research on PTQ quantizes the network without training but aims at removing outliers in the full precision parameters. This can be achieved by splitting an outlier neuron with a large magnitude into two parts [99], where the magnitude can be halved. Alternatively, one can scale down outlier magnitude and multiply it back in subsequent layers, a.k.a weight equalization in [98]. Another solution is to treat the outliers and normal values in the distribution separately, by keeping two sets of quantization parameters [115, 116]. PTQ can also be applied with only activation statistics [98] or pseudo generated data [92, 93].

Another line of PTQ research [117, 118, 119] trains with a very slight portion (a.k.a calibration set) $\tilde{\mathcal{D}} \subseteq \mathcal{D}$ from the original training data, and significantly improves the performance. These approaches target at reconstruction error minimization (REM) by solving the following problem:

$$\min_{\mathbf{w},\mathbf{s}} \|\hat{\mathbf{w}}^\top \hat{\mathbf{a}} - \mathbf{w}^\top \mathbf{a}\|^2, \quad \text{s.t. } \hat{\mathbf{w}} = \mathcal{Q}_b(\mathbf{w}). \tag{2.11}$$

The REM is usually conducted in a greedy manner, i.e., proceed to the matrix multiplication only after the training of previous ones. Meanwhile, Equation (2.11) can be solved quickly with the calibration set $\tilde{\mathcal{D}}$. It is also theoretically justified to be more sample-efficient given limited training size [120] when compared with conventional end-to-end training.

Challenges and Our Focus. We introduce two challenges in network quantization below, which motivate the research focus in this thesis. The first challenge of network quantization lies in the trade-off between QAT and PTQ. While QAT performs generally better, it takes a long period of training over the full time, thus prolongs the production cycle and gives rise to privacy concerns. On the other hand, PTQ is fast and light-weighted to obtain a quantized model in general. However, it usually suffers from a large performance drop, especially when quantized with low bit-width. In Chapter 4, we shall provide our solution that can simultaneously improve the quantized performance without suffering the issues in QAT.

A second challenge is the pursuit of extremely low-bit quantization, especially the binarization of neural networks. As shown in [45], a binary network enjoys more than 50× speed up with "XNOR" operations, which is even faster than integer operations in multi-bit quantized networks. Nevertheless, binarization usually leads to a significant drop in performance. Despite there are a number of approaches [47, 48] proposed to improve network binarization, it is still unclear in what way it degrades the neural network. In Chapter 5, we explore the reasons behind the performance drop and provide our corresponding solution.

Finally, we highlight that the success of quantization depends on both the task and the corresponding model architecture. While existing quantization efforts have achieved remarkable success in computer vision, there is few explorations to other domains such as natural language processing (NLP). Especially, with the recent growth of pre-trained language models [121, 11], there is an increasing demand for model quantization in NLP. Our work in Chapter 4 and Chapter 5 are mostly grounded on Transformer-based models, which also serve as insightful explorations of network quantization in natural language understanding.

2.1.3 Knowledge Distillation

As first proposed in [34], knowledge distillation has been prosperously studied in recent years [33, 52, 122, 53, 50, 123, 124]. Knowledge distillation aims to transfer the information embedded in a teacher network to the student network. Depending on the transfer target, knowledge distillation can be divided into logits-based approaches and hidden representation based approaches, as shown in Figure 2.4. We introduce each type of knowledge distillation in the following.

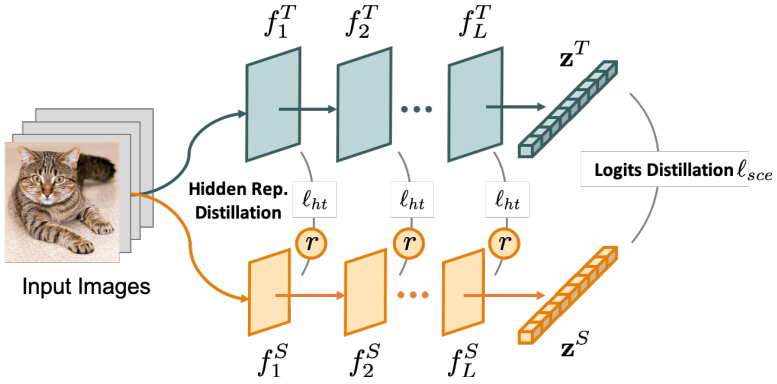


Figure 2.4: Two types of knowledge distillation (i.e., hidden representation distillation and logits distillation) between the teacher network (green) and student network (orange).

Logits Distillation. Usually this can be achieved by minimizing the soft cross entropy ℓ_{sce} between the teacher logits $\mathbf{z}^T$ and student logits $\mathbf{z}^S$ as follows:

$$\ell_{sce} = -\sum_i \text{softmax}(\mathbf{z}_i^S) \cdot \log\Big(\text{softmax}(\mathbf{z}_i^T/\tau)\Big), \tag{2.12}$$

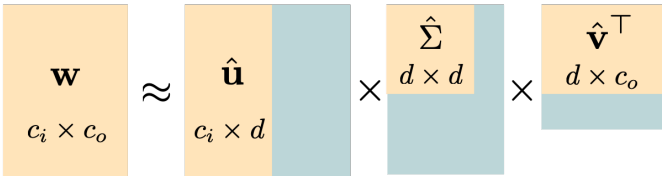
where softmax$(\cdot)$ is the softmax function, and τ is the temperature that is normally set to 1. A large value of τ encourages smooth labels that prefer correlations among classes, and vice-versa.

Hidden Representation Distillation. Hidden representation distillation, on the other hand, minimizes the discrepancy between the intermediate layer of teacher and student networks [33, 52, 51]. For example, FitNet adopts hints training with layer-wise L_2 distances between the teacher and student network as the objective function, i.e.:

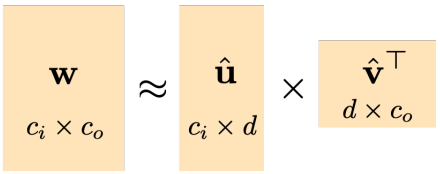
$$\ell_{ht} = \sum_{l=1}^{L} \frac{1}{2} \|f_l^T(\mathbf{x}; \mathbf{w}^T) - r \circ f_l^S(\mathbf{x}; \mathbf{w}^S, \mathbf{w}^r)\|^2, \tag{2.13}$$

where f_l^T and f_l^S denote the l-th layer of the teacher and student network, and $r(\cdot)$ is some regressor function parameterized by $\mathbf{w}^r$ on top of the student layer to match the output dimensions, such as a linear layer. After minimizing ℓ_{ht}, one can shift back to the logit based knowledge distillation objective function ℓ_{sce} defined in Equation (2.12).

Apart from FitNet, there are also other attempts with different distillation metrics. For instance, Attention Transfer (AT) [52] regularizes the task loss with normalized L_2 distances over the feature maps. FSP [122] distills the knowledge from the flow of solution procedure matrices based



(a) Singular Value Decomposition.



(b) Low-rank Decomposition.

Figure 2.5: Two common matrix factorization approaches: (a) Singular value decomposition, and (b) low-rank decomposition for matrix parameters.

on the CNN feature maps. ABDistill [53] explores the knowledge inside the activation boundaries of hidden representations. Translider [125] further explores the degree of knowledge transfer throughout the training process.

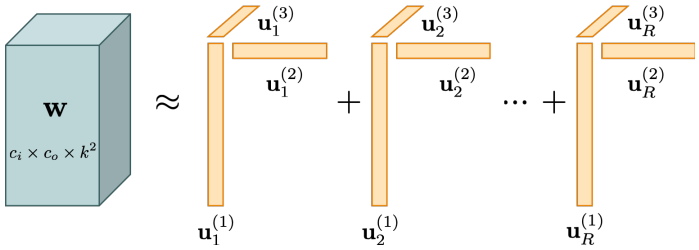
As previously mentioned, knowledge distillation over hidden representations also relates to network pruning, where the original model acts as the teacher to provide supervision signals to the pruned student network [25, 50]. Similarly, there are also attempts on combining knowledge distillation with quantization, where the soft labels from the full precision teacher model is shown to better benefit the quantized student model in the quantization-aware training [38, 6].

2.1.4 Matrix/Tensor Decomposition

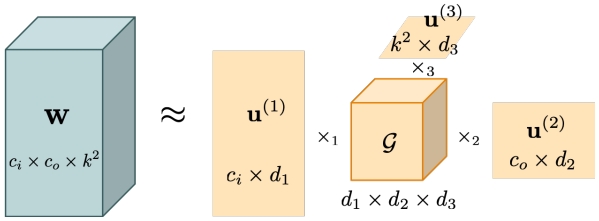
Neural network parameters often exhibit high dimensionality, yet it is usually believed to be highly over-parameterized. Matrix/Tensor decomposition serves as useful tools to reduce the dimensionality of network parameters [35, 55, 36, 54]. Below we briefly review popular approaches of matrix and tensor decomposition in network compression.

Matrix Decomposition. The matrix parameter of a network mostly appears in linear layers. As shown in Figure 2.5(a), singular value decomposition (SVD) for network parameter $\mathbf{w} \in \mathrm{R}^{c_i \times c_o}$ can be conducted in the following way:

$$\mathbf{w} \;=\; \mathbf{u}\Sigma\mathbf{v}^\top \;\approx\; \hat{\mathbf{u}}\hat{\Sigma}\hat{\mathbf{v}}^\top, \tag{2.14}$$



(a) Canonical Polyadic Decomposition.



(b) Tucker Decomposition.

Figure 2.6: Two common tensor factorization approaches: (a) Canonical Polyadic (CP) decomposition, and (b) Tucker decomposition for tensor parameters.

where $\mathbf{u} \in \mathrm{R}^{c_i \times c_i}$ and $\mathbf{v} \in \mathrm{R}^{c_o \times c_o}$ are unitary matrices, and $\Sigma \in \mathrm{R}_+^{c_i \times c_o}$ is a rectangular diagonal matrix with non-negative elements. To represent $\mathbf{w}$ in a compact way, one can truncate the decomposed matrices into $\hat{\mathbf{u}} \in \mathrm{R}^{c_i \times d}, \hat{\mathbf{v}} \in \mathrm{R}^{c_o \times d}$ and $\hat{\Sigma} \in \mathrm{R}_+^{d \times d}$ respectively. Usually the truncation hyper-parameter is set manually, where a small d leads to a large error from the original $\mathbf{w}$, while a large d results in a less compact model.

Aside from SVD, another common approach is low-rank matrix decomposition, as shown in Figure 2.5(b). By decomposing $\mathbf{w}$ into two smaller matrices $\mathbf{u} \in \mathrm{R}^{c_i \times d}$ and $\mathbf{v} \in \mathrm{R}^{c_o \times d}$, one can solve the following problem:

$$\min_{\mathbf{u},\mathbf{v}} \quad \|\mathbf{w} - \mathbf{u}\mathbf{v}^\top\|_F^2. \tag{2.15}$$

Matrix decomposition can be also applied for convolutional kernels, where the 4-dimensional tensor $\mathbf{w} \in \mathrm{R}^{c_i \times c_o \times k \times k}$ can be reshaped as $\bar{\mathbf{w}} \in \mathrm{R}^{c_o \times c_i k^2}$. In such cases, the convolution operations can be done through purely matrix multiplications by expanding the feature map $\bar{\mathbf{h}} \in \mathrm{R}^{c_i k^2, nhw}$, which is also known as the im2col [126] operation.

Tensor Decomposition. Tensor decomposition provides more flexibility in the way of network compression. Below we illustrate tensor decomposition in three dimensions, and introduce two common approaches: Canonical Polyadic (CP) decomposition [55] and Tucker

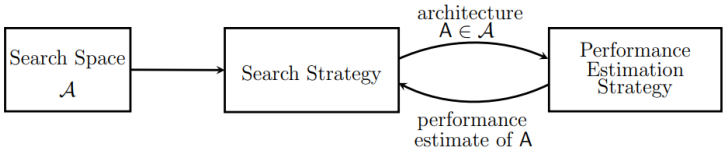


Figure 2.7: An overview of NAS framework from [1]. Given the predefined search space, the search strategy keeps sampling new architectures from the search space. The sampled architectures are then evaluated and ranked based on the performance estimation strategy, which provides feedback to the search strategy to improve its next iteration.

decomposition [36]. Specifically, CP factorizes the convolutional kernel $\mathbf{w} \in \mathrm{R}^{c_o \times c_i \times k^2}$ by three sub-matrices as follows:

$$\mathbf{w} \approx \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \otimes \mathbf{u}_r^{(2)} \otimes \mathbf{u}_r^{(3)}. \tag{2.16}$$

where $\mathbf{u}^1 \in \mathrm{R}^{c_o \times R}, \mathbf{u}^2 \in \mathrm{R}^{c_i \times R}$ and $\mathbf{u}^3 \in \mathrm{R}^{h \times R}, \mathbf{u}^4 \in \mathrm{R}^{w \times R}$ are sub-matrices to recover $\mathbf{w}$, and $\otimes$ is the out product. In practice there is no exact solution for three or higher order tensors. The workhorse algorithm of CP is the alternating least square method, which aims at solving the following problem:

$$\min_{\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)}} \quad \|\mathbf{w} - \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \otimes \mathbf{u}_r^{(2)} \otimes \mathbf{u}_r^{(3)}\|_F^2. \tag{2.17}$$

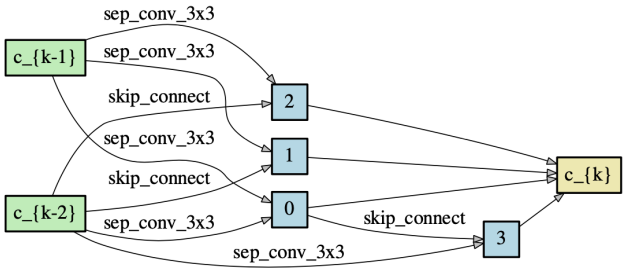
The above equation can be solved by fixing two variables and solve the rest one, which can be reduced to the least square problem.

Tucker decomposition [36] is another common approach to factorize the convolutional kernels. For same convolutional kernel $\mathbf{w}$, Tucker decomposition yields a core tensor $\mathcal{G} \in \mathrm{R}^{d_1 \times d_2 \times d_3}$ together with three factor matrices $\mathbf{u}^{(1)} \in \mathrm{R}^{c_i \times d_1}$, $\mathbf{u}^{(2)} \in \mathrm{R}^{c_o \times d_2}$ and $\mathbf{u}^{(3)} \in \mathrm{R}^{k^2 \times d_3}$ as follows:

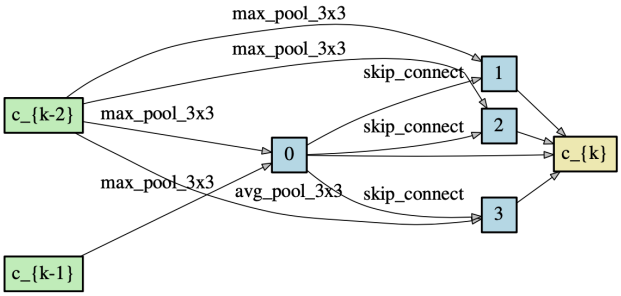
$$\mathbf{w} \approx \mathcal{G} \times_1 \mathbf{u}^{(1)} \times_2 \mathbf{u}^{(2)} \times_3 \mathbf{u}^{(3)}, \tag{2.18}$$

where $\times_k$ is the k-mode product, i.e., multiplying a tensor along the k-th dimension with a matrix or a vector. For a more comprehensive study on tensor decomposition, we refer readers to [127].

Interestingly, we highlight that the decomposition of network parameters also exhibits the change of network architectures, where each sub-matrix acts as a new layer inserted into the network. For instance, the Tucker decomposition over the output and input channels c_o and



(a) Normal Cell.



(b) Reduction Cell.

Figure 2.8: The illustration of cell-based NAS for convolutional neural networks from [2].

c_i leads to the architecture similar to the bottleneck layer in vision models [128, 129, 130]. Therefore, matrix/tensor decomposition can be viewed as an alternative to the design of efficient neural architectures.

2.2 Neural Architecture Search

As an orthogonal approach to network compression, neural architecture search (NAS) aims to automatically search for efficient structures, which has been extensively studied in recent years [3, 61, 131, 132, 57, 59, 133]. NAS can be generally categorized into three components: search space, search strategy, and performance estimation. Given the three aspects, NAS follows the search strategy to automatically explore architectures within a predefined search space, and find the best architecture based on the performance estimation method. An overview of NAS is given in Figure 2.7. Below we introduce each of the three NAS ingredients in detail.

2.2.1 Search Space

The search space of the NAS algorithm is usually predefined manually prior to the searching. Usually, there are three kinds of search space in

NAS: cell-based search, width and/or depth search, or compression-based search.

Cell-based Search. Cell-based search space is widely used in NAS algorithm such as NAS [3], ENAS [61] and DARTS [2]. A cell is typically composed of several nodes, which is the minimal granularity in the searching algorithm. The searched network architecture can thus be obtained by stacking multiple searched cells, which is also known as micro NAS. As an illustrative example in Figure 2.8, we show the searched normal cell and reduction cell of the convolutional neural network from [2]. Figure 2.8 follows the conventional design of cell-based search space: there are usually 7 nodes in a cell, where the first two nodes are the output of previous cells, and the last node is the output of the current cell. For the four nodes in the middle, each shall connect to the other two sampled nodes. Each connection samples from the following operations: a. identity, b. zero, c. 3×3 and d. 5×5 separable convolutions, e. 3×3 and f. 5×5 dilated separable convolutions, g. 3×3 max pooling, and h. 3×3 average pooling. By default, all operations are of stride one so as to keep the spatial resolution of feature maps, except for the reduction cells located at the 1/3 and 2/3 of the total depth of the network.

Aside from cell-based micro NAS that repeats the searched cell to build the network, one can search for different cells individually for the network, a.k.a macro NAS. While macro NAS enlarges the search space that allows the network design to be more flexible, it may thus lead to sub-optimal architecture solutions in practice [70].

Width and Depth Search. The width and depth of a deep neural network are also crucial for its performance. To learn the width of neural networks, MetaPrune [62] first trains a meta-network that contains all width configurations and then uses evolutionary search to find the best performing off-spring. AutoSlim [63], on the other hand, trains a slimmable network that is adaptive to different channel numbers, and then adopts greedy search to sequentially reduce the network width. Similar slimmable networks are also explored on pre-trained language models [134, 135, 136, 41]. There are also efforts that jointly consider the cell-based search space with network width, such as FBNetV2 [64]. Aside from network width, network depth (i.e., number of layers) [58] is another important dimension to improve. For instance, TAS [58] follows

DARTS [131] that imposes learnable parameters to configure the depth and width of the network through end-to-end training.

Compression-based Search. Recent efforts in NAS further considers the elements in network compression as the search space. These strands of research aim to automatically discover the best network compression strategy for the efficient neural architecture model. The search space can be, for instance, different bit-width for network quantizaiton [66, 68, 67], or different sparsity for network pruning [65, 66]. Based on the assumption that different layers exhibit various sensitivity to compression strategies, these efforts are shown to outperform conventional network compression methods.

2.2.2 Search Strategy

Common NAS search strategies can be categorized into differentiable methods [2, 59, 71, 137], reinforcement learning [3, 61, 57], and evolutionary algorithms [72, 73, 62, 138].

Differentiable Methods. Differentiable neural architecture search (DARTS) is originally proposed in [2], and is followed by a number of works such as ProxylessNAS [59], SNAS [71] and DSNAS [137]. We present the illustration of DARTS in Figure 2.9.

Specifically, given the network parameters $\mathbf{w}$ and the associated architecture parameters θ, the architecture output $\bar{\mathbf{o}}$ is represented by the weighted sum of different operations/branches $\mathbf{o}$ as

$$\bar{\mathbf{o}}_i = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)} \mathbf{o}_i(\mathbf{x}), \tag{2.19}$$

where $\mathbf{x}$ is the network input. After the searching, the best architecture i is selected by the element with largest magnitude, i.e., $\theta_i^* = \arg\max(\theta)$. Ideally, the differentiable methods usually can be formulated as the following bi-level optimization problem [139]:

$$\begin{aligned} &\min_{\theta} \ \mathcal{L}_{val}(\mathbf{w}^*(\theta), \theta), \\ &\text{s.t.} \quad \mathbf{w}^*(\theta) = \arg\min_{\boldsymbol{w}} \mathcal{L}_{train}(\mathbf{w}, \theta), \end{aligned} \tag{2.20}$$

where $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ are the loss functions such as the cross-entropy during training and evaluation. The lower level finds the best network parameters $\mathbf{w}^*(\theta)$ corresponding to the associated architecture θ, while

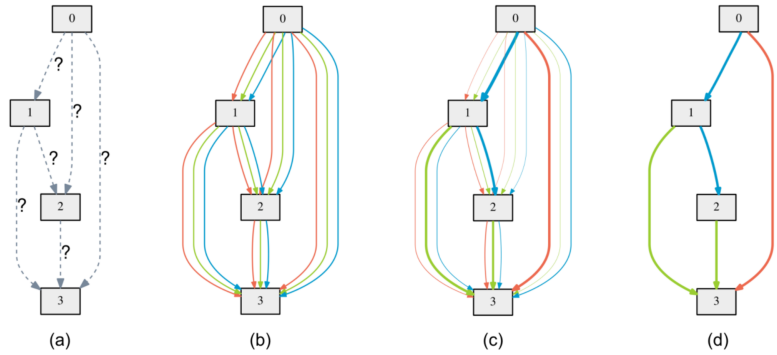


Figure 2.9: The overview of differentiable neural architecture search (DARTS) from [2]. (a) shows the initialized network with unknown operations; (b) shows the continuously relaxed architecture parameters for different connections among cell nodes; (c) illustrates the searching process with preferred operations being thicker; In (d) the operations with the highest probability for each connection are picked to finalize the architecture.

the upper level seeks to optimize over θ given the optimal parameter $\mathbf{w}$. In practice, however, it is intractable to solve Equation (2.20) due to the exhaustive time consumption in the lower level optimization. Instead, another solution is to optimize $\mathbf{w}$ and θ alternatively. Despite it may not lead to optimal solutions, empirical results show the effectiveness of such alternative optimization. More details can be found in [2].

Differentiable search-based NAS is usually simple and efficient and can be readily implemented. However, such approaches also suffer from local minima that requires early stopping techniques [137]. Meanwhile, as the architecture parameters α are coupled with network parameters, it is also not well understood how the scaling issues in the network affect the choice of operations. Given its advantage and drawbacks above, differentiable methods are still actively studied in the NAS community.

Reinforcement Learning. RL is initially applied for NAS in [3], and is further full-filled by plenty of works such as ENAS [61] and MNASNet [57]. The reinforcement learning acts as an hyper-parameter controller $\pi(\theta)$ parameterized by θ for architecture selection. The current network structure thus can be encoded into the states, while the controller output architectures $a \sim \pi(\theta)$ as actions. A majority of RL-based methods such as NAS [3] and ENAS [61] follow the policy gradient method [74], and a recurrent neural network is used for the policy network. For instance, as illustrated in Figure 2.10, the recurrent network sequentially output layer-wise network operations in different

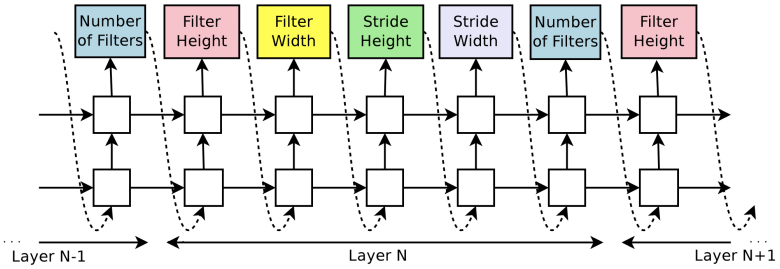


Figure 2.10: Recurrent neural network for the RL controller to search convolutional neural networks [3] .

time steps. A key of RL-based algorithms is the design of the reward function. A simple reward function is the evaluation accuracy on the validation set [3, 61]. Recent works further incorporate the efficiency of neural architectures, by introducing penalty on the model size in the reward function. For instance, the reward in [57] is designed as:

$$\mathcal{R} = \text{Acc}(a) \times [\frac{\mathcal{B}(a)}{B}]^{\gamma}, \quad \text{where } \gamma = \begin{cases} \alpha & \text{if } \mathcal{B}(a) < B \\ \beta & \text{otherwise} \end{cases}, \tag{2.21}$$

where $\mathcal{B}\Big(a\Big)$ is the network budget function (e.g. FLOPs) and B is the budget constraint, and α, β are application-specific constants.

In a similar spirit to the bi-level optimization in Equation (2.20), the RL-based NAS seeks to maximize the expectation of reward function $\mathcal{R}\Big(a, \mathbf{w}^*(a)\Big)$ based on the associated optimal parameter $\mathbf{w}^*(a)$ as follows:

$$\begin{aligned} &\max_{\theta} J(\theta) = \text{E}_{a\sim\pi_\theta}\mathcal{R}(a)\Big(a, \mathbf{w}^*(a)\Big), \\ &\text{s.t. } \mathbf{w}^*(a) = \arg\min_{\mathbf{w}(a)} \mathcal{L}\Big(a, \mathbf{w}(a)\Big) \text{ and } \mathcal{B}\Big(a\Big) \leq B, \end{aligned} \tag{2.22}$$

where the REINFORCE algorithm [140] can be applied to calculate the gradient.

Evolutionary Methods. Evolutionary search offers population-based paradigms to solve the challenging optimization in NAS. An overview of the evolutionary process for NAS is presented in Figure 2.11 Specifically, each network architecture is encoded into an individual in the population, which can be represented by binary strings according to pre-set mapping rules. In the initial space, a set of population is initialized manually from the architectures in the predefined search space. The population is then fed to the fitness evaluation module, which uses the evaluation

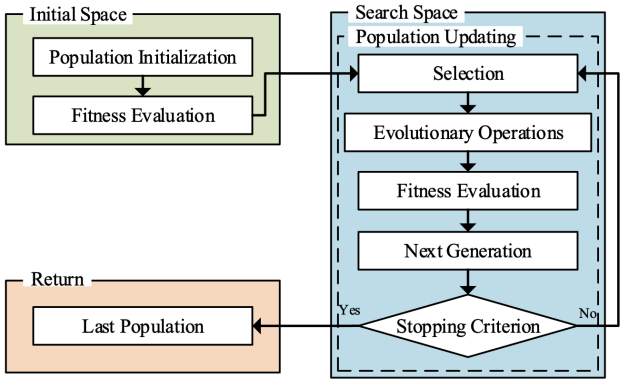


Figure 2.11: The framework of evolutionary NAS from [4].

accuracy or model size to judge the goodness of architectures. In the search space, the algorithm repeatedly goes through the following steps: 1) select the promising generations; 2) apply evolutionary operations to population encodings to generate new candidates; 3) evaluate the fitness of new candidates; 4) pick out the good offspring as the next generation; 5) determine if the stopping criteria is met (i.e., reaching the maximum iteration). The algorithm will return the final searched architecture given the stopping signal, otherwise, it goes back to step 1).

Genetic algorithms (GA) [75] is most widely used for evolutionary NAS on image classification [72, 141, 142, 143, 62] and language modeling [144, 138, 73, 145]. Other common strategies for evolutionary NAS include Genetic Programming (GP) [76], Particle Swarm Optimization (PSO) [146], and Ant Colony Optimization (ACO) [77]. A complete survey on evolutionary methods for NAS can be found in [4].

2.2.3 Performance Estimation

Performance estimation is crucial for NAS algorithms for different searching strategies. However, it is always a trade-off for performance estimation between efficiency and accuracy. As previously mentioned in Equation (2.20), it is computationally intractable to train each network architecture to convergence before evaluating its performance, especially on large-scale problems such as ImageNet [7]. To mitigate the issue, we introduce several solutions below and point out the potential challenges that motivate our research on NAS in the thesis.

Performance Prediction. Instead of evaluating each individual network architecture, one can directly predict the network performance

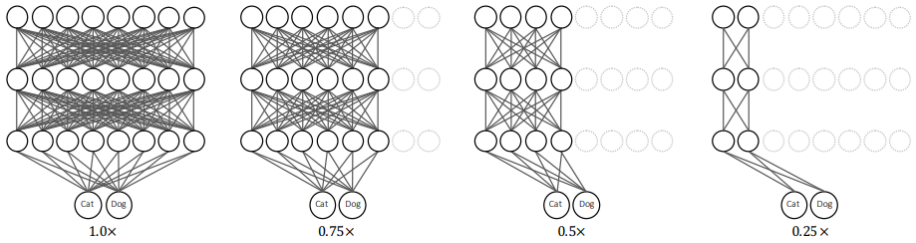


Figure 2.12: An illustration of parameter sharing from [5]. The four choices of widths are ordinally overlapped along channel indices.

given the historical records from other architectures. [147, 148, 149, 150, 59, 60]. Given the observed performance from existing candidates, one can extrapolate the learning curves [147, 151, 149] of network architectures. Another way is to train a separate predictor that takes the architecture encodings as the input and directly outputs the performance [150, 60]. The training data of the predictor lie in the format of [architecture, accuracy] pairs, which is scarce in nature. Therefore, the predictor model is usually designed to be light-weighted so as to be more sample efficient, such as LSTMs or fully connected MLPs used in [150].

Network Morphisms. Network morphisms aim to inherit the weights from previously trained network architectures, so as to avoid the time-consuming re-training of new candidate models [152, 72, 153, 154, 155, 156]. A key point in network morphisms is the equivalent architecture modification. Given the same input, the network exhibits exactly the same output before and after modification, such that the knowledge inside the previous network can be inherited in the new architecture. Net2Net[152] is one of the early attempts for network morphisms, which is able to adjust the width or depth of networks with function-preserving transformations. There are also recent efforts that target at steepest descent with architecture splitting [157, 158, 159], which are found to achieve lower minima on the new loss landscape of split architectures. As the side-effect of network morphisms, the network can only grow large to satisfy the strict knowledge inheritance from previous models. Consequently, the NAS algorithm may have less flexibility to look back and refine smaller network architectures.

Parameter Sharing. Finally, we introduce parameter sharing, the most popular solution for efficient performance estimation. Parameter sharing is built for one-shot neural architecture search [78, 132, 160, 63, 64], which first trains a supernet that include all network candidates as

its sub-paths. The sub-network parameters can be shared in the level of convolutional kernels, channels, or even layers from the supernet. As an illustrative example, Figure 2.12 shows four different choices of network widths overlapped ordinally along channel indices. After the training of supernet, one can directly evaluate different sub-networks from the supernet, which greatly improves the efficiency of performance estimation, e.g., only a few GPU days in [2, 71, 59]

Challenge and Our Focus. Despite the improved efficiency of NAS algorithms, one-shot NAS with parameter sharing suffers underestimation of network performance. Intuitively, as different networks are coupled inside the supernet, their training can mutually influence each other, which makes their performance highly correlated. Aside from intuition, there is little rigorous study so far that analyzes the potential benefits and drawbacks of parameter sharing. In Chapter 6, we shall investigate the mechanism behind parameter sharing for a better understanding of one-shot NAS algorithms.

□ **End of chapter.**

Chapter 3

Few-shot Network Pruning via Cross Distillation

In this chapter, we study few-shot network pruning, a new yet practical setting of compressing deep neural networks. Most existing prevalent pruning pipeline requires fine-tuning with sufficient training data to reboot the accuracy of compressed models. However, access to the training data may bring privacy and security issues. As a compromise between privacy and performance, we investigate the following problem: given few samples per class, how can we effectively prune a network with negligible performance drop? We find the core challenge of few-shot network pruning lies in high estimation errors from the original network during inference, since the compressed network can easily overfit on the few training instances. The estimation errors could accumulate and propagate layer-wisely, and finally deteriorate the network output. To address the problem, we propose cross distillation, a novel layer-wise knowledge distillation approach. By interweaving hidden layers of the teacher and student network, layer-wisely accumulated estimation errors can be effectively reduced. Extensive experiments on benchmark datasets demonstrate that cross distillation can significantly improve the pruned network's accuracy when only a few training instances are available.

3.1 Introduction

Deep neural networks (DNNs) have achieved remarkable success in a wide range of applications, however, they suffer from substantial computation and energy costs. In order to obtain light-weighted DNNs, network pruning techniques have been widely developed in recent

years [35, 25, 26, 161, 27].

Despite the success of previous efforts, a majority of them rely on the whole training data to reboot the compressed models, which could suffer from security and privacy issues. For instance, to provide a general service of network pruning, the reliance on the training data may result in data leakage for customers.

To take care of security issues in network pruning, some recent works [91, 162, 90] motivate from knowledge distillation [34, 33], and propose data-free fine-tuning by constructing pseudo inputs from the pre-trained teacher network. However, these methods highly rely on the quality of the pseudo inputs and are therefore limited to small-scale problems.

In order to obtain more scalable pruning algorithms given the security concern, a compromise between privacy and performance is to compress the network with few-shot training instances, e.g., 1-shot for one training instances per class. Prevalent works [50, 163] along this line extend knowledge distillation by minimizing layer-wise estimation errors (e.g., Euclidean distances) between the teacher and student network. The success of these approaches largely comes from the layer-wise supervision from the teacher network. Nevertheless, a key challenge in few-shot network pruning is rarely investigated in previous efforts: as there are few-shot training samples available, the student network tend to over-fit on the training set and consequently suffer from high estimation errors from the teacher network during inference. Moreover, the estimation errors could propagate and accumulate layer-wisely [43] and finally deteriorate the student network.

To deal with the above challenge, we proceed along with few-shot network pruning and propose *cross distillation*, a novel layer-wise knowledge distillation approach. Cross distillation can effectively reduce the layer-wisely accumulated errors in the few-shot setting, leading to a more powerful and generalizable student network. Specifically, to correct the errors accumulated in previous layers of the student network, we direct the teacher's hidden layers to the student network, which is called correction. Meanwhile, to make the teacher aware of the errors accumulated on the student network, we reverse the strategy by directing the student's hidden layers to the teacher network. With error-aware supervision from the teacher, the student can better mimic the teacher's behavior, which is called imitation. The correction and

imitation compensate each other, and to find a proper trade-off, we propose to take convex combinations between either loss functions of the two procedures, or hidden layers of the two networks. To better understand the proposed method, we also give some theoretical analysis on how convex combination of the two loss functions manipulates the layer-wisely propagated errors, and why cross distillation is capable of improving the student network.

Extensive experiments and ablation studies are conducted on popular network architectures and benchmark datasets, and the results demonstrate that our proposed method can effectively reduce the estimation errors and improve the pruned model in the few-shot setting, outperforming a number of competitive baselines.

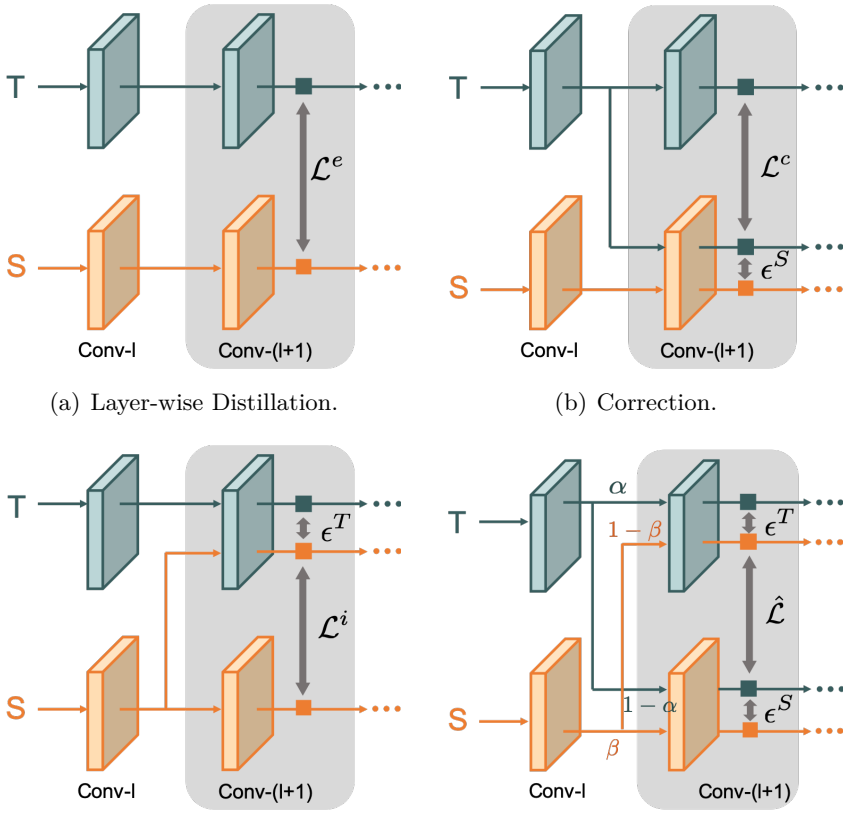
3.2 Methodology

Problem Setup. Given few-shot training instances $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ with N samples in total, we aim at pruning a well-trained network with its performance maximally preserved. Towards that end, common approaches for few-shot network pruning follow layer-wise knowledge distillation. Unlike the standard knowledge distillation [34], such an approach receives layer-wise supervision from the teacher, and is shown to be more sample-efficient both empirically [25, 26] and theoretically [120].

Specifically, based on the notation in Section 2, we denote the teacher network (i.e. the original model) as $\mathcal{F}^T$ and the student network (i.e., the pruned model) as $\mathcal{F}^S$, respectively. The corresponding l-th convolutional feature map of are denoted as $\mathbf{h}_l^T$ and $\mathbf{h}_l^S$ for the teacher network and student network respectively. Batch normalization layers are omitted as they can be readily fused into convolutional. In the following, we drop the layer index l for clean notation. As is shown in Figure 3.1(a), with previous layers being fixed, layer-wise distillation aims to find the optimal $\mathbf{w}_*^S$ that minimizes the Euclidean distance between $\mathbf{h}^T$ and $\mathbf{h}^S$, i.e.,

$$\mathbf{w}_*^S = \arg\min_{\mathbf{w}^S} \frac{1}{N} \|\mathbf{w}^T * \mathbf{h}^T - \mathbf{w}^S * \mathbf{h}^S\|_F^2 + \lambda \mathcal{R}(\mathbf{w}^S), \tag{3.1}$$

where for simplicity we denote $\mathcal{L}^e(\mathbf{w}^S) = \|\mathbf{w}^T * \mathbf{h}^T - \mathbf{w}^S * \mathbf{h}^S\|_F^2$ as the *estimation error*, $\mathcal{R}(\mathbf{w}^S)$ is the regularization introduced in Section 2.1.1, and λ tunes the regularization strength. Despite one can obtain a compact network with Equation (3.1), it is hardly noticed that the student network $\mathcal{F}^S$ tends to suffer from high estimation errors



(a) Layer-wise Distillation. (b) Correction.

(c) Imitation. (d) Soft Cross Distillation.

Figure 3.1: The four categories of layerwise distillation. a) is the traditional pattern; b) guides the teacher to student in order to compensate estimation errors on the student; c) guides the student to the teacher to make it aware of the errors on the student; d) offers a soft connection to balance b) and c) .

during testing, since it can be easily over-fitted to the few-shot training instances. Moreover, the estimation error shall propagate and enlarge layer-wisely [43], and finally lead to a large performance drop on $\mathcal{F}^S$.

3.2.1 Cross Distillation: Formulation

To address the above issue, we propose cross distillation, a novel layer-wise distillation method for few-shot network pruning. Since the estimation errors are accumulated on the student network $\mathcal{F}^S$ and $\mathbf{h}^T$ are taken as the target during layer-wise distillation, we direct $\mathbf{h}^T$ to $\mathcal{F}^S$ in substitution of $\mathbf{h}^S$ to reduce the historically accumulated errors, as is shown in Figure 3.1(b). We thus minimize the mean square error of convolutional outputs, which is defined as *correction loss*:

$$\mathcal{L}^c(\mathbf{w}^S) = \|\mathbf{w}^T * \mathbf{h}^T - \mathbf{w}^S * \mathbf{h}^T\|_F^2. \tag{3.2}$$

In the forward pass of $\mathcal{F}^S$, however, directing $\mathbf{h}^T$ to $\mathcal{F}^S$ results in inconsistency $\epsilon^S = \|\mathbf{w}^S * \mathbf{h}^T - \mathbf{w}^S * \mathbf{h}^S\|_F^2$ because $\mathcal{F}^S$ takes $\mathbf{h}^T$ from $\mathcal{F}^T$ in the training while it is expected to behave along during inference. Therefore, minimizing the regularized $\mathcal{L}^c$ could lead to a biasedly optimized student net.

In order to maintain the consistency during forward pass for $\mathcal{F}^S$ and simultaneously make the teacher aware of the accumulated errors on the student net, we can inverse the strategy by guiding $\mathbf{h}^S$ to $\mathcal{F}^T$, as is shown in Figure 3.1(c). We call this process as *imitation*, since the student network tries to mimic the behavior of the teacher network given its current estimations. Similarly we minimize the mean square error between the corresponding convolutional outputs, defined as the *imitation loss*:

$$\mathcal{L}^i(\mathbf{w}^S) = \|\mathbf{w}^T * \mathbf{h}^S - \mathbf{w}^S * \mathbf{h}^S\|_F^2. \tag{3.3}$$

Despite the teacher network now can provide error-aware supervised signal, such connection brings inconsistency on the teacher network, i.e., $\epsilon^T = \|\mathbf{w}^T * \mathbf{h}^S - \mathbf{w}^T * \mathbf{h}^T\|_F^2$. As a result of ϵ^T, the errors in $\mathbf{h}^S$ is be enlarged by $\mathbf{w}^T$ during layer-wise propagation, leading to deviated supervision for $\mathcal{F}^S$ that deteriorates the distillation.

Consequently, the correction loss $\mathcal{L}^c$ and the imitation loss $\mathcal{L}^i$ compensate each other. We thus introduce convex combination for the two terms tuned by μ as follows:

$$\tilde{\mathcal{L}} = \mu\mathcal{L}^c + (1-\mu)\mathcal{L}^i, \quad \mu \in [0, 1]. \tag{3.4}$$

Substituting $\mathcal{L}^e$ in Equation (3.1) with $\tilde{\mathcal{L}}$ yields the objective function for cross distillation.

3.2.2 Theoretical Analysis

The inconsistency gaps ϵ^T and ϵ^S of cross distillation make it still unclear how the proposed method manipulates the propagation of estimation errors, and why minimizing the regularized $\tilde{\mathcal{L}}$ is in the right direction to improve the student net $\mathcal{F}^S$. To theoretically justify cross distillation, we follow [164] to substitute $\mathcal{L}^e$ with $\tilde{\mathcal{L}}$, and equivalently reformulate the unconstrained problem in Equation (3.1) to the constrained optimization problem as

$$\min_{\mathbf{w}^S \in \mathcal{C}} \tilde{\mathcal{L}}, \quad \mathcal{C} = \{\mathbf{w}^S \mid \mathcal{R}(\mathbf{w}^S) \leq \epsilon(\lambda)\}, \tag{3.5}$$

where $\mathcal{C}$ is a compact set determined by the regularization $\mathcal{R}(\mathbf{w}^S)$ and λ. With Equation (3.5), we can now bound the gap of cross entropy between $\mathcal{F}^T$ and $\mathcal{F}^S$ for classification[1] with the following theorem.

Theorem 3.2.1. *Suppose both $\mathcal{F}^T$ and $\mathcal{F}^S$ are L-layer convolutional neural networks followed by the un-pruned softmax fully-connected layer. If the activation functions $\sigma(\cdot)$ are Lipchitz-continuous such as* ReLU(), *the gap of softmax cross entropy $\mathcal{L}^{ce}$ between the network logits $\mathbf{o}^T = \mathcal{F}^T(\mathbf{x})$ and $\mathbf{o}^S = \mathcal{F}^S(\mathbf{x})$ can be bounded by*

$$|\mathcal{L}^{ce}(\mathbf{o}^T;\mathbf{y}) - \mathcal{L}^{ce}(\mathbf{o}^S;\mathbf{y})| \leq C\tilde{\mathcal{L}}_L + \sum_{l=1}^{L-1}\prod_{k=l}^{L} C_k^{'}(\mu)\tilde{\mathcal{L}}_l, \tag{3.6}$$

where C and $C^{'}(\mu)$ are constants and $C^{'}(\mu)$ is linear in μ.

Theorem 3.2.1 shows that 1) the gap of cross entropy between the student network $\mathcal{F}^S$ and teacher network $\mathcal{F}^T$ is upper bounded by $\tilde{\mathcal{L}}$, and therefore layer-wise minimization of the constrained optimization problem in Equation (3.5) could decrease the gap of cross entropy and finally improve $\mathcal{F}^S$. 2) The tightness of the upper bound is controlled by the trade-off hyper-parameter μ, which is a L-th order polynomial. A proper choice of μ may lead to a tighter bound that could better decrease the cross entropy gap.

We provide a proof sketch to Theorem 3.2.1 in two parts. We first show the Lipchitz continuity for the softmax cross entropy function in Lemma 1, then we show the layer-wise propagation of estimation errors in a recursive way in Lemma 2. Theorem 3.2.1 can be easily verified by combining Lemma 1 and Lemma 2.

Lemma 1. *For the network logits $\mathbf{o} = \mathcal{F}(\mathbf{x}) \in R^d$ and labels $\mathbf{y} \in R^d$, the softmax cross entropy $\mathcal{L}^{ce}(\mathbf{o};\mathbf{y}) = -\sum_{i=1}^{d} y_i \log \frac{\exp(o_i)}{\sum_{j=1}^{d}\exp(o_j)}$ is C-Lipchitz continuous for some constant $C > 0$.*

Proof. Note that

$$\mathcal{L}^{ce}(\mathbf{o};\mathbf{y}) = -\sum_{i=1}^{d} y_i o_i + \log\sum_{j=1}^{d}\exp(o_j), \tag{3.7}$$

where we have used the fact $\sum_{i=1}^{d} y_i = 1$ since $\mathbf{y}$ is a one-hot vector. The first term is linear in $\mathbf{o}$ and therefore satisfies the Lipchitz continuity. We now turn to verify the Lipchitz continuity of the function $\phi(\mathbf{o}) =$

[1] For regression problems, a similar theorem can be established as well.

$\log \sum_{i=1}^{d} \exp(o_i)$. According to the intermediate value theorem, for $\forall \mathbf{o}^S, \mathbf{o}^T \in R^d$, $\exists t \in [0,1]$ such that for $\bar{\mathbf{o}} = t\mathbf{o}^T + (1-t)\mathbf{o}^S$, we have

$$\begin{aligned} |\phi(\mathbf{o}^T) - \phi(\mathbf{o}^S)| &= |\nabla\phi(\bar{\mathbf{o}})^\top(\mathbf{o}^T - \mathbf{o}^S)| \leq \|\nabla\phi(\bar{\mathbf{o}})\|_1 \|\mathbf{o}^T - \mathbf{o}^S\|_\infty \\ &= \|\mathbf{o}^T - \mathbf{o}^S\|_\infty \leq C_0 \|\mathbf{o}^T - \mathbf{o}^S\|, \end{aligned} \tag{3.8}$$

where the third line comes from the Holder's inequality, the fourth line comes from the fact that $\nabla\phi(\bar{\mathbf{o}})$ is a softmax function lying on a simplex, and the last line is due to the equivalence among norms. With Equation (3.8), one can easily verify that

$$\begin{aligned} &|\mathcal{L}^{ce}(\mathbf{o}^T;\mathbf{y}) - \mathcal{L}^{ce}(\mathbf{o}^S;\mathbf{y})| = |\sum_{i=1}^{d} y_i(o_i^T - o_i^S) + \phi(\mathbf{o}^T) - \phi(\mathbf{o}^S)| \\ &\leq \|\mathbf{y}\|\|\mathbf{o}^T - \mathbf{o}^S\| + |\phi(\mathbf{o}^T) - \phi(\mathbf{o}^S)| \ \leq \ (C_0+1)\|\mathbf{o}^T - \mathbf{o}^S\| \\ &= (C_0+1)\|\mathbf{W}\mathbf{h}_L^T - \mathbf{W}\mathbf{h}_L^S\| \\ &\leq C\|\mathbf{h}_L^T - \mathbf{h}_L^S\| \end{aligned} \tag{3.9}$$

where we have used facts that $\|\mathbf{y}\| = 1$, $\mathbf{o}^T = \mathbf{W}\mathbf{h}_L^T$, $\mathbf{o}^S = \mathbf{W}\mathbf{h}_L^S$ with $\mathbf{W}$ as shared parameters of the last layer, and $C = (C_0+1)\cdot\|\mathbf{W}\|$. □

Lemma 2. *Suppose both $\mathcal{F}^T$ and $\mathcal{F}^S$ are activated by the Lipchitz-continuous function $\sigma(\cdot) = \mathrm{ReLU}(\cdot)$, the estimation error $\mathcal{L}_L^r$ at layer L can be bounded by the layerwise objective function $\tilde{\mathcal{L}}_l$ as follows:*

$$\mathcal{L}_L^r \leq \sum_{l=1}^{L-1} \prod_{k=l}^{L} C_k(\mu)\tilde{\mathcal{L}}_l + \tilde{\mathcal{L}}_L, \tag{3.10}$$

where $C_k(\mu)$ is some constant linear in μ in the k-th layer.

Proof. Recall that $\mathbf{h}_L^T = \sigma(\mathbf{W}_L^T * \mathbf{h}_{L-1}^T)$. To facilitate the following analysis, we apply the im2col operation to equivalently transform the convolution to matrix multiplication, i.e. $\bar{\mathbf{h}}_L^T = \sigma(\bar{\mathbf{W}}_L^T \bar{\mathbf{h}}_{L-1}^T)$, where $\bar{\mathbf{h}}_L^T \in R^{c_o \times (Nc_ikk)}$ and $\bar{\mathbf{W}}_L^T \in R^{c_o \times (c_ikk)}$ are matrices. Then

$$\begin{aligned} &\mathcal{L}_L^r = \|\mathbf{h}_L^T - \mathbf{h}_L^S\| = \|\bar{\mathbf{h}}_L^T - \bar{\mathbf{h}}_L^S\| \\ &= \|\sigma(\bar{\mathbf{W}}_L^T\bar{\mathbf{h}}_{L-1}^T) - \sigma(\bar{\mathbf{W}}_L^S\bar{\mathbf{h}}_{L-1}^S)\| \ \leq \ \|\bar{\mathbf{W}}_L^T\bar{\mathbf{h}}_{L-1}^T - \bar{\mathbf{W}}_L^S\bar{\mathbf{h}}_{L-1}^S\| \\ &= \|\bar{\mathbf{W}}_L^T\bar{\mathbf{h}}_{L-1}^T - \bar{\mathbf{W}}^S\bar{\mathbf{h}}_{L-1}^T + \bar{\mathbf{W}}^S\bar{\mathbf{h}}_{L-1}^T - \bar{\mathbf{W}}_L^S\bar{\mathbf{h}}_{L-1}^S\| \\ &\leq \mathcal{L}_{L-1}^c + \epsilon_{L-1}^S \ \leq \ \mathcal{L}_{L-1}^c + \|\bar{\mathbf{W}}_L^S\| \cdot \mathcal{L}_{L-1}^r, \end{aligned} \tag{3.11}$$

where the first inequality comes from the Lipchitz continuity of the $\mathrm{ReLU}(\cdot)$ function, and the rest can be readily obtained by applying the

triangle inequality. Similarly, we have

$$\mathcal{L}_L^r \leq \mathcal{L}_{L-1}^i + \|\bar{\mathbf{W}}_L^T\| \cdot \mathcal{L}_{L-1}^r \tag{3.12}$$

By taking the convex combination of Equation (3.11) and Equation (3.12) for some $\mu \in [0, 1]$, we have

$$\begin{aligned} \mathcal{L}_L^r &\leq \mu(\mathcal{L}_{L-1}^i + \|\bar{\mathbf{W}}_L^S\| \cdot \mathcal{L}_{L-1}^r) + (1-\mu)(\mathcal{L}_{L-1}^i + \|\bar{\mathbf{W}}_L^T\| \cdot \mathcal{L}_{L-1}^r) \\ &\leq \tilde{\mathcal{L}}_L + C_L(\mu)\mathcal{L}_{L-1}^r \;\leq\; \tilde{\mathcal{L}}_L + \sum_{l=1}^{L-1}\prod_{k=l}^{L} C_k(\mu)\tilde{\mathcal{L}}_l, \end{aligned} \tag{3.13}$$

where we define $C_k(\mu) = \mu\|\bar{\mathbf{W}}_k^S\| + (1-\mu)\|\bar{\mathbf{W}}_k^T\|$, and the last line is obtained recursively with $\mathcal{L}_0^r = \|\mathbf{x} - \mathbf{x}\| = 0$ at the network input of $\mathcal{F}^T$ and $\mathcal{F}^S$. □

Finally, by combining Equation (3.9) with Equation (3.13) together and define $C'(k) = C \cdot C(k)$, Equation (3.6) in Theorem 3.2.1 can be readily verified.

3.2.3 Soft Cross Distillation

Although the minimization of $\tilde{\mathcal{L}}$ is theoretically verified, the computation of $\tilde{\mathcal{L}}$ involves two loss terms with four convolutions to compute per batch of data, which doubles the training time. Here we propose another variant to balance $\mathcal{L}^c$ and $\mathcal{L}^i$ by empirically soften the hard connection of $\mathbf{h}^S$ and $\mathbf{h}^T$, as is shown in Figure 3.1(d). We define feature maps $\hat{\mathbf{h}}^T$ and $\hat{\mathbf{h}}^S$ after cross connection as the convex combination of $\mathbf{h}^T$ and $\mathbf{h}^S$, i.e.,

$$\begin{bmatrix} \hat{\mathbf{h}}^T \\ \hat{\mathbf{h}}^S \end{bmatrix} = \begin{bmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{bmatrix} \begin{bmatrix} \mathbf{h}^T \\ \mathbf{h}^S \end{bmatrix}, \tag{3.14}$$

where $\alpha, \beta \in [0, 1]$ are the hyper-parameters that adjust the percentage of cross connection. When $\alpha = 1$ and $\beta = 1$, this reduces to the estimation error in Equation (3.1). Similarly, correction and imitation correspond to $(\alpha, \beta) = (1, 0)$ and $(\alpha, \beta) = (0, 1)$ respectively. Meanwhile, α and β control the magnitude of inconsistencies ϵ^T and ϵ^S, allowing more flexibility for the cross connection.

Based on the assumption that $\|\mathbf{h}^T\| \approx \|\mathbf{h}^S\|$, the convex combination ensures the norm of input to be nearly identical after cross connection, and thus prevent magnitude propagation across layers. We define the

loss of soft cross distillation as

$$\hat{\mathcal{L}}(\mathbf{w}^S) = \|\sigma(\mathbf{w}^T * \hat{\mathbf{h}}^T) - \sigma(\mathbf{w}^S * \hat{\mathbf{h}}^S)\|_F^2, \tag{3.15}$$

which can substitute the estimation error $\mathcal{L}^e$ in Equation (3.1) as an alternative way for cross distillation.

3.2.4 Pruning via Proximal Operator

Given the formulation of cross distillation, we apply the proximal operator [165] to conduct network pruning. As previously introduced in Section 2.1.1, we choose $\mathcal{R} = \|\mathbf{w}^S\|_1 = \sum_{i,j,h,w} |W^S_{ijhw}|$ for unstructured pruning, and $\mathcal{R}(\mathbf{w}^S) = \|\mathbf{w}^S\|_{2,1} = \sum_i \|\mathbf{w}^S_i\|_2$, where $\mathbf{w}^S_i \in R^{c_o \times k \times k}$ for structured pruning (channel pruning) in Equation (3.1). Recall that the proximal gradient descent [165] iteratively update $\mathbf{w}^S$ by:

$$\mathbf{w}^S_{t+1} = \text{Prox}_{\lambda\mathcal{R}}(\mathbf{w}^S_t - \eta\nabla\tilde{\mathcal{L}}(\mathbf{w}^S_t)), \tag{3.16}$$

where $\text{Prox}_{\lambda\mathcal{R}}(u) = \arg\min_x \frac{1}{2}\|x-u\|_F^2 + \mathcal{R}(x)$ is the proximal operator for $\mathcal{R}$, tuned by λ. When $\mathcal{R}$ is chosen as $\|\cdot\|_1$, the proximal operator can be expressed as the soft-threshold determined by λ, i.e.,

$$\text{Prox}_{\lambda\|\cdot\|_1}(W^S_{ijhw}) = \begin{cases} W^S_{ijhw} - \lambda & W^S_{ijhw} > \lambda \\ 0 & |W^S_{ijhw}| \leq \lambda \\ W^S_{ijhw} + \lambda & W^S_{ijhw} < -\lambda \end{cases}. \tag{3.17}$$

For structured pruning, since $\mathcal{R} = \|\mathbf{w}^S\|_{2,1}$ is separable w.r.t. $\mathbf{w}^S_i$, the proximal operator for $\text{Prox}_{\lambda\|\cdot\|_{2,1}}(\mathbf{w}^S_i)$ can be computed as

$$\text{Prox}_{\lambda\|\cdot\|_2}(\mathbf{w}^S_i) = \max(1 - \frac{\lambda}{\|\mathbf{w}^S_i\|_2}, 0) \cdot \mathbf{w}^S_i, \tag{3.18}$$

and the solution to Equation (3.16) can be obtained group-wisely from Equation (3.18).

As suggested by previous works [39, 25], we linearly increase λ to smoothly prune the student network, which empirically gives better results. Given the maximum number of training steps T and the target sparsity ratio r assigned by users, we update λ by $\lambda_t = r + (1-r) * t/T$. An overall workflow of our proposed method is given in Algorithm 1.

Algorithm 1 Cross distillation for few-shot network pruning

Input:
The pre-trained teacher model $\mathcal{F}^T$
Training samples $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$
Target sparsity ratio r
Output:
The compact student model $\mathcal{F}^S$

```
for l = 1, ...L do
  for t = 1, ...T do
    Forward pass {x_n}_{n=1}^N to obtain h_{l-1}^T and h_{l-1}^S
    Calculate the loss in Equation (3.4) or (3.15)
    Update w_t^S with SGD/Adam optimizer
    Obtain w_{t+1}^S with Prox_{λR} in Equation (3.17) or (3.18)
    Increase the pruning threshold λ_t linearly
  end for
end for
```

3.3 Experiments

We conduct a series of experiments to verify the effectiveness of cross distillation for few-shot network compression. We take both channel pruning (i.e., structured pruning) and unstructured pruning for demonstration, both of which are popular approaches to reduce computational FLOPs and sizes of neural networks. To better understand the proposed method, we also provide further analysis on how cross distillation help reduces the estimation error against varying size of the training set. Our implementation in PyTorch is available at https://github.com/haolibai/Cross-Distillation.git.

3.3.1 Setup

Throughout the experiment, we use VGG [9] and ResNet [128] as base networks, and evaluations are performed on CIFAR-10 and ImageNet ILSVRC-12. As we consider the setting of few-shot image classification, we randomly select K-shot instances per class from the training set. All experiments are averaged over five runs with different random seeds, and results of means and standard deviations are reported [2].

Baselines and Implementations. For structured pruning, we compare our proposed methods against: 1) L1-norm pruning [81], a data-free approach; 2) Back-propagation (BP) based fine-tuning on L1-norm

[2] Note that for each run, we remove all the randomness such as data augmentation and data shuffling, and therefore the results are generally reproducible with the random seed fixed.

pruned models; 3) FitNet [33] and 4) FSKD [50], both of which are knowledge distillation methods; 5) ThiNet [26] and 6) Channel Pruning (CP) [25], both of which are layer-wise regression based channel pruning methods. We adopt the implementation[3] from [166] for 1) L1-Norm. Based on the pruned models by 1), we perform fine-tuning with back-propagation by minimizing the cross entropy or the FitNet loss, denoted as 2) BP and 3) FitNet respectively. For 4) ThiNet, our implementation is based on the published code[4]. For 5) CP, we re-implement the paper based on its TensorFlow version[5] and reproduce the results in Table 1 of the paper. For both ThiNet and CP, we use all feature map patches for regression instead of sampling a subset of them, the latter of which lead to a significant drop of accuracy when only limited training instances are available. For unstructured pruning, we modify 1) to element-wise L1-norm based pruning [39]. Besides, 4) FSKD, 5) ThiNet and 6) CP are removed since they are only applicable in channel pruning.

For our proposed method, we compare to three variants for ablation study: pruning without cross distillation by solving Equation 3.1 (abbr. w/o CD), cross distillation by solving Equation 3.4 (abbr. CD) and soft cross distillation by solving Equation 3.15 (abbr. SCD). For CD, we choose $\mu = 0.6$ for VGG networks and $\mu = 0.9$ for ResNets. For SCD, we set $(\alpha, \beta) = (0.9, 0.3)$ on VGG networks and $(0.9, 0.5)$ on ResNets. Sensitivity analysis on these hyper-parameters are presented later.

We follow the standard way [9, 128] in pre-training the model on CIFAR-10, and adopt the checkpoint from the official release of torchvision[6] for ILSVRC-12. Similar to [50], we do not adopt data augmentation so as to better simulate the few-shot setting. During the pruning, we adopt the ADAM optimizer for all these methods, and adjust the learning rate within [1e-5, 1e-3] to obtain proper performance. Each layer is optimized for 3,000 iterations, where the sparsity ratio linearly increases within the first 1,000 iterations. After layer-wise training, we further fine-tune the network for a few more epochs with back-propagation.

[3] https://github.com/Eric-mingjie/rethinking-network-pruning/tree/master/imagenet/l1-norm-pruning

[4] https://github.com/Roll920/ThiNet

[5] https://github.com/Tencent/PocketFlow

[6] https://pytorch.org/docs/stable/torchvision/models.html

Table 3.1: Structured pruning schemes of VGG-16 on CIFAR-10 and ResNet-34 on ILSVRC-12.

VGG-16					ResNet-34				
Schemes	Params (M)	P. (%) ↓	FLOPs (G)	F. (%) ↓	Schemes	Params (M)	P. (%) ↓	FLOPs (G)	F. (%) ↓
Orig.	14.99	-	0.314	-	Orig.	21.80	-	3.68	-
VGG-50%	4.53	69.78	0.082	73.95	Res-30%	19.71	9.59	2.97	19.15
VGG-A	6.11	59.26	0.208	33.76	Res-50%	18.33	15.91	2.51	31.47
VGG-B	4.37	70.83	0.137	56.37	Res-70%	16.92	22.37	2.05	44.26
VGG-C	2.92	80.55	0.061	80.45	Res-70%+	12.79	41.32	1.85	49.78

Table 3.2: The top-1 testing accuracy (%) of structured pruning with VGG-16 on CIFAR-10 with different training sizes. We use VGG-50% as the pruning scheme, and the original accuracy of the original model is 93.51%.

Methods	1	2	3	5	10	50
L1-norm	$14.36_{\pm 0.00}$	$14.36_{\pm 0.00}$	$14.36_{\pm 0.00}$	$14.36_{\pm 0.00}$	$14.36_{\pm 0.00}$	$14.36_{\pm 0.00}$
BP	$49.24_{\pm 1.76}$	$49.32_{\pm 1.88}$	$51.39_{\pm 1.53}$	$55.73_{\pm 1.19}$	$57.48_{\pm 0.91}$	$64.69_{\pm 0.43}$
FSKD	$47.91_{\pm 1.82}$	$55.44_{\pm 1.71}$	$61.76_{\pm 1.39}$	$65.69_{\pm 1.08}$	$72.20_{\pm 0.74}$	$75.46_{\pm 0.49}$
FitNet	$48.51_{\pm 2.51}$	$71.51_{\pm 2.03}$	$76.22_{\pm 1.95}$	$81.10_{\pm 1.13}$	$85.40_{\pm 1.02}$	$88.46_{\pm 0.76}$
ThiNet	$58.06_{\pm 1.71}$	$72.07_{\pm 1.68}$	$75.37_{\pm 1.59}$	$78.03_{\pm 1.24}$	$81.15_{\pm 0.85}$	$86.12_{\pm 0.45}$
CP	$66.03_{\pm 1.56}$	$75.23_{\pm 1.49}$	$77.98_{\pm 1.47}$	$81.53_{\pm 1.29}$	$83.59_{\pm 0.78}$	$87.27_{\pm 0.27}$
w/o CD	$65.57_{\pm 1.61}$	$75.44_{\pm 1.69}$	$78.40_{\pm 1.53}$	$81.20_{\pm 1.13}$	$84.07_{\pm 0.83}$	$87.67_{\pm 0.29}$
CD	$\mathbf{69.25_{\pm 1.39}}$	$\mathbf{80.65_{\pm 1.47}}$	$\mathbf{82.08_{\pm 1.41}}$	$\mathbf{84.91_{\pm 0.98}}$	$\mathbf{86.61_{\pm 0.71}}$	$87.64_{\pm 0.24}$
SCD	$68.53_{\pm 1.59}$	$76.83_{\pm 1.43}$	$80.16_{\pm 1.32}$	$84.28_{\pm 1.19}$	$86.30_{\pm 0.79}$	$\mathbf{88.65_{\pm 0.33}}$

Pruning Schemes. The structured pruning schemes are similar to those used in [81, 50]. For the VGG-16 network, we denote the three pruning schemes in [50] in the ascending order of sparsity as VGG-A, VGG-B and VGG-C respectively. We further prune 50% channels layer-wisely and denote the resulting scheme as VGG-50%. For ResNet-34, we remove r% channels in the middle layer of the first three residual blocks with some sensitive layers skipped (e.g., layer 2, 8, 14, 16). The last residual block is kept untouched. The resulting structured pruning schemes are denoted as Res-r%. Besides, we further remove 50% channels for the last block to reduce more FLOPs when $r = 70\%$, denoted as Res-70%+. The reduction of model sizes and computational FLOPs for structured pruned models are shown in Table 3.1.

In terms of unstructured pruning, we follow a similar pattern in [39] by removing $r = \{50\%, 70\%, 90\%, 95\%\}$ parameters for both the VGG network and ResNet, and each layer is treated equally.

3.3.2 Results

Structured Pruning. We evaluate structured pruning with VGG-16 on CIFAR-10 and ResNet-34 on ILSVRC-12. Table 3.2 and 3.3 shows the results with different number of training instances when the pruning

Table 3.3: The top-5 testing accuracy (%) of structured pruning with ResNet-34 on ILSVRC-12 with different training sizes. The first three columns use 50, 100 and 500 randomly sampled training instances, while the last three columns use $K = 1, 2, 3$ samples per class. We use Res-50% as the pruning scheme, and the top-1 and top-5 accuracies of the original model are 73.32% and 91.40%.

Methods	50	100	500	1	2	3
L1-norm	$72.94_{\pm 0.00}$	$72.94_{\pm 0.00}$	$72.94_{\pm 0.00}$	$72.94_{\pm 0.00}$	$72.94_{\pm 0.00}$	$72.94_{\pm 0.00}$
BP	$83.18_{\pm 1.86}$	$84.32_{\pm 1.29}$	$85.34_{\pm 0.89}$	$85.76_{\pm 0.73}$	$86.05_{\pm 0.51}$	$86.29_{\pm 0.56}$
FSKD	$82.53_{\pm 1.52}$	$84.58_{\pm 1.13}$	$86.67_{\pm 0.78}$	$87.08_{\pm 0.76}$	$87.23_{\pm 0.52}$	$87.20_{\pm 0.43}$
FitNet	$86.86_{\pm 1.81}$	$87.12_{\pm 1.63}$	$87.73_{\pm 0.96}$	$87.66_{\pm 0.84}$	$88.61_{\pm 0.76}$	$\mathbf{89.32_{\pm 0.78}}$
ThiNet	$85.67_{\pm 1.57}$	$85.54_{\pm 1.39}$	$86.97_{\pm 0.89}$	$87.42_{\pm 0.76}$	$87.52_{\pm 0.68}$	$87.53_{\pm 0.50}$
CP	$86.34_{\pm 1.24}$	$86.38_{\pm 1.37}$	$87.41_{\pm 0.80}$	$88.03_{\pm 0.66}$	$87.98_{\pm 0.49}$	$88.21_{\pm 0.37}$
w/o CD	$86.51_{\pm 1.71}$	$86.61_{\pm 1.20}$	$87.92_{\pm 0.75}$	$87.98_{\pm 0.60}$	$88.63_{\pm 0.49}$	$88.82_{\pm 0.38}$
CD	$86.95_{\pm 1.59}$	$87.60_{\pm 1.13}$	$88.34_{\pm 0.69}$	$88.17_{\pm 0.73}$	$88.57_{\pm 0.40}$	$88.59_{\pm 0.41}$
SCD	$\mathbf{87.42_{\pm 1.69}}$	$\mathbf{87.73_{\pm 1.17}}$	$\mathbf{88.60_{\pm 0.82}}$	$\mathbf{88.40_{\pm 0.61}}$	$\mathbf{88.84_{\pm 0.48}}$	$88.87_{\pm 0.35}$

schemes are fixed. It can be observed that both CD and SCD generally outperform the rest baselines on both networks, whereas CD enjoys a larger advantage on VGG-16 while SCD is superior on ResNet-34. Meanwhile, as the training size decreases, cross distillation brings more advantages comparing to the rest baselines, indicating that the layer-wise regression can benefit more from cross distillation when the student network over-fits more seriously on fewer training samples.

Next we fix the training size and change the pruning schemes. We keep $K = 5$ on CIFAR-10 and $K = 1$ on ILSVRC-12, and the results are listed in Table 3.4 and Table 3.5 respectively. Again on both datasets our proposed cross distillation performs consistently better compared to the rest approaches. Besides, the gain from cross distillation becomes larger as the sparsity of the student network increases (e.g., VGG-C and ResNet-70%+). We suspect that networks with sparser structures tend to suffer more from higher estimation errors, which poses more necessity for cross distillation to reduce the errors.

Unstructured Pruning. For unstructured pruning, here we present results of the VGG-16 network on ILSVRC-12 dataset. Similar to structured pruning, we first fix the pruning scheme and vary the training size, and the results are given in Table 3.6. It can be observed that both CD and SCD significantly outperform the rest methods. Comparing to structured pruning, cross distillation brings even more improvement in unstructured pruning. One reason to explain it could be the irregular sparsity of network parameters can better compensate the layer-wisely

Table 3.4: The top-1 testing accuracy (%) of different structured pruning schemes with VGG-16 on CIFAR-10. 10 samples per class are used.

Methods	VGG-50%	VGG-A	VGG-B	VGG-C
L1-norm	$14.36_{\pm 0.00}$	$88.32_{\pm 0.00}$	$32.87_{\pm 0.00}$	$10.00_{\pm 0.00}$
BP	$55.73_{\pm 1.19}$	$93.10_{\pm 0.09}$	$87.17_{\pm 0.49}$	$62.45_{\pm 1.25}$
FSKD	$65.69_{\pm 1.08}$	$93.52_{\pm 0.23}$	$90.69_{\pm 0.12}$	$81.79_{\pm 1.01}$
FitNet	$85.40_{\pm 1.02}$	$93.50_{\pm 0.06}$	$92.42_{\pm 0.32}$	$84.65_{\pm 1.53}$
ThiNet	$81.15_{\pm 0.85}$	$93.61_{\pm 0.05}$	$92.20_{\pm 0.16}$	$79.19_{\pm 0.91}$
CP	$83.59_{\pm 0.78}$	$\mathbf{93.70_{\pm 0.04}}$	$92.29_{\pm 0.15}$	$80.82_{\pm 0.73}$
w/o CD	$84.07_{\pm 0.83}$	$\mathbf{93.69_{\pm 0.07}}$	$92.35_{\pm 0.14}$	$83.90_{\pm 0.78}$
CD	$\mathbf{86.61_{\pm 0.71}}$	$\mathbf{93.65_{\pm 0.08}}$	$\mathbf{92.60_{\pm 0.11}}$	$\mathbf{85.81_{\pm 0.80}}$
SCD	$86.30_{\pm 0.79}$	$\mathbf{93.70_{\pm 0.07}}$	$\mathbf{92.68_{\pm 0.13}}$	$85.10_{\pm 0.75}$

Table 3.5: The top-5 testing accuracy (%) of different structured pruning schemes with ResNet-34 on ILSVRC-12. 1 sample per class is used.

Methods	Res-30%	Res-50%	Res-70%	Res-70%+
L1-norm	$84.54_{\pm 0.00}$	$72.94_{\pm 0.00}$	$31.84_{\pm 0.00}$	$15.30_{\pm 0.00}$
BP	$88.66_{\pm 0.59}$	$85.76_{\pm 0.73}$	$80.04_{\pm 0.90}$	$63.25_{\pm 1.05}$
FSKD	$89.56_{\pm 0.52}$	$87.08_{\pm 0.76}$	$80.82_{\pm 0.62}$	$67.04_{\pm 0.56}$
FitNet	$88.56_{\pm 0.58}$	$87.66_{\pm 0.84}$	$\mathbf{82.72_{\pm 0.88}}$	$68.31_{\pm 0.81}$
ThiNet	$89.74_{\pm 0.65}$	$87.42_{\pm 0.76}$	$79.40_{\pm 0.66}$	$63.65_{\pm 0.78}$
CP	$89.65_{\pm 0.78}$	$88.03_{\pm 0.66}$	$81.13_{\pm 0.85}$	$68.18_{\pm 0.79}$
w/o CD	$\mathbf{90.34_{\pm 0.53}}$	$87.98_{\pm 0.60}$	$82.11_{\pm 0.71}$	$69.03_{\pm 0.92}$
CD	$90.08_{\pm 0.47}$	$88.17_{\pm 0.65}$	$\mathbf{82.71_{\pm 0.76}}$	$\mathbf{73.53_{\pm 0.74}}$
SCD	$\mathbf{90.32_{\pm 0.58}}$	$\mathbf{88.40_{\pm 0.61}}$	$82.65_{\pm 0.68}$	$69.47_{\pm 0.79}$

accumulated errors on $\mathcal{F}^S$.

Similarly, we test our methods with different sparsities and hold the training size fixed as $K = 1$, and Table 3.7 shows the results. As the sparsity r increases, cross distillation brings more improvement, especially on VGG-95% with a nearly 10% and 14% increase of accuracy for CD and SCD respectively.

3.3.3 Further Analysis

The Estimation Errors v.s. Inconsistency. Cross distillation brings the inconsistencies ϵ^T, ϵ^S that could affect the reduction of estimation errors $\mathcal{L}^e$. To quantitatively investigate the effects, we compare ϵ^T, ϵ^S as well as $\mathcal{L}^e$ at different layers of the VGG-16 network on the test set of CIFAR-10. We take three student networks trained by the correction loss $\mathcal{L}^c$, the imitation loss $\mathcal{L}^i$ as well as soft distillation loss $\hat{\mathcal{L}}$ respectively We choose unstructured pruning with VGG-90% as the pruning scheme and vary K between $\{1, 10\}$, and the results are shown in 3.2(a), Figure 3.2(b) and Figure 3.2(c) respectively. Note that we have normalized the loss values by dividing the nonzero leftmost bar in each sub-figure.

Table 3.6: The top-5 testing accuracy (%) of unstructured pruning with VGG-16 on ILSVRC-12 with different training sizes. The first three columns use 50, 100 and 500 randomly sampled training instances, while the last three columns use $K = 1, 2, 3$ samples per class. We use Res-90% as the pruning scheme, and the top-1 and top-5 accuracies of the original model are 73.72% and 91.51%.

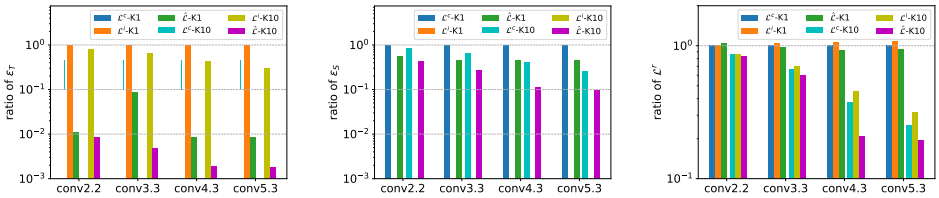
Methods	50	100	500	1	2	3
L1-norm	$0.53_{\pm 0.00}$	$0.53_{\pm 0.00}$	$0.53_{\pm 0.00}$	$0.53_{\pm 0.00}$	$0.53_{\pm 0.00}$	$0.53_{\pm 0.00}$
BP	$42.87_{\pm 2.07}$	$48.78_{\pm 1.43}$	$65.47_{\pm 1.15}$	$71.25_{\pm 0.97}$	$74.85_{\pm 0.71}$	$76.04_{\pm 0.48}$
FitNet	$52.66_{\pm 2.93}$	$57.09_{\pm 2.14}$	$76.59_{\pm 1.45}$	$80.14_{\pm 1.23}$	$82.27_{\pm 0.70}$	$83.14_{\pm 0.51}$
w/o CD	$78.73_{\pm 1.78}$	$83.29_{\pm 1.12}$	$85.04_{\pm 0.93}$	$85.36_{\pm 0.61}$	$85.21_{\pm 0.41}$	$85.49_{\pm 0.46}$
CD	$\mathbf{83.81_{\pm 1.49}}$	$86.21_{\pm 1.09}$	$87.19_{\pm 0.96}$	$87.61_{\pm 0.82}$	$87.78_{\pm 0.45}$	$87.86_{\pm 0.39}$
SCD	$83.67_{\pm 1.52}$	$\mathbf{86.72_{\pm 1.23}}$	$\mathbf{87.82_{\pm 1.04}}$	$\mathbf{88.14_{\pm 0.74}}$	$\mathbf{88.23_{\pm 0.61}}$	$\mathbf{88.38_{\pm 0.43}}$

Table 3.7: The top-5 testing accuracy (%) of unstructured pruning with VGG-16 on ILSVRC-12 with different pruning schemes. 1 sample per class is adopted.

Methods	VGG-50%	VGG-70%	VGG-90%	VGG-95%
L1-norm	$89.21_{\pm 0.00}$	$66.91_{\pm 0.00}$	$0.56_{\pm 0.00}$	$0.51_{\pm 0.00}$
BP	$90.61_{\pm 0.20}$	$88.08_{\pm 0.19}$	$71.25_{\pm 0.97}$	$42.37_{\pm 1.59}$
FitNet	$88.36_{\pm 0.46}$	$86.76_{\pm 0.67}$	$80.14_{\pm 1.23}$	$59.08_{\pm 1.78}$
w/o CD	$91.47_{\pm 0.12}$	$91.16_{\pm 0.10}$	$85.21_{\pm 0.41}$	$66.74_{\pm 1.36}$
CD	$91.58_{\pm 0.06}$	$91.24_{\pm 0.14}$	$87.61_{\pm 0.49}$	$76.65_{\pm 1.23}$
SCD	$\mathbf{91.68_{\pm 0.09}}$	$\mathbf{91.54_{\pm 0.11}}$	$\mathbf{88.14_{\pm 0.61}}$	$\mathbf{80.64_{\pm 1.03}}$

It can be observed that the student net trained by $\mathcal{L}^c$ has a large ϵ^S with $\epsilon^T = 0$, and vice versa for that trained by $\mathcal{L}^i$. On the contrary, the student net trained by $\hat{\mathcal{L}}$ shows both lower ϵ^T and ϵ^S, and the estimation error $\mathcal{L}^e$ is properly reduced as well. The results indicate that by properly controlling the magnitude of inconsistencies ϵ^T and ϵ^S with soft connection, cross distillation can indeed reduce estimation errors $\mathcal{L}^e$ and improve the student network.

Generalization Ability. One potential issue that troubles us is the generalization of cross distillation, since the training of CD and SCD is somehow biased comparing to w/o CD that directly minimizes the estimation error $\mathcal{L}^e$. Since estimation errors $\mathcal{L}^e$ among feature maps and cross entropy $\mathcal{L}^{ce}$ of logits directly reflect the closeness between $\mathcal{F}^T$ and $\mathcal{F}^S$ during inference, we compare both results among student nets obtained by w/o CD, CD and SCD respectively. We again take unstructured pruning with VGG-90% on the test set of CIFAR-10, and the rest settings are kept unchanged. For ease of comparison, we similarly divide values of CD and SCD by those obtained by w/o CD. Ratios



(a) ϵ^T. (b) ϵ^S. (c) $\mathcal{L}^e$.

Figure 3.2: The comparisons among inconsistencies ϵ^T, ϵ^S as well as estimation errors $\mathcal{L}^e$ on the test set of CIFAR-10. The colors denote what kind of loss and values of K are adopted for training. Best viewed in color.

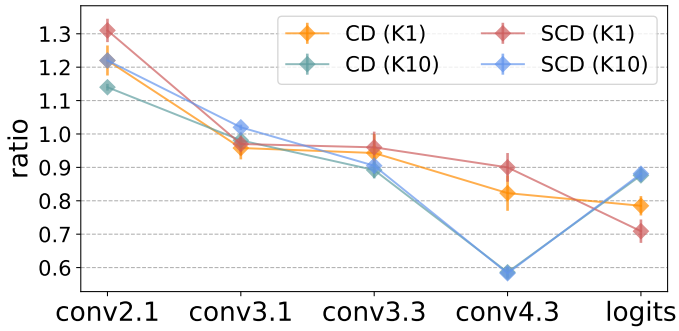


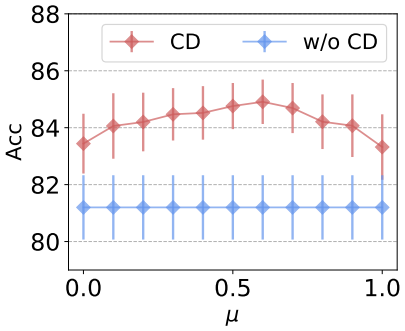
Figure 3.3: The estimation errors $\mathcal{L}^e$ of CD and SCD, both of which are divided by w/o CD. Best viewed in color.

smaller than 1 indicate a more generalizable student net.

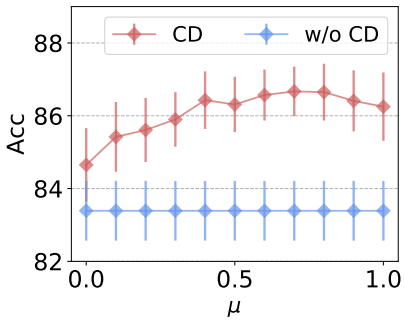
From Figure 3.3, we can find that while the ratios in shallower layers are above 1, they rapidly go down at deeper layers such as conv4.3 as well as the logits, which is consistent with Figure 3.3 that cross distillation tends to better benefit deeper layers. Moreover, although increasing K from 1 to 10 gives lower ratios of $\mathcal{L}^e$ at convolutional layers, the ratios of $\mathcal{L}^{ce}$ increases at the network logits, which lead to less improvement for classification when more training samples are available. The phenomenons are consistent with the results in Table 3.2, Table 3.3 and Table 3.6. In summary, cross distillation can indeed generalize well when $\mathcal{F}^T$ and $\mathcal{F}^S$ are properly mixed in the few-shot setting.

Sensitivity Analysis

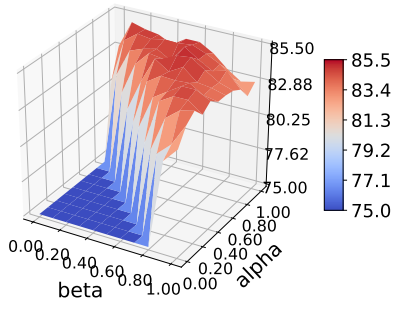
Finally, we present sensitivity analysis for cross distillation. We perform grid search by varying $\mu \in [0, 1]$ for CD and $(\alpha, \beta) \in [0, 1]^2$ for SCD at an interval of 0.1. We take VGG-16 for structured pruning and ResNet-56 for unstructured pruning on CIFAR-10 with $K = 5$, while ILSVRC-12 experiments adopt the same setting of μ and (α, β) found



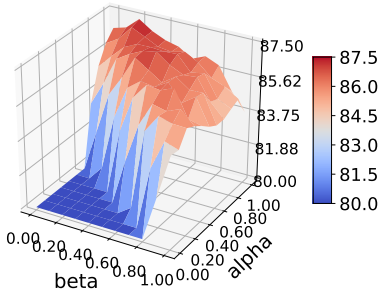
(a) CD on VGG-16.



(b) CD on ResNet-56.



(c) SCD on VGG-16.



(d) SCD on ResNet-56.

Figure 3.4: Sensitivity analysis of $\mu \in [0, 1]$ for CD and (α, β) on $[0, 1]^2$ for SCD.

by these experiments. From Figure 3.4(a) and 3.4(b), CD consistently outperforms w/o CD, where the best configurations appear at around $\mu = 0.6$ for VGG-16 and $\mu = 0.9$ for ResNet-56. Furthermore, we found that simply using the correction loss $\mu = 0.0$ or the imitation loss $\mu = 1.0$ also achieves reasonable results[7]. In terms of SCD in Figure 3.4(c) and 3.4(d) , we find that on left regions $\{(\alpha, \beta)|\alpha+\beta < 1\}$ $\mathcal{F}^T$ and $\mathcal{F}^S$ permute the input too much and thereon lead to significant drops of performance. For right regions $\{(\alpha, \beta)|\alpha + \beta > 1\}$, most configurations consistently outperform w/o CD $(1.0, 1.0)$, and the peaks occur somewhere in the middle of the regions.

3.4 Conclusion

In this chapter, we present cross distillation, a new layer-wise knowledge distillation approach for few-shot network pruning. Cross distillation consists of correction and imitation, where the former seeks to reduce the error propagation on the student network, while the latter makes

[7]The accuracies are 83.44% and 83.32% respectively on VGG-16, and 84.93% and 86.63% respectively on ResNet-56.

the student network better learn the behavior of the teacher given the same noise input. Cross distillation aims at a proper balance between correction and imitation so as to avoid over-fitting to the few-shot data, leading to a more generalizable pruned model. Extensive experiments and analysis demonstrate the superiority of our approach against various counterparts in few-shot network pruning. We believe the proposed approach can better benefit model compression challenged by data privacy and security issues.

□ **End of chapter.**

Chapter 4

Efficient Post-Training Quantization of Pre-trained Language Models

In this chapter, we study the problem of post-training quantization for pre-trained language models (PLMs). Network quantization has gained increasing attention with the rapid growth of large pre-trained language models (PLMs). However, most existing quantization methods for PLMs follow quantization-aware training (QAT) that requires end-to-end training with full access to the dataset. Therefore, they suffer from slow training, data security and huge memory consumption on large PLMs. We seek to mitigate these issues by post-training quantization (PTQ). Specifically, we propose module-wise quantization error minimization (MREM), an efficient solution for PTQ. By partitioning the PLM into multiple modules, we minimize the reconstruction error of each module separately. Meanwhile, each partitioned module can be trained locally on separate computing devices, which brings nearly the theoretical training speed-up. We conduct extensive experiments on prevalent PLMs over natural language understanding and reading comprehension tasks, both of which verify the improved efficiency and performance of our proposed solution.

4.1 Introduction

Large pre-trained language models (PLMs) have achieved remarkable success in various natural language processing tasks [121, 11, 17, 167, 168,

169]. However, the increasing size and computation overhead also make it prohibitive to deploy these PLMs on resource-constrained devices. To obtain compact PLMs, various model compression methods have been proposed, such as pruning [40, 170], knowledge distillation [171, 172, 51], weight-sharing [173, 174, 175, 176], dynamic computation with adaptive depth or width [41, 135, 136], and quantization [114, 177].

Among these methods, network quantization enjoys the reduction of both model size and computation overhead without modifying the network architecture and is thus extensively studied [114, 177, 6, 178]. However, despite their remarkable performance, these methods mostly follow quantization-aware training (QAT) and thus suffer from the following challenges: 1) QAT requires extensive training over the full dataset, which greatly prolongs the training time; 2) recent QAT methods [6, 178] further combine knowledge distillation to enhance the performance, which consumes even more memory with the presence of the teacher model, and thus makes the training of large PLMs prohibited on resource-limited devices; 3) QAT requires full access to the training set, which may give rise to data security issues when exposing them to third-party organizations for the quantization service.

Given the above challenges, post-training quantization (PTQ) serves as an appealing alternative. In contrast to QAT, PTQ is efficient in both training time and memory consumption. Usually, only a small portion of training data is required to calibrate the batch normalization statistics after convolutional layers [98] or clipping thresholds in quantization functions [117], i.e. re-estimating these values with the small amount of data. Nevertheless, PTQ is still not fully investigated in PLMs. Existing popular PTQ solutions [117, 119] are mostly developed for convolutional neural networks, by minimizing the layer-wise reconstruction error of convolutional layers or linear layers incurred by quantization. In the context of PLMs, however, as there are multiple linear layers in the multi-head self-attention and feed-forward network of the transformer model, sequentially tackling each of them may lead to sub-optimal solutions.

In this chapter, we propose ***module-wise reconstruction error minimization*** (MREM) to improve the performance of post-training quantization in PLM, while simultaneously maintain its efficiency w.r.t training time, memory consumption and data usage. Specifically, we partition the PLM into multiple modules, where each module consists of multiple Transformer layers. By minimizing the module-

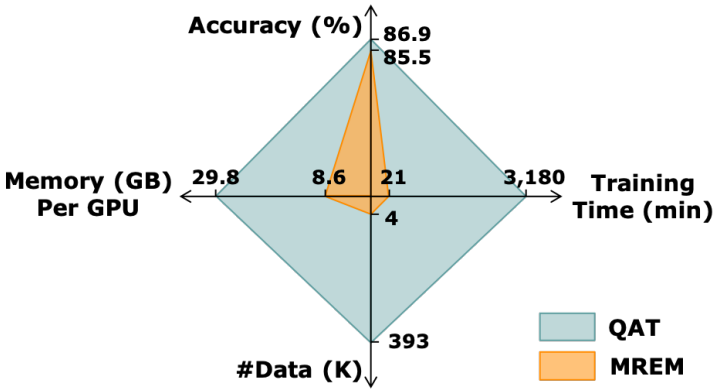
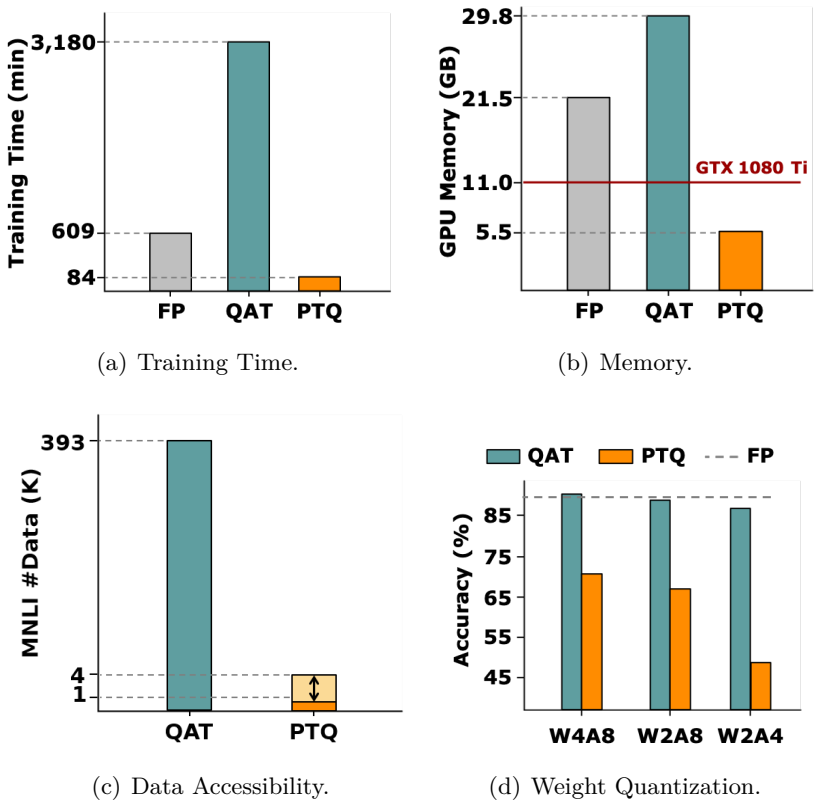


Figure 4.1: An illustrative comparison between our parallel post-training quantization method (MREM) and QAT on four dimensions. The results are based on a quantized BERT-large model with 4-bit weights and 8-bit activations over the MNLI dataset. Best viewed in color.

wise reconstruction error, we can jointly optimize multiple quantized components in a module. Meanwhile, the module granularity can also be flexibly adjusted depending on the memory constraints of computing devices. Similar block-wise objective has been previously considered in [179]. However, they require the second-order Hessian matrix which can be computationally prohibitive for large PLMs. Instead, to design a more efficient quantization pipeline for PLMs, we further propose a new ***model-parallel strategy*** based on model partition. By allocating each module on an individual computing device, all modules perform local training in parallel, achieving nearly the theoretical speed-up (e.g., 4× on 4 GPUs) compared with the sequential training [117, 119, 179]. Furthermore, we find that the naive parallel training suffers from reconstruction error propagation, since each quantized module passes this error to its successor before it is converged. Inspired by teacher forcing [180], we use the full-precision module to serve as a guide to the next quantized module, which provides clean input to break the reconstruction error propagation, and improves the quantization performance.

Empirical results on the GLUE and SQuAD benchmarks show that our proposed MREM not only significantly improves the performance for post-training quantization, but also enjoys advantages of fast training, light memory consumption and improved data security over QAT. For instance, as is shown in Figure 4.1, the BERT-large model trained by parallel MREM can achieve 85.5% accuracy based on only 4K training samples. Moreover, it consumes merely one-third of memory per GPU



(a) Training Time. (b) Memory.

(c) Data Accessibility. (d) Weight Quantization.

Figure 4.2: Comparison between QAT and PTQ over four dimensions. We use a BERT-large model over MNLI dataset for illustration. The full-precision (FP) fine-tuning is also included as a baseline. We follow the procedure in [6] for QAT, and REM in Equation (2.11) for PTQ. The training time and memory in (a) and (b) are measured by 4-bit weights and 8-bit activations (i.e., W4A8) on an NVIDIA V100.

and is more than **150**× faster than previous QAT training.

4.2 Motivation

In this section, we show that it is important yet challenging to do the post-training quantization of pre-trained language models. Before diving into details, we first recall the necessary backgrounds as introduced in Section 2.

4.2.1 Quantization Background

Network quantization replaces the original full-precision weight or activations $\mathbf{x} \in \mathbb{R}^{m \times n}$ with its lower-bit counterpart $\hat{\mathbf{x}}$. According to Equation (2.4), this can be achieved by $\hat{\mathbf{x}} = \mathcal{Q}_b(\mathbf{x}) = s \cdot \Pi_{\Omega(b)}(\mathbf{x}/s)$, where $\Omega(b) = \{-2^{b-1}, ..., 0, ..., 2^{b-1} - 1\}$ is the set of b-bit integers, and

$\Pi(\cdot)$ is the projection function that maps $\mathbf{x}/s$ to its closest integer.

In the context of Transformer quantization, we follow the default setting in previous works [114, 6]: we quantize both the network weights and activations in each matrix multiplication. We use symmetric uniform quantization for weights, embeddings, and activations, except activations after the self attention and GeLU function. For these two activations, we adopt asymmetric quantization since the elements involved are mostly positive. We skip the quantization for all layer-normalization layers, skip connections, biases and the last classification head due to limited computation overhead or large performance drop. Below we introduce two common branches in the quantization literature: quantization-aware training and post-training quantization.

As mentioned in Section 2.1.2, the training of quantized models can be generally divided into two branches: *quantization-aware training* (QAT) and *post-training quantization* (PTQ). Specifically, QAT requires the full training set $\mathcal{D}$ to conduct end-to-end training by Equation (2.10), which is usually time-consuming. Moreover, recent state-of-the-art QAT attempts on PLMs combine knowledge distillation to train the quantized model [6], which further burden the quantization pipeline. On the other hand, PTQ requires only a small subset of the training set $\tilde{\mathcal{D}} \subseteq \mathcal{D}$, and can be finished quickly. As mentioned in Section 2.1.2, reconstruction error minimization (REM) [117, 118, 119, 181] is a popular PTQ method. Here we repeat its objective function in Equation (2.11), which minimizes the distance between the multiplication output of the quantized and the full-precision counterpart as follows:

$$\min_{\mathbf{w},s} \|\hat{\mathbf{w}}^\top \hat{\mathbf{a}} - \mathbf{w}^\top \mathbf{a}\|^2, \quad \text{s.t. } \hat{\mathbf{w}} = \mathcal{Q}_b(\mathbf{w}), \tag{4.1}$$

where $\mathbf{w}$ and $\mathbf{a}$ are weights and activations, and $\hat{\mathbf{w}}$ and $\hat{\mathbf{a}}$ are their quantized representations. Such objective is solved layer by layer sequentially over the calibration dataset $\tilde{\mathcal{D}}$. In this work, we extend from REM for post-training quantization given its previous success.

4.2.2 Why Post-training Quantization?

In this section, we discuss the difference between PTQ and QAT along four dimensions of a quantization pipeline: 1) training time; 2) memory footprint; 3) data accessibility and 4) performance. According to Figure 4.2, we summarize the findings in the following paragraphs.

Training Time. As QAT iterates over the full training set $\mathcal{D}$ for multiple epochs, it is much more time-consuming than PTQ. Note that recent QAT methods [6, 178] further combine two-stage knowledge distillation [51], which even prolongs the training compared with the full-precision (FP) fine-tuning. As shown in Figure 4.2(a), QAT can take nearly four times longer than FP.

Memory Footprint. The increasing size of recent large PLMs makes it prohibited to conduct QAT on memory-limited computing resources. From Figure 4.2(b), QAT [6] even consumes 8.3GB more memory than FP when combined with knowledge distillation to store the full-precision teacher model. On the other hand, PTQ only caches intermediate results during the layer-wise REM in Equation (2.11), which can be fed into a single GTX 1080 Ti. Therefore, PTQ is also applicable on memory-limited computing devices.

Data Accessibility. The quantization service can be usually offered by some third-party organizations, where data security is always of high priority. As QAT requires access to the entire training set, it inevitably increases the risk of data exposure. PTQ, on the other hand, needs only a small amount of calibration data $\tilde{\mathcal{D}} \subseteq \mathcal{D}$, and can be easily constructed by randomly sampling 1K $\sim$ 4K instances from $\mathcal{D}$, as shown in Figure 4.2(c). Therefore, most original training instances are kept untouched and data security can be largely preserved.

Performance. When fine-tuned over the entire training set, QAT usually maintains better quantized performance than PTQ. From Figure 4.2(d), the performances of QAT are close to FP results, and remain steady across different bit-widths, i.e., W4A8, W2A8 and W2A4. However, the performances of PTQ drop significantly, which has been the main concern to address.

In summary, PTQ is superior to QAT with regard to training efficiency, memory consumption, and data accessibility. Nevertheless, it is still sometimes less preferred than QAT due to its severe performance drop especially for low quantization bit-width [114, 177, 6]. In this paper, we aim at improving the performance of post-training quantization for PLMs, while preserving its merits of fast training, light memory footprint, and data consumption.

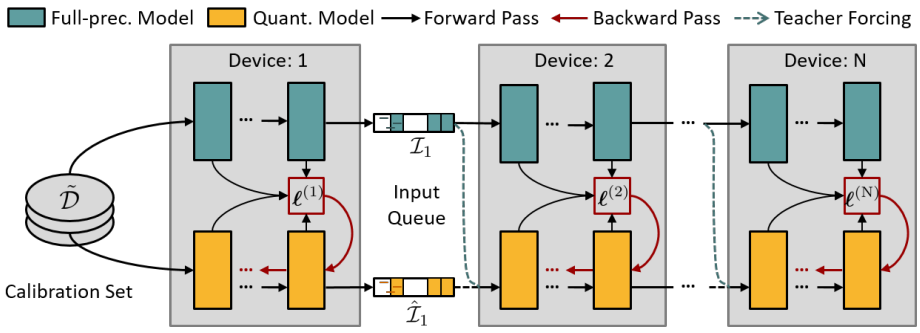


Figure 4.3: The overview of the proposed module-wise reconstruction error minimization (MREM). We partition both the full precision model and quantized model into multiple modules on different computing devices. By collecting tensors from the input queue, MREM can be conducted locally without waiting for the predecessor. Teacher forcing can be applied to mitigate the issue of reconstruction error propagation.

4.3 Methodology

In this section, we propose our solution to improve post-training quantization of large pre-trained language models. The proposed solution consists of three parts: we first extend the existing reconstruction error minimization to fit Transformer models in a module-wise granularity. Then we show that such module-wise minimization can be conducted in pipeline parallel, which further speeds up PTQ. Finally, we incorporate teacher enforcing as an additional trick to accelerate the convergence of parallel training. An overview of our solution can be found in Figure 4.3.

4.3.1 Module-wise Reconstruction Error Minimization

Given the transformer model, we propose *module-wise reconstruction error minimization* (MREM) for post-training quantization of pre-trained language models. As we will see in the following, the granularity of REM plays an important role in the performance and memory consumption for PTQ.

Existing REM [119] solves Equation (2.11) for each matrix multiplication, which is the minimal granularity in a network. However, a standard transformer layer consists of a Multi-Head Attention (MHA) and a Feed-Forward Network (FFN), both of which consist of multiple matrix multiplications. Greedily tackling each matrix multiplication in REM thus may lead to sub-optimal quantized solutions. Moreover, the reconstruction error shall propagate and enlarge along with transformer layers, and finally deteriorate the output [163].

Towards that end, the proposed module-wise reconstruction error minimization admits larger granularity by jointly optimizing all the coupled linear layers inside each module. Specifically, given a transformer model with L transformer layers, embedding layers and the classification head, we partition them into N modules, where the n-th module include $[l_n, l_{n+1})$ transformer layers with l_n being the first layer of this module[1]. MREM aims at minimizing the joint reconstruction errors between all quantized FFN output $\hat{\boldsymbol{f}}_l$ in the module from their full-precision counterpart $\boldsymbol{f}_l$ as follows:

$$\min_{\mathbf{w}_n, \mathbf{s}_n} \ell^{(n)} \triangleq \sum_{l \in [l_n, l_{n+1})} \|\hat{\boldsymbol{f}}_l - \boldsymbol{f}_l\|^2, \tag{4.2}$$

where $\mathbf{w}_n$ and $\mathbf{s}_n$ denote all learnable parameters and quantization step sizes within the n-th module. Similar to REM, MREM can be optimized sequentially: given the previously trained modules, only parameters and quantization step sizes in the current module are optimized. Besides the grouped Transformer layers, we also minimize the MSE loss in the Transformer embedding and output logits respectively

Note that the number of modules N can be adjusted depending on the memory constraint of computing resources. When $N = 1$, this reduces to intermediate-layer knowledge distillation [51], which can be memory-demanding when quantizing large PLMs on a single GPU. Meanwhile, it is also preferred to evenly partition the model so as to ensure a balanced memory footprint across different modules.

4.3.2 Accelerated Parallel Training

Based on the proposed MREM, we further propose a new model parallel strategy to further accelerate the training. As shown in Figure 6.2, we put different modules on individual computing devices. A set of **input queues** $\boldsymbol{\mathcal{I}} = \{\boldsymbol{\mathcal{I}}_1, ..., \boldsymbol{\mathcal{I}}_{N-1}\}$ is deployed between each pair of adjacent modules. For the n-th module, the queue collects its output of the most recent t_0 steps, i.e., $\boldsymbol{\mathcal{I}}_n^t = \{\boldsymbol{f}_{l_n}^t, \boldsymbol{f}_{l_n}^{t-1}, ..., \boldsymbol{f}_{l_n}^{t-t_0+1}\}$. Meanwhile, the $(n+1)$-th module can always sample with replacement $\boldsymbol{f}_{l_n} \sim \boldsymbol{\mathcal{I}}_n^t$ from the queue without waiting for the n-th module. Similar rules hold for the quantized module and their input queues $\hat{\boldsymbol{\mathcal{I}}}$ as well. The design of the input queue resembles *stale synchronous parallel* [182] which stores the stale parameters in a local cache so as to reduce the waiting time among

[1]Note that the embedding layers and the classification head are incorporated in the first and last module respectively.

workers, where t_0 is the stale threshold.

The training workflow is as follows. Initially, the modules are computed one after another for the first t_0 step to fill in the input queue, after which parallel training takes place. The module samples the input from the queue and calculates the loss $\ell^{(n)}$ correspondingly for $n = 1, ..., N$. In the meanwhile, the input queue is also updated with the rule of *first-in-first-out* throughout the training. In the backward pass, we constrain the gradients to propagate locally within each module, without affecting its predecessors. Such a design can avoid the load imbalance issue from straggler modules, bringing nearly the theoretical $N\times$ speed-up.

Annealed Teaching Forcing

Since all modules proceed with training simultaneously instead of the sequential manner, the next module takes the output from the queue before its predecessor is fully optimized. Therefore, the reconstruction error from the predecessor is propagated to the following modules before it is sufficiently minimized.

Inspired by teacher forcing [180] in training recurrent networks, the output $\boldsymbol{f}_{l_n}$ from the n-th full-precision module naturally serves as the clean input to the $(n+1)$-th quantized module to substitute $\hat{\boldsymbol{f}}_{l_n}$. Thus $\boldsymbol{f}_{l_n}$ stops the propagation of the reconstruction error accumulated on the quantized module. Nevertheless, such an approach breaks the connection to previous quantized modules and may suffer from forward inconsistency between training and inference [183] on the quantized model. To achieve a proper trade-off, we take the convex combination between the full-precision $\boldsymbol{f}_{l_n}$ and quantized $\hat{\boldsymbol{f}}_{l_n}$ as follows:

$$\tilde{\boldsymbol{f}}_{l_n} = \lambda \boldsymbol{f}_{l_n} + (1-\lambda)\hat{\boldsymbol{f}}_{l_n}, \quad \lambda \in [0,1], \tag{4.3}$$

where the hyper-parameter λ controls the strength of teacher forcing. $\lambda = 1$ gives the full correction of reconstruction error but with forward inconsistency, while $\lambda = 0$ reduces to the conventional setting that suffers from the propagated reconstruction error. We adopt a linear decay strategy for λ: $\lambda_t = \max(1 - \frac{t}{T_0},\ 0)$, where T_0 is the preset maximum steps of the decay. Intuitively, a large λ is desired at the beginning when each module is rarely optimized. Later, a small λ is preferred to transit to normal training such that the forward inconsistency can be bridged. The remaining $T - T_0$ steps stick to normal training without teacher forcing,

Algorithm 2 Efficient Post-training Quantization.

procedure MAIN :
Partition the PLM into N modules
Fill in the input queues $\boldsymbol{\mathcal{I}}$, $\hat{\boldsymbol{\mathcal{I}}}$
for n in 1, ..., N **do**
  ▷ *run in parallel*
  while $t < T$ **do**
    $\boldsymbol{f}_{l_{n-1}} \sim \boldsymbol{\mathcal{I}}^t_{n-1}$, $\hat{\boldsymbol{f}}_{l_{n-1}} \sim \hat{\boldsymbol{\mathcal{I}}}^t_{n-1}$
    $\boldsymbol{f}^t_{l_n}, \hat{\boldsymbol{f}}^t_{l_n} \leftarrow$ MREM $(\boldsymbol{f}_{l_{n-1}}, \hat{\boldsymbol{f}}_{l_{n-1}}, t)$
    Update $\boldsymbol{\mathcal{I}}^t_n$, $\hat{\boldsymbol{\mathcal{I}}}^t_n$ with $\boldsymbol{f}^t_{l_n}, \hat{\boldsymbol{f}}^t_{l_n}$
  end while
end for
return the Quantized PLM

Algorithm 3 Module-wise Reconstruction Error Minimization.

procedure MREM $(\boldsymbol{f}_{l_{n-1}}, \hat{\boldsymbol{f}}_{l_{n-1}}, t)$:
if $t < T_0$ **then**
  $\lambda_t \leftarrow \max(1 - \frac{t}{T_0},\ 0)$
  Compute $\tilde{\boldsymbol{f}}_{l_{n-1}}$ by Equation (4.3)
end if
Compute the full-precision module output $\boldsymbol{f}^t_{l_n}$
Compute the quantized module output $\hat{\boldsymbol{f}}^t_{l_n}$
Compute the loss $\ell^{(n)}$ by Equation (4.2)
$\mathbf{w}^{t+1}_n \leftarrow \text{Update}(\mathbf{w}^t_n, \frac{\partial \ell^{(n)}}{\partial \mathbf{w}^t_n}, \eta^t)$
$\mathbf{s}^{t+1}_n \leftarrow \text{Update}(\mathbf{s}^t_n, \frac{\partial \ell^{(n)}}{\partial \mathbf{s}^t_n}, \eta^t)$
return $\boldsymbol{f}^t_{l_n}, \hat{\boldsymbol{f}}^t_{l_n}$

so as to make each quantized module adapt to its own predecessors.

Comparison with Pipeline Parallelism.

Notably, our MREM with stale synchronous parallel is different from the recent pipeline parallel [184, 185]. Pipeline parallel adopts end-to-end training with synchronous updates between adjacent modules, which gives rise to bubble time on computing devices. While GPipe [184] divides the original data batch into M pipelined micro-batches, it still has the bubble time of $O(\frac{N-1}{N+M-1})$ under N partitions. On the one hand, a larger N or smaller M would increase the bubble time. On the other hand, a larger M leads to small batches that still cannot fully exploit the computing power, which again affects the acceleration rate. Differently, our parallel strategy conducts local training with stale synchronous updates of the input queue. Hence there is negligible bubble

time when the straggler is faster than the staleness threshold t_0, which can be easily satisfied with balanced module partitions or larger t_0.

Finally, an overview of the proposed parallel module-wise reconstruction error minimization is shown in Algorithm 2 and Algorithm 3. The Update$(\cdot)$ in Algorithm 3 can be any gradient update function such as AdamW [186] with learning rate η^t.

4.4 Experiments

In this section, we empirically verify our proposed module-wise knowledge distillation for post-training quantization of pre-trained language models. We first introduce the experimental setup in Section 4.4.1. Then we present main results in Section 4.4.2, including thorough comparisons with QAT and REM, as well as other existing quantization solutions. In Section 4.4.3, we provide more discussions on a variety of factors in our approach, such as the effect of teacher forcing, the number of model partitions and calibration data size.

4.4.1 Experimental Setup

Datasets and Metrics. We evaluate post-training quantization w.r.t. both text classification on the GLUE dataset [187], and reading comprehension on SQuAD benchmarks [188]. The size of calibration data is by default $|\tilde{\mathcal{D}}| = 4,096$, with instances randomly sampled from the full training set. As RTE and MRPC from GLUE contain less than 4,096 samples, we use the full training set for training. We leave the effect of data size in Section 4.4.3. For each experiment, we repeat ten times with different calibration sets. Both the mean and standard deviations are reported.

We use the same evaluation metrics as [11, 6]. Additionally, we report the size (MB), the inference FLOPs (G), as well as the training time. The computation of FLOPs follows [47, 32].

Implementation. We use the standardly fine-tuned BERT-base and BERT-large models[2] on downstream tasks for post-training quantization. We implement MREM in both the sequential training (abbv. MREM-S) in Section 4.3.1 and parallel training with teaching forcing (abbv.

[2]We follow the default fine-tuning hyper-parameter settings in Huggingface: https://github.com/huggingface/transformers.

MREM-P) in Section 4.3.2. For each module, we train for $2,000$ steps with an initial learning rate of $1e^{-4}$ on GLUE tasks, and $4,000$ steps with an initial learning rate of $5e^{-5}$ on SQuAD datasets. The learning rate follows linear decay as adopted in [11]. By default, we partition the model into 4 modules on 4 NVIDIA-V100 GPUs. The analysis of the training steps and the number of model partitions will be provided later.

For baselines, we mainly compare with QAT and REM, where the former acts as the upper bound of quantization performance, and the latter studies the granularity effect in PTQ training. We conduct QAT following the state-of-the-art training pipeline [6], i.e., intermediate-layer distillation followed by prediction-layer distillation, which takes 6 training epochs in total. Detailed hyper-parameter settings can be found in [6]. In terms of REM, we follow the practice in [119, 181] to minimize the reconstruction error after each matrix multiplication, as introduced in Section 4.2.1. For a fair comparison of each method, we use the same quantization scheme, i.e., TWN [95] or LAQ [104] for 2-bit and 4-bit weight quantization, and LSQ [31] for activation quantization. Unlike QAT that picks the best model based on the development set results, MREM is only tested once after training, which ensures data security of the development set. We leave the comparison with more existing quantization approaches in Section 4.4.2.

4.4.2 Main Results

Comparison with QAT and REM

We first compare MREM-S and MREM-P with QAT and REM on BERT-base and BERT-large models over MNLI and SQuAD benchmarks. The results are summarized in Table 4.1, Table 4.2 and Table 4.3 respectively. Following Section 4.2.2, we again summarize the results from the four dimensions:

Performance. It can be found that our proposed MREM-S improves the performance by REM significantly given the same training time, and is much more close QAT. For instance, according to MNLI results in Table 4.1, MREM-S with 4-bit weight quantization on BERT-base and BERT-large achieves the matched accuracy of $83.5\%_{\pm 0.1}$ and $86.1\%_{\pm 0.1}$, which is $10.2\% \uparrow$ and $16.1\% \uparrow$ better than REM, and only $1.1\% \downarrow$ and $0.8\% \downarrow$ inferior to QAT respectively. It is also found that BERT-base even outperforms BERT-large on MNLI. We speculate that the matrix-

wise granularity in REM may lead to worse solutions with more layers in the network.

Moreover, when all modules are simultaneously trained, MREM-P is generally close or only slightly inferior to MREM-S. From SQuAD 1.1 results in Table 4.2, MREM-P can even outperform MREM-S with the "W2-E2-A4" quantized BERT-large model (i.e., the EM score and F1 score are on average 0.4% ↑ and 0.2% ↑ respectively).

Training Time. Our proposed MREM also enjoys significantly less training time than QAT. On MNLI, for instance, BERT-base only takes 36 minutes for the 4-bit quantized PTQ training, which is about 36× faster than QAT and 6× faster than full-precision fine-tuning. Comparing with REM, the training pipeline of MREM is also simpler, since it does not need to cache the output after every matrix multiplication as done in REM. We shall further discuss this in Section 4.4.3. Moreover, when armed with the proposed parallel training, MREM-P is further **4**× faster than MREM-S, which achieves the theoretical speed up with four computing devices. These together bring a total **144**× saving of training time than QAT on MNLI.

Memory Consumption. While the module-wise training inevitably consumes more memory than REM, it still takes only around a third of the memory by QAT, and a half of that by the full-precision fine-tuning. For instance, while QAT takes 29.8GB memory on BERT-large, MREM only consumes **10.8**GB memory, which can be even fed into a cheap NVIDIA GTX 1080 Ti. Moreover, for long sequence input (i.e., 384 tokens on the SQuAD dataset), QAT over BERT-large may suffer from memory overflow even on an NVIDIA V100 with 32GB memory. QAT with gradient accumulation inevitably doubles the training time under the same configuration (i.e., underlined figures ("__") in Table 4.2 and Table 4.3). On the other hand, such issues can be easily mitigated in either REM or our MREM.

Table 4.1: The main results of our proposed MREM-S and MREM-P against QAT on the MNLI dataset. "#Bits (W-E-A)" represents the bit number for weights of Transformer layers, word embedding, and activations. Acc-m and Acc-mm denotes the validation accuracy on the matched and mismatched sections of MNLI respectively.

	#Bits (W-E-A)	Quant Method	BERT-base					BERT-large				
			Time (min)	Mem (GB)	# Data (K)	Acc m(%)	Acc mm(%)	Time (min)	Mem (GB)	# Data (K)	Acc m(%)	Acc mm(%)
MNLI	*full-prec*	N/A	220	8.6	393	84.5	84.9	609	21.5	393	86.7	85.9
	4-4-8	QAT	1,320	11.9	393	84.6	84.9	3,180	29.8	393	86.9	86.7
		REM	28	2.5	4	$73.3_{\pm 0.3}$	$74.9_{\pm 0.2}$	84	5.5	4	$70.0_{\pm 0.4}$	$71.8_{\pm 0.3}$
		MREM-S	36	4.6	4	$83.5_{\pm 0.1}$	$83.9_{\pm 0.1}$	84	10.8	4	$86.1_{\pm 0.1}$	$85.9_{\pm 0.1}$
		MREM-P	9	$3.7_{\times 4}$	4	$83.4_{\pm 0.1}$	$83.7_{\pm 0.1}$	21	$8.6_{\times 4}$	4	$85.5_{\pm 0.1}$	$85.4_{\pm 0.2}$
	2-2-8	QAT	882	11.9	393	84.4	84.6	2,340	29.8	393	86.5	86.1
		REM	24	2.5	4	$71.6_{\pm 0.4}$	$73.4_{\pm 0.4}$	64	5.5	4	$66.9_{\pm 0.4}$	$68.6_{\pm 0.7}$
		MREM-S	24	4.6	4	$82.7_{\pm 0.2}$	$82.7_{\pm 0.2}$	64	10.8	4	$85.4_{\pm 0.2}$	$85.3_{\pm 0.2}$
		MREM-P	6	$3.7_{\times 4}$	4	$82.3_{\pm 0.2}$	$82.6_{\pm 0.2}$	16	$8.6_{\times 4}$	4	$84.6_{\pm 0.2}$	$84.6_{\pm 0.1}$
	2-2-4	QAT	875	11.9	393	83.5	84.2	2,280	29.8	393	85.8	85.9
		REM	24	2.5	4	$58.3_{\pm 0.5}$	$60.6_{\pm 0.6}$	64	5.5	4	$48.8_{\pm 0.6}$	$51.4_{\pm 0.8}$
		MREM-S	24	4.6	4	$81.1_{\pm 0.2}$	$81.5_{\pm 0.2}$	64	10.8	4	$83.6_{\pm 0.2}$	$83.7_{\pm 0.2}$
		MREM-P	6	$3.7_{\times 4}$	4	$80.8_{\pm 0.2}$	$81.2_{\pm 0.2}$	16	$8.6_{\times 4}$	4	$83.0_{\pm 0.3}$	$83.2_{\pm 0.2}$

Table 4.2: The main results of our proposed MREM-S and MREM-P against QAT on SQuAD v1.1 dataset. "—" denotes results with two gradient accumulation steps under the same batch size due to memory constraint.

SQuAD v1.1	#Bits (W-E-A)	Quant Method	BERT-base					BERT-large				
			Time (min)	Mem (GB)	# Data (K)	EM (%)	F1 (%)	Time (min)	Mem (GB)	# Data (K)	EM (%)	F1 (%)
	full-prec	-	177	11.7	88	81.5	88.7	488	30.4	88	86.9	93.1
	4-4-8	QAT	428	18.4	88	80.2	87.9	1,920	27.0	88	86.7	93.0
		REM	65	3.1	4	$46.1_{\pm 0.5}$	$60.0_{\pm 0.5}$	175	7.3	4	$68.3_{\pm 0.1}$	$79.3_{\pm 0.1}$
		MREM-S	76	6.4	4	$79.4_{\pm 0.1}$	$87.2_{\pm 0.1}$	200	14.5	4	$86.2_{\pm 0.1}$	$92.5_{\pm 0.1}$
		MREM-P	19	$5.5_{\times 4}$	4	$79.6_{\pm 0.1}$	$87.3_{\pm 0.1}$	50	$12.3_{\times 4}$	4	$86.0_{\pm 0.1}$	$92.4_{\pm 0.1}$
	2-2-8	QAT	335	18.4	88	79.3	87.2	1,200	27.0	88	86.1	92.5
		REM	60	3.1	4	$40.1_{\pm 0.4}$	$55.0_{\pm 0.4}$	160	7.3	4	$66.4_{\pm 0.5}$	$77.7_{\pm 0.3}$
		MREM-S	60	6.4	4	$77.8_{\pm 0.2}$	$86.0_{\pm 0.1}$	156	14.5	4	$85.4_{\pm 0.1}$	$91.9_{\pm 0.1}$
		MREM-P	15	$5.5_{\times 4}$	4	$77.7_{\pm 0.2}$	$85.9_{\pm 0.2}$	39	$12.3_{\times 4}$	4	$85.3_{\pm 0.2}$	$91.8_{\pm 0.1}$
	2-2-4	QAT	331	18.4	88	77.1	85.9	1,186	27.0	88	84.7	93.1
		REM	60	3.1	4	$10.4_{\pm 0.2}$	$24.6_{\pm 0.2}$	160	7.3	4	$28.3_{\pm 0.6}$	$45.0_{\pm 0.5}$
		MREM-S	60	6.4	4	$72.7_{\pm 0.2}$	$82.5_{\pm 0.2}$	156	14.5	4	$81.4_{\pm 0.3}$	$89.4_{\pm 0.2}$
		MREM-P	15	$5.5_{\times 4}$	4	$73.0_{\pm 0.3}$	$82.7_{\pm 0.2}$	39	$12.3_{\times 4}$	4	$81.8_{\pm 0.3}$	$89.6_{\pm 0.2}$

Table 4.3: The main results of our proposed MREM-S and MREM-P against QAT on SQuAD v2.0 dataset. "__" denotes results with two gradient accumulation steps under the same batch size due to memory constraint.

	#Bits (W-E-A)	Quant Method	BERT-base					BERT-large				
			Time (min)	Mem (GB)	# Data (K)	EM (%)	F1 (%)	Time (min)	Mem (GB)	# Data (K)	EM (%)	F1 (%)
SQuAD v2.0	*full-prec*	-	255	11.7	130	74.5	77.7	730	30.4	130	77.7	81.0
	4-4-8	QAT	662	18.4	130	74.4	77.5	2,820	28.3	130	77.4	80.5
		REM	60	3.1	4	$53.1_{\pm 0.4}$	$53.6_{\pm 0.4}$	175	7.3	4	$58.2_{\pm 0.2}$	$61.4_{\pm 0.3}$
		MREM-S	76	6.4	4	$73.0_{\pm 0.1}$	$76.3_{\pm 0.1}$	200	14.5	4	$76.4_{\pm 0.1}$	$79.7_{\pm 0.1}$
		MREM-P	19	$5.5_{\times 4}$	4	$72.6_{\pm 0.2}$	$75.9_{\pm 0.2}$	50	$12.3_{\times 4}$	4	$76.3_{\pm 0.1}$	$79.6_{\pm 0.1}$
	2-2-8	QAT	508	17.5	130	73.0	76.2	1,680	28.3	130	76.7	80.0
		REM	60	3.1	4	$51.5_{\pm 0.2}$	$51.8_{\pm 0.2}$	160	7.3	4	$56.3_{\pm 0.2}$	$59.5_{\pm 0.2}$
		MREM-S	60	6.4	4	$71.4_{\pm 0.2}$	$74.8_{\pm 0.2}$	156	14.5	4	$75.4_{\pm 0.2}$	$78.7_{\pm 0.1}$
		MREM-P	15	$5.5_{\times 4}$	4	$70.8_{\pm 0.4}$	$74.3_{\pm 0.4}$	39	$12.3_{\times 4}$	4	$75.3_{\pm 0.3}$	$78.6_{\pm 0.3}$
	2-2-4	QAT	505	17.5	130	71.4	74.6	1,655	28.3	130	75.4	78.9
		REM	60	3.1	4	$39.3_{\pm 1.5}$	$41.4_{\pm 1.3}$	160	7.3	4	$42.9_{\pm 0.8}$	$44.2_{\pm 0.7}$
		MREM-S	60	6.4	4	$67.2_{\pm 0.3}$	$70.6_{\pm 0.2}$	156	14.5	4	$71.3_{\pm 0.3}$	$74.8_{\pm 0.2}$
		MREM-P	15	$5.5_{\times 4}$	4	$66.1_{\pm 0.5}$	$69.8_{\pm 0.5}$	39	$12.3_{\times 4}$	4	$71.5_{\pm 0.3}$	$75.0_{\pm 0.3}$

Data Size. REM and our proposed MREM follow the common practice of PTQ, relying on only 4,096 randomly sampled training instances on both MNLI and SQuAD, which is a tiny fraction of the original dataset used in QAT. We shall provide more discussion on the effect of calibration size in Section 4.4.3.

In summary, our proposed MREM-S improves post-training quantization on PLMs significantly, while still enjoys fast training, light memory consumption, and data security. Furthermore, with parallel training, the proposed MREM-P further pushes the advantages of PTQ without an apparent performance drop.

Comparison with Existing Methods

In the next, we compare our MREM with a number of existing state-of-the-art BERT quantization methods. They include various QAT approaches such as Q-BERT [177], Quant-Noise [189] and TernaryBERT [6], as well as the PTQ baseline GOBO [116]. Their results are taken from the original papers, respectively.

From Table 4.4, both our proposed MREM-S and MREM-P outperform existing PTQ approaches in most cases, and even achieves results close to QAT approaches. For example, the "W4-E4-A8" quantized MREM-S and MREM-P have the averaged accuracies of 83.5% and 83.4% on MNLI respectively, both of which are on par with "W2/4-E8-A8" quantized Q-BERT. In terms of the "W2-E2-A8" quantized models, our MREM-S and MREM-P surpass GOBO by **11.7**% ↑ and **11.3**% ↑ on MNLI-m respectively.

Table 4.4: Results on the GLUE development set. "PTQ" indicates whether the approach belongs to post-training quantization. "Avg." denotes the average results of all tasks.

Quant Method	**#Bits (W-E-A)**	**Size (MB)**	**PTQ**	**MNLI-m**	**QQP**	**QNLI**	**SST-2**	**CoLA**	**STS-B**	**MRPC**	**RTE**	**Avg.**
-	*full-prec.*	418	-	84.9	91.4	92.1	93.2	59.7	90.1	86.3	72.2	83.9
Q-BERT	2-8-8	43	✗	76.6	-	-	84.6	-	-	-	-	-
Q-BERT	2/4-8-8	53	✗	83.5	-	-	92.6	-	-	-	-	-
Quant-Noise	PQ	38	✗	83.6	-	-	-	-	-	-	-	-
TernaryBERT	2-2-8	28	✗	83.3	90.1	91.1	92.8	55.7	87.9	87.5	72.9	82.7
GOBO	3-4-32	43	✓	83.7	-	-	-	-	88.3	-	-	-
GOBO	2-2-32	28	✓	71.0	-	-	-	-	82.7	-	-	-
MREM-S	4-4-8	50	✓	$83.5_{\pm 0.1}$	$90.2_{\pm 0.1}$	$91.2_{\pm 0.1}$	$91.4_{\pm 0.4}$	$55.1_{\pm 0.8}$	$89.1_{\pm 0.1}$	$84.8_{\pm 0.0}$	$71.8_{\pm 0.0}$	$82.4_{\pm 0.1}$
	2-2-8	28	✓	$82.7_{\pm 0.2}$	$89.6_{\pm 0.1}$	$90.3_{\pm 0.2}$	$91.2_{\pm 0.4}$	$52.3_{\pm 1.0}$	$88.7_{\pm 0.1}$	$86.0_{\pm 0.0}$	$71.1_{\pm 0.0}$	$81.5_{\pm 0.2}$
MREM-P	4-4-8	50	✓	$83.4_{\pm 0.1}$	$90.2_{\pm 0.1}$	$91.0_{\pm 0.2}$	$91.5_{\pm 0.4}$	$54.7_{\pm 0.9}$	$89.1_{\pm 0.1}$	$86.3_{\pm 0.0}$	$71.1_{\pm 0.0}$	$82.2_{\pm 0.1}$
	2-2-8	28	✓	$82.3_{\pm 0.2}$	$89.4_{\pm 0.1}$	$90.3_{\pm 0.2}$	$91.3_{\pm 0.4}$	$52.9_{\pm 1.2}$	$88.3_{\pm 0.2}$	$85.8_{\pm 0.0}$	$72.9_{\pm 0.0}$	$81.6_{\pm 0.2}$

Table 4.5: Ablation studies for teacher forcing on BERT-base and BERT-large over the MNLI dataset. We report the matched accuracy with different training steps.

#Bits (W-E-A)	# Steps	BERT-base		BERT-large	
		w/o TF	w TF	w/o TF	w TF
2-2-8	250	$79.6_{\pm 0.3}$	$80.7_{\pm 0.2}$	$82.1_{\pm 0.4}$	$83.1_{\pm 0.2}$
	500	$81.0_{\pm 0.3}$	$81.6_{\pm 0.2}$	$83.4_{\pm 0.3}$	$84.1_{\pm 0.3}$
	2,000	$82.2_{\pm 0.2}$	$82.7_{\pm 0.2}$	$84.3_{\pm 0.3}$	$84.6_{\pm 0.2}$
	4,000	$82.3_{\pm 0.3}$	$82.5_{\pm 0.2}$	$84.5_{\pm 0.2}$	$84.7_{\pm 0.2}$
2-2-4	250	$73.9_{\pm 0.5}$	$77.3_{\pm 0.4}$	$76.5_{\pm 0.9}$	$79.3_{\pm 0.4}$
	500	$77.9_{\pm 0.2}$	$79.0_{\pm 0.2}$	$80.0_{\pm 0.5}$	$81.4_{\pm 0.2}$
	2,000	$80.4_{\pm 0.2}$	$80.8_{\pm 0.2}$	$82.5_{\pm 0.4}$	$83.0_{\pm 0.3}$
	4,000	$80.7_{\pm 0.2}$	$81.0_{\pm 0.2}$	$83.1_{\pm 0.1}$	$83.3_{\pm 0.3}$

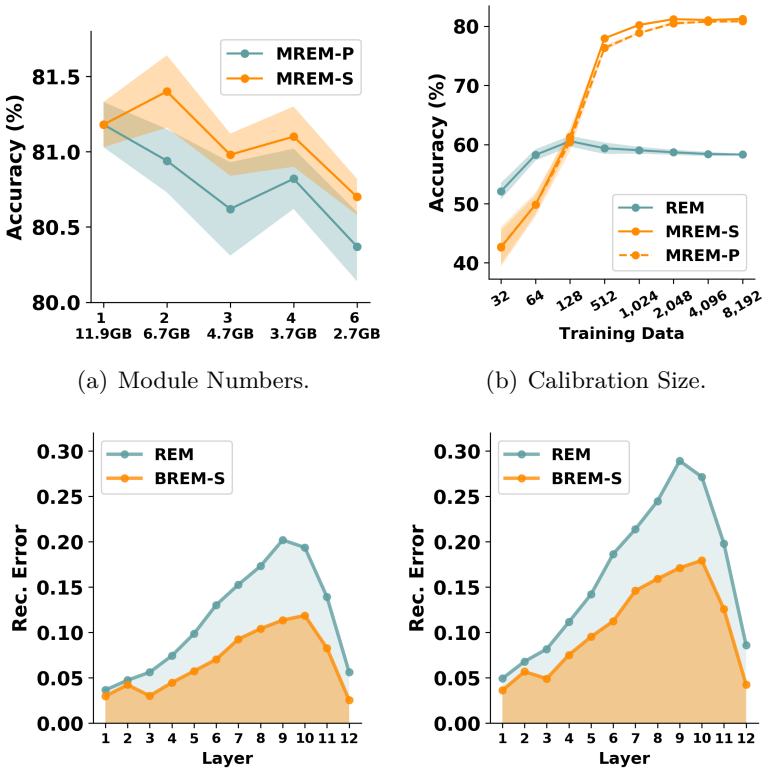
4.4.3 Discussions

In this section, we provide further discussions to better understand the proposed approach. By default, all experiments in this section are based on the BERT-base model over MNLI dataset.

Teacher Forcing

We now study how teacher forcing benefits MREM-P with different training steps, and results are listed in Table 4.5. It can be found that teacher forcing can bring consistent improvement for both BERT-base and BERT-large models. Moreover, the gain of teacher forcing is more significant with fewer training steps or lower quantization bit-width, i.e., **3.4**% ↑ and **2.8**% ↑ on the "W2-E2-A4" quantized BERT-base and BERT-large respectively. This matches our intuition that fewer training steps or higher compression ratio give larger reconstruction error, when the clean input from the full-precision model can benefit more. As the increase of training steps brings only marginal improvement and diminishes the effect of teacher forcing, we by default set the training steps to $2,000$.

Additionally, we also plot training loss curves of the four modules under 250 and 2,000 training steps in Figure 4.5. We find that: 1) loss curves with teacher forcing is apparently lower, especially when trained with fewer steps, which matches the observations in Table 4.5; 2) the modules close to the end tend to benefit more from teacher forcing, since the propagated reconstruction error can be better corrected by the clean input from the full-precision model.



(a) Module Numbers. (b) Calibration Size.

(c) Rec. Error Propagation (A8). (d) Rec. Error Propagation (A4).

Figure 4.4: Discussions on the proposed MREM approach. In (a) and (b), the solid line and shaded area denote the averaged results and standard deviation of a "W2-E2-A4" quantized BERT-base model from 10 different seeds. (c) and (d) visualize the propagation of reconstruction error on both "W2-E2-A8" and "W2-E2-A4" quantized BERT-base model.

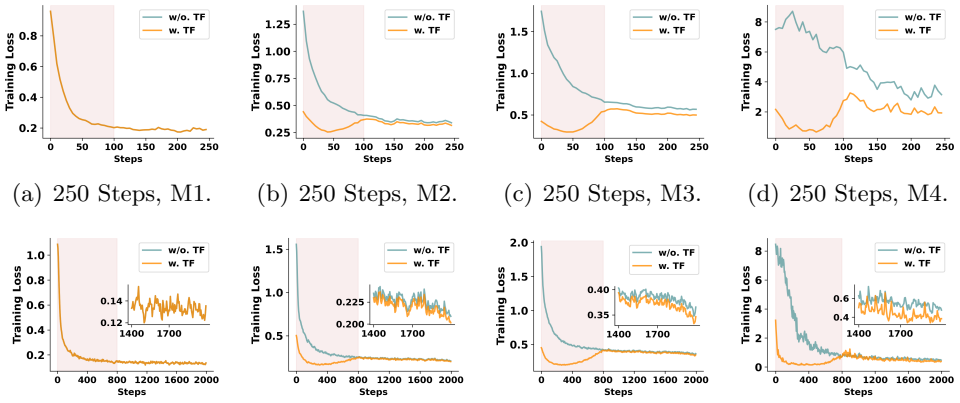
Further Comparisons with REM

Here we provide further discussions with REM on the training efficiency. Note that as both REM and MREM-S follow the sequential training procedure, i.e., the intermediate results after training of the current stage should be cached and reloaded for the next stage. However, as there are amounts of matrix multiplications in the transformer, such a procedure for REM can be time-consuming. While REM and MREM take roughly the same amount of time according to results in Section 4.4.2, REM is only iterated for 250 steps on MNLI and 500 on SQuAD, while MREM takes $2,000$ steps and $4,000$ steps respectively.

We also provide results when REM takes the same amount of training steps with MREM-S in Table 4.6. It can be found that even with $2,000$ iterations, REM is still inferior to MREM-S across all quantization bit-widths. Meanwhile, REM nearly takes around $9\times$ more training

#Bits (W-E-A)	Quant (Method)	# Steps	Time (min)	Mem (GB)	Acc m(%)	Acc mm(%)
4-4-8	REM	200	36	2.5	$73.3_{\pm 0.3}$	$74.9_{\pm 0.2}$
	REM	2,000	319	2.5	$81.8_{\pm 0.2}$	$82.5_{\pm 0.1}$
	MREM-S	2,000	36	4.6	$83.5_{\pm 0.1}$	$83.9_{\pm 0.1}$
2-2-8	REM	200	24	2.5	$71.6_{\pm 0.4}$	$73.4_{\pm 0.4}$
	REM	2,000	213	2.5	$78.7_{\pm 0.2}$	$79.2_{\pm 0.2}$
	MREM-S	2,000	24	4.6	$82.7_{\pm 0.2}$	$82.7_{\pm 0.2}$
2-2-4	REM	200	24	2.5	$58.3_{\pm 0.5}$	$60.6_{\pm 0.6}$
	REM	2,000	213	2.5	$73.0_{\pm 0.3}$	$74.4_{\pm 0.4}$
	MREM-S	2,000	24	4.6	$81.1_{\pm 0.2}$	$81.5_{\pm 0.2}$

Table 4.6: Comparison of REM with our MREM on BERT-base model over the MNLI dataset.



(a) 250 Steps, M1. (b) 250 Steps, M2. (c) 250 Steps, M3. (d) 250 Steps, M4.

(e) 2,000 Steps, M1. (f) 2,000 Steps, M2. (g) 2,000 Steps, M3. (h) 2,000 Steps, M4.

Figure 4.5: The training loss curves with and without teacher forcing (TF) in MREM-P. The red area denotes the warm-up stage in the first 40% training steps. (a), (b), (c) and (d) in the first row are the four modules (i.e., M1-M4) trained for 250 steps, and (e), (f), (g) and (h) in the second row are trained for 2,000 steps.

time than MREM. Therefore, the module-wise granularity in MREM not only improves the quantization performance with more layer-wise dependencies considered, but also makes the training pipeline efficient with fewer stages to cache intermediate results.

Number of Modules.

We verify the effect of model partition on the final quantized performance. It can be observed from Figure 4.4(a) that by varying the module number within $\{1, 2, 3, 4, 6\}$, fewer model partitions give slightly better performance, as layer-wise dependencies can be better incorporated for reconstruction error minimization. However, this comes at the sacrifice of more memory consumption. Therefore. as a trade-off we partition the model into 4 modules by default. Finally, MREM-S and MREM-P

Table 4.7: Comparison of BERT-base results with and without per-channel quantization (PCQ) on MNLI.

#Bits (W-E-A)	Methods	w/o PCQ		w PCQ	
		Acc m(%)	Acc mm(%)	Acc m(%)	Acc mm(%)
4-4-8	REM	$73.3_{\pm 0.3}$	$74.9_{\pm 0.2}$	$75.9_{\pm 0.3}$	$77.4_{\pm 0.2}$
	MREM	$83.5_{\pm 0.1}$	$83.9_{\pm 0.2}$	$83.6_{\pm 0.1}$	$84.0_{\pm 0.1}$
2-2-8	REM	$71.6_{\pm 0.4}$	$73.4_{\pm 0.4}$	$74.1_{\pm 0.5}$	$75.6_{\pm 0.5}$
	MREM	$82.7_{\pm 0.2}$	$82.7_{\pm 0.2}$	$82.8_{\pm 0.1}$	$82.9_{\pm 0.1}$
2-2-4	REM	$58.3_{\pm 0.5}$	$60.6_{\pm 0.6}$	$59.3_{\pm 0.4}$	$62.0_{\pm 0.4}$
	MREM	$81.1_{\pm 0.2}$	$81.5_{\pm 0.2}$	$81.1_{\pm 0.2}$	$81.5_{\pm 0.3}$

perform on par generally given different model modules. Thus one can safely adopt MREM-P even with more model partitions.

Size of Calibration Data.

The size of calibration data directly relates to the security and privacy issues in post-training quantization. To learn its effects, we vary the calibration data size $|\tilde{\mathcal{D}}|$ within $\{32, 64, 128, 512, 1024, 2048, 4096, 8192\}$, and list the results of both MREM and REM. From Figure 4.4(b), it can be found that while REM is ahead of MREM with less than 128 training samples, it rises slowly and saturates at around 60% afterwards. We hypothesis that the simple training objective in REM does not require too many training instances for optimization. MREM-S, on the other hand, can better exploit the calibration data when more training instances are available, since the module-wise granularity admits higher flexibility for the optimization. As we find the diminishing gain to increase the training size after $4,096$ samples, we by default take $4,096$ samples.

Reconstruction Error Propagation.

Finally, we visualize the propagation of reconstruction error for both "W2-E2-A8" and "W2-E2-A4" quantized BERT-base models in Figure 4.4(c) and Figure 4.4(d) respectively. It can be observed that our MREM achieves both lower values and slower rising rates of the reconstruction error than REM across all layers, which demonstrates the advantage of incorporating more layers for reconstruction error minimization. Interestingly, while the reconstruction error generally gets enlarged layer-wisely in the first ten layers, it begins to shrink at the last two layers. We speculate this is due to the effect of classification head that encourages concentrated hidden representations.

Per-channel Quantization

Per-channel Quantization (PCQ) is prevalent in the post-training quantization of convolution neural networks [117, 119, 181]. To learn its effect in PLMs, we assign different quantization step-sizes at each output dimension, which is also known as row-wise quantization in [6]. The PCQ results from REM and MREM are shown in Table 4.7. It can be found that while PCQ slightly improves REM by 1.0% to 2.5%, the gain is very incremental on MREM. We hypothesis that more training steps of MREM can better adjust the quantization distribution for PLMs. Our results are also similar to the findings in [6], where the row-wise quantization brings little improvement.

4.5 Conclusion

In this chapter, we study post-training quantization for pre-trained language models. We show that existing quantization-aware training solutions suffer from slow training, huge memory consumption, and data privacy issues when accessing the full training set. To mitigate these issues, we propose module-wise reconstruction error minimization, an efficient solution to quantize PLMs. MREM can be conducted either sequentially or in parallel, where the parallel training can achieve the speedup close to the theoretical limit without apparent performance degradation. Experimental results show that the proposed solution greatly improves the performance. Meanwhile, it significantly reduces the training time and memory overhead with only thousands of training instances.

□ **End of chapter.**

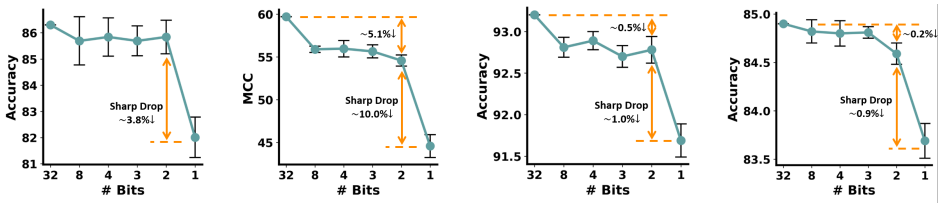
Chapter 5

BinaryBERT: Pushing the Limit of BERT Quantization

In this chapter, we study the problem of BERT binarization, which is the limit of BERT quantization. With adequate training resources, we explore how much can we improve a binarized BERT on natural language understanding tasks. Specifically, we find that a binary BERT is hard to be trained directly than a ternary counterpart due to its complex and irregular loss landscape. Therefore, we propose ternary weight splitting, which initializes BinaryBERT by equivalently splitting from a half-sized ternary network. The binary model thus inherits the good performance of the ternary one, and can be further enhanced by fine-tuning the new architecture after splitting. In the meanwhile, ternary weight splitting can also be conducted adaptively, depending on the resource constraints of various edge devices. Thus it allows to output a series of binary models with different sizes. Empirical results show that BinaryBERT has a significant reduction in model size, with only a slight performance drop compared with the full-precision model. We also achieve state-of-the-art compression results on the GLUE and SQuAD benchmarks.

5.1 Introduction

Recent pre-trained language models have achieved remarkable performance improvement in various natural language tasks [121, 11]. However, the improvement generally comes at the cost of increasing model size and computation, which limits the deployment of these huge pre-trained language models to edge devices. Various methods have been recently proposed to compress these models, such as knowledge distillation [171,



(a) MRPC. (b) CoLA. (c) SST-2. (d) MNLI-m.

Figure 5.1: Performance of quantized BERT with varying weight bit-widths and 8-bit activation. We report the mean results with standard deviations from 10 seeds on MRPC, CoLA, SST-2, and 3 seeds on MNLI-m, respectively.

172, 51], pruning [40, 170], low-rank approximation [190, 174], weight-sharing [173, 174, 176], dynamic networks with adaptive depth and/or width [41, 135, 136], and quantization [114, 177, 189, 6].

Among all these model compression approaches, quantization is a popular solution as it does not require designing a smaller model architecture. Instead, it compresses the model by replacing each 32-bit floating-point parameter with a low-bit fixed-point representation. Existing attempts try to quantize pre-trained models [114, 177, 189] to even as low as ternary values (2-bit) with minor performance drop [6]. However, none of them achieves the binarization (1-bit). As the limit of quantization, weight binarization could bring at most 32× reduction in model size and replace most floating-point multiplications with additions. Moreover, quantizing activations to 8-bit or 4-bit further replaces the floating-point addition with int8 and int4 addition, decreasing the energy burden and the area usage on chips [28].

In this chapter, we explore to binarize BERT parameters with quantized activations, pushing BERT quantization to the limit. We find that directly training a binary network is rather challenging. According to Figure 5.1, there is a sharp performance drop when reducing weight bit-width from 2-bit to 1-bit, compared to other bit configurations. To explore the challenges of binarization, we analyze the loss landscapes of models under different precisions both qualitatively and quantitatively. It is found that while the full-precision and ternary (2-bit) models enjoy relatively flat and smooth loss surfaces, the binary model suffers from a rather steep and complex landscape, which poses great challenges to the optimization.

Motivated by the above empirical observations, we propose *ternary weight splitting*, which takes the ternary model as a proxy to bridge the gap between the binary and full-precision models. Specifically,

ternary weight splitting equivalently converts both the quantized and latent full-precision weights in a well-trained ternary model to initialize BinaryBERT. Therefore, BinaryBERT retains the good performance of the ternary model, and can be further refined on the new architecture. While neuron splitting is previously studied [152, 157] for a full-precision network, our ternary weight splitting is much more complex due to the additional equivalence requirement of quantized weights. Furthermore, the proposed BinaryBERT also supports *adaptive splitting*. It can adaptively perform splitting on the most important ternary modules while leaving the rest as binary, based on efficiency constraints such as model size or floating-point operations (FLOPs). Therefore, our approach allows flexible sizes of binary models for various edge devices' demands.

Empirical results show that BinaryBERT split from a half-width ternary network is much better than a directly-trained binary model with the original width. On the GLUE and SQuAD benchmarks, our BinaryBERT has only a slight performance drop compared to the full-precision BERT-base model, while being **24**× smaller. Moreover, BinaryBERT with the proposed importance-based adaptive splitting also outperforms other splitting criteria across a variety of model sizes.

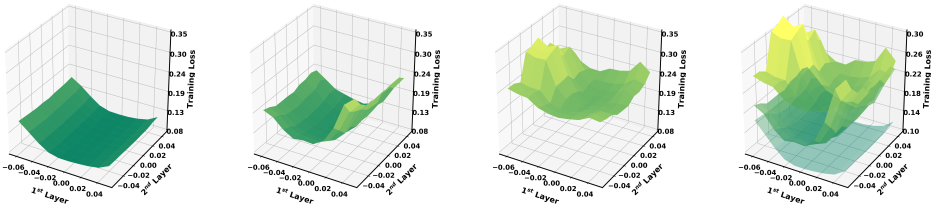
5.2 Motivation

In this section, we show that it is challenging to train a binary BERT with conventional binarization approaches directly. Before diving into details, we first review necessary backgrounds for ternarization and binarization.

As we consider the setting with adequate training resources, we follow the standard quantization-aware training procedure [29] as introduced in Section 2.1.2. Recent TernaryBERT [6] follows Ternary-Weight-Network (TWN) [95] to quantize the elements in $\mathbf{w}$ to three values $\{\pm\alpha, 0\}$. To avoid confusion, we use superscript t and b for the latent full-precision weights and quantized weights in ternary and binary models, respectively. Recall in Equation (2.6) that TWN ternarizes each element w_i^t in the ternary weight $\mathbf{w}^t$ by

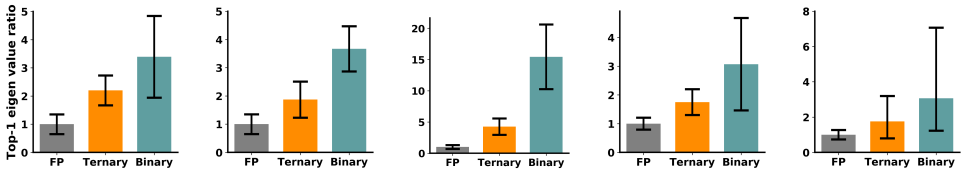
$$\hat{w}_i^t = \mathcal{Q}(w_i^t) = \begin{cases} \alpha \cdot \text{sign}(w_i^t) & |w_i^t| \geq \Delta \\ 0 & |w_i^t| < \Delta \end{cases}, \tag{5.1}$$

where $\text{sign}(\cdot)$ is the sign function, $\Delta = \frac{0.7}{n}\|\mathbf{w}^t\|_1$ and $\alpha = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}|w_i^t|$



(a) Full-precision Model. (b) Ternary Model. (c) Binary Model. (d) All Together.

Figure 5.2: Loss landscapes visualization of the full-precision, ternary and binary models on MRPC. For (a), (b) and (c), we perturb the (latent) full-precision weights of the value layer in the 1st and 2nd Transformer layers, and compute their corresponding training loss. (d) shows the gap among the three surfaces by stacking them together.



(a) MHA-QK. (b) MHA-V. (c) MHA-O. (d) FFN-Mid. (e) FFN-Out.

Figure 5.3: The top-1 eigenvalues of parameters at different Transformer parts of the full-precision (FP), ternary and binary BERT. For easy comparison, we report the ratio of eigenvalue between the ternary/binary models and the full-precision model. The error bar is estimated of all Transformer layers over different data mini-batches.

with $\mathcal{I} = \{i \mid \hat{w}_i^t \neq 0\}$.

Binarization represent network parameters within only two values [28, 45, 97, 47]. For instance, Binary-Weight-Network (BWN) [97] in Equation (2.9) takes the form of

$$\hat{w}_i^b = \mathcal{Q}(w_i^b) = \alpha \cdot \text{sign}(w_i^b), \ \alpha = \frac{1}{n}\|\mathbf{w}^b\|_1. \tag{5.2}$$

Despite the appealing properties of network binarization, we show that it is non-trivial to obtain a binary BERT with these binarization approaches.

5.2.1 Sharp Performance Drop with Weight Binarization

To study the performance drop of BERT quantization, we train the BERT model with full-precision, {8,4,3,2,1}-bit weight quantization and 8-bit activations on MRPC and MNLI-m from the GLUE bench-

mark [187]. We use loss-aware weight quantization (LAQ) [104] for 8/4/3-bit weight quantization, TWN [95] for weight ternarization and BWN [97] for weight binarization. Meanwhile, we adopt 8-bit uniform quantization for activations. We follow the default experimental settings detailed in Section 5.4.1.

From Figure 5.1, the performance drops mildly from 32-bit to as low as 2-bit, i.e., around 0.6% ↓ on MRPC and 0.2% ↓ on MNLI-m. However, when reducing the bit-width to one, the performance drops sharply, i.e, $\sim$ 3.8% ↓ and $\sim$ 0.9% ↓ on the two tasks, respectively. Therefore, weight binarization may severely harm the performance, which may explain why most current approaches stop at 2-bit weight quantization [177, 116, 6]. To further push weight quantization to the limit, a first step is to study the potential reasons behind the sharp drop from ternarization to binarization.

5.2.2 Exploring the Quantized Loss Landscape

Visualization. To learn about the challenges behind the binarization, we first visually compare the loss landscapes of full-precision, ternary, and binary BERT models. Following [117], we extract parameters $\mathbf{w}_x, \mathbf{w}_y$ from the value layers of multi-head attention in the first two Transformer layers, and assign the following perturbations on parameters:

$$\tilde{\mathbf{w}}_x = \mathbf{w}_x + x \cdot \mathbf{1}_x, \quad \tilde{\mathbf{w}}_y = \mathbf{w}_y + y \cdot \mathbf{1}_y, \tag{5.3}$$

where $x \in \{\pm 0.2\bar{w}_x, \pm 0.4\bar{w}_x, ..., \pm 1.0\bar{w}_x\}$ are perturbation magnitudes based the absolute mean value $\bar{w}_x$ of $\mathbf{w}_x$, and similar rules hold for y. $\mathbf{1}_x$ and $\mathbf{1}_y$ are vectors with all elements being 1. For each pair of (x, y), we evaluate the corresponding training loss and plot the surface in Figure 5.2.

As can be seen, the full-precision model (Figure 5.2(a)) has the lowest overall training loss, and its loss landscape is flat and robust to the perturbation. For the ternary model (Figure 5.2(b)), despite the surface tilts up with larger perturbations, it looks locally convex and is thus easy to optimize. This may also explain why the BERT model can be ternarized without severe accuracy drop [6]. However, the loss landscape of the binary model (Figure 5.2(c)) turns out to be both higher and more complex. By stacking the three landscapes together (Figure 5.2(d)), the loss surface of the binary BERT stands on the top with a clear margin with the other two. The steep curvature of the loss

surface reflects a higher sensitivity to binarization, which attributes to the training difficulty.

Steepness Measurement. To quantitatively measure the steepness of loss landscape, we start from a local minima $\mathbf{w}$ and apply the second order approximation to the curvature. According to the Taylor's expansion, the loss increase induced by quantizing $\mathbf{w}$ can be approximately upper bounded by

$$\ell(\hat{\mathbf{w}}) - \ell(\mathbf{w}) \approx \boldsymbol{\epsilon}^\top \mathbf{H} \boldsymbol{\epsilon} \leq \lambda_{\max} \|\boldsymbol{\epsilon}\|^2, \tag{5.4}$$

where $\boldsymbol{\epsilon} = \mathbf{w} - \hat{\mathbf{w}}$ is the quantization noise, and $\lambda_{\max}$ is the largest eigenvalue of the Hessian $\mathbf{H}$ at $\mathbf{w}$. Note that the first-order term is skipped due to $\nabla \ell(\mathbf{w}) = 0$. Thus we take $\lambda_{\max}$ as a quantitative measurement for the steepness of the loss surface. Following [177] we adopt the power method to compute $\lambda_{\max}$. As it is computationally expensive to estimate $\mathbf{H}$ for all $\mathbf{w}$ in the network, we consider them separately as follows: (1) the query/key layers (MHA-QK), (2) the value layer (MHA-V), (3) the output projection layer (MHA-O) in the multi-head attention, (4) the intermediate layer (FFN-Mid), and (5) the output layer (FFN-Out) in the feed-forward network. Note that we group key and query layers as they are used together to calculate the attention scores.

From Figure 5.3, the top-1 eigenvalues of the binary model are higher both on expectation and standard deviation compared to the full-precision baseline and the ternary model. For instance, the top-1 eigenvalues of MHA-O in the binary model are $\sim 15\times$ larger than the full-precision counterpart. Therefore, the quantization loss increases of the full-precision and ternary model are tighter bounded than the binary model in Equation (5.4). The highly complex and irregular landscape by binarization thus poses more challenges to the optimization.

5.3 Proposed Method

5.3.1 Ternary Weight Splitting

Given the challenging loss landscape of binary BERT, we propose *ternary weight splitting* (TWS) that exploits the flatness of ternary loss landscape as the optimization proxy of the binary model. As is shown in Figure 5.4, we first train the half-sized ternary BERT to convergence, and then split both the latent full-precision weight $\mathbf{w}^t$ and quantized $\hat{\mathbf{w}}^t$ to their

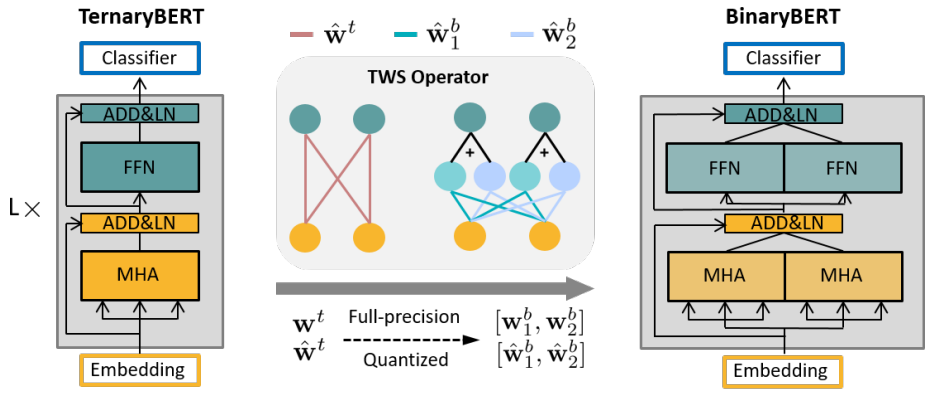


Figure 5.4: The overall workflow of training BinaryBERT. We first train a half-sized ternary BERT model, and then apply ternary weight splitting operator (Equations (5.6) and (5.7)) to obtain the latent full-precision and quantized weights as the initialization of the full-sized BinaryBERT. We then fine-tune BinaryBERT for further refinement.

binary counterparts $\mathbf{w}_1^b, \mathbf{w}_2^b$ and $\hat{\mathbf{w}}_1^b, \hat{\mathbf{w}}_2^b$ via the *TWS operator*. To inherit the performance of the ternary model after splitting, the TWS operator requires the splitting equivalency (i.e., the same output given the same input):

$$\mathbf{w}^t = \mathbf{w}_1^b + \mathbf{w}_2^b, \quad \hat{\mathbf{w}}^t = \hat{\mathbf{w}}_1^b + \hat{\mathbf{w}}_2^b \ . \tag{5.5}$$

While solution to Equation (5.5) is not unique, we constrain the latent full-precision weights after splitting $\mathbf{w}_1^b, \mathbf{w}_2^b$ to satisfy $\mathbf{w}^t = \mathbf{w}_1^b + \mathbf{w}_2^b$ as

$$w_{1,i}^b = \begin{cases} a \cdot w_i^t & \text{if } \ \hat{w}_i^t \neq 0 \\ b + w_i^t & \text{if } \ \hat{w}_i^t = 0, w_i^t > 0 \\ b & \text{otherwise} \end{cases}, \tag{5.6}$$

$$w_{2,i}^b = \begin{cases} (1-a)w_i^t & \text{if } \ \hat{w}_i^t \neq 0 \\ -b & \text{if } \ \hat{w}_i^t = 0, w_i^t > 0 \\ -b + w_i^t & \text{otherwise} \end{cases}, \tag{5.7}$$

where $a \in (0, 1)$ and $b \geq 0$ are the variables to solve. To obtain the expressions for a and b, we denote $\mathcal{I} = \{i \mid \hat{w}_i^t \neq 0\}$, $\mathcal{J} = \{j \mid \hat{w}_j^t = 0 \text{ and } w_j^t > 0\}$ and $\mathcal{K} = \{k \mid \hat{w}_k^t = 0 \text{ and } w_k^t < 0\}$. According to the BWN quantizer introduced in S†ection 5.2, we have

$$\hat{w}_{1,i}^b = \alpha_1 \text{sign}(w_{1,i}^b),$$

where

$$\alpha_1 = \frac{1}{n}\Big[\sum_{i\in\mathcal{I}} |a w_i^t| + \sum_{i\in\mathcal{J}} |w_j^t + b| + \sum_{i\in\mathcal{K}} |b|\Big].$$

Similarly,

$$\hat{w}_{2,i}^b = \alpha_2 \text{sign}(w_{2,i}^b),$$

where

$$\alpha_2 = \frac{1}{n}\Big[\sum_{i\in\mathcal{I}} |(1-a)w_i^t| + \sum_{j\in\mathcal{J}} |-b| + \sum_{k\in\mathcal{K}} |w_k^t - b|\Big].$$

Since $\hat{\mathbf{w}}^t = \hat{\mathbf{w}}_1^b + \hat{\mathbf{w}}_2^b$, for those $\hat{w}_i^t = \hat{w}_{1,i}^b + \hat{w}_{2,i}^b = 0$, we have

$$\begin{aligned}
&\frac{1}{n}\Big[\sum_{i\in\mathcal{I}} |aw_i^t| + \sum_{j\in\mathcal{J}} |w_j^t + b| + \sum_{k\in\mathcal{K}} |b|\Big] \\
&= \frac{1}{n}\Big[\sum_{i\in\mathcal{I}} |(1-a)w_i^t| + \sum_{j\in\mathcal{J}} |-b| + \sum_{k\in\mathcal{K}} |w_k^t - b|\Big].
\end{aligned}$$

By assuming $0 < a < 1$ and $b > 0$, this can be further simplified to

$$a\sum_{i\in\mathcal{I}} |w_i^t| + \sum_{j\in\mathcal{J}} |w_j^t| = (1-a)\sum_{i\in\mathcal{I}} |w_i^t| + \sum_{k\in\mathcal{K}} |w_k^t|,$$

which gives the solution of a as

$$a = \frac{\sum_{i\in\mathcal{I}} |w_i^t| + \sum_{j\in\mathcal{J}} |w_j^t| - \sum_{k\in\mathcal{K}} |w_k^t|}{2\sum_{i\in\mathcal{I}} |w_i^t|}. \tag{5.8}$$

We empirically find the solution satisfies $0 < a < 1$. To solve b, we take $|\cdot|$ as the cardinality of the set. Then for $\hat{w}_i^t \neq 0$, from $\hat{w}_i^t = \hat{w}_{1,i}^b + \hat{w}_{2,i}^b$, we have

$$\begin{aligned}
&\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} |w_i^t| = \alpha_1 + \alpha_2 \\
&= \frac{1}{n}\Big[\sum_{i\in\mathcal{I}} |aw_i^t| + \sum_{j\in\mathcal{J}} |w_j^t + b| + \sum_{k\in\mathcal{K}} |b|\Big] \\
&+ \frac{1}{n}\Big[\sum_{i\in\mathcal{I}} |(1-a)w_i^t| + \sum_{j\in\mathcal{J}} |-b| + \sum_{k\in\mathcal{K}} |w_k^t - b|\Big] \\
&= \frac{1}{n}\Big[\sum_{i\in\mathcal{I}} |w_i^t| + \sum_{j\in\mathcal{J}} |w_j^t| + \sum_{k\in\mathcal{K}} |w_k^t| \\
&+ 2\sum_{j\in\mathcal{J}} |b| + 2\sum_{k\in\mathcal{K}} |b|\Big] \\
&= \frac{1}{n}\Big[\sum_{i=1}^{n} |w_i^t| + 2(|\mathcal{J}| + |\mathcal{K}|)\cdot b\Big].
\end{aligned}$$

Thus the solution for b is

$$b = \frac{\frac{n}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} |w_i^t| - \sum_{i=1}^{n} |w_i^t|}{2(|\mathcal{J}| + |\mathcal{K}|)}, \tag{5.9}$$

which satisfies $b > 0$.

With Equation (5.8) and Equation (5.9), the ternary weight splitting in Equation (5.6) and Equation (5.7) can be performed on the fly immediately given a half-sized ternary BERT.

Quantization Details. Following [6], for each weight matrix in the Transformer layers, we use layer-wise ternarization (i.e., one scaling parameter for all elements in the weight matrix). For word embedding, we use row-wise ternarization (i.e., one scaling parameter for each row in the embedding). After splitting, each of the two split matrices has its own scaling factor.

Aside from weight binarization, we simultaneously quantize activations before all matrix multiplications, which could accelerate inference on specialized hardwares [177, 114]. Following [114, 6], we skip the quantization for all layer-normalization (LN) layers, skip connections, and bias as their calculations are negligible compared to matrix multiplication. The last classification layer is also not quantized to avoid a large accuracy drop.

Training with Knowledge Distillation. Knowledge distillation is shown to benefit BERT quantization [6]. Following [51, 6], we first perform *intermediate-layer distillation* from the full-precision teacher network's embedding $\mathbf{E}$, layer-wise MHA output $\mathbf{M}_l$ and FFN output $\mathbf{F}_l$ to the quantized student counterpart $\hat{\mathbf{E}}, \hat{\mathbf{M}}_l, \hat{\mathbf{F}}_l$ $(l = 1, 2, ...L)$. We aim to minimize their mean sqaured errors, i.e., $\ell_{emb} = \text{MSE}(\hat{\mathbf{E}}, \mathbf{E})$, $\ell_{mha} = \sum_l \text{MSE}(\hat{\mathbf{M}}_l, \mathbf{M}_l)$, and $\ell_{ffn} = \sum_l \text{MSE}(\hat{\mathbf{F}}_l, \mathbf{F}_l)$. Thus the objective function is

$$\ell_{int} = \ell_{emb} + \ell_{mha} + \ell_{ffn}. \tag{5.10}$$

We then conduct *prediction-layer distillation* by minimizing the soft cross-entropy (SCE) between quantized student logits $\hat{\mathbf{y}}$ and teacher logits $\mathbf{y}$, i.e.,

$$\ell_{pred} = \text{SCE}(\hat{\mathbf{y}}, \mathbf{y}). \tag{5.11}$$

Further Fine-tuning. After splitting from the half-sized ternary model, the binary model inherits its performance on a new architecture with full width. However, the original minimum of the ternary model may not hold in this new loss landscape after splitting. Thus we further fine-tune with prediction-layer distillation to look for a better solution. We dub

the resulting model as BinaryBERT.

5.3.2 Adaptive Splitting

Our proposed approach also supports *adaptive splitting* that can flexibly adjust the width of BinaryBERT, based on the parameter sensitivity to binarization and resource constraints of edge devices. Specifically, given the resource constraints $\mathcal{C}$ of various edge devices (e.g., model size and computational FLOPs), we first train a mixed-precision model adaptively (with sensitive parts being ternary and the rest being binary), and then split ternary weights into binary ones. Therefore, adaptive splitting finally enjoys consistent arithmetic precision (1-bit) for all weight matrices, which is usually easier to deploy than the mixed-precision counterpart.

Sensitivity Measurement. Intuitively, we assign ternary values to weight matrices that are more sensitive to quantization. The adaptive splitting requires to first estimate the quantization sensitivity vector $\mathbf{u} \in \mathbb{R}_+^Z$, where Z is the total number of splittable weight matrices in all Transformer layers, the word embedding layer and the pooler layer. We study the sensitivity $\mathbf{u}$ in two aspects: the Transformer parts, and the Transformer layers. For Transformer parts, we follow the weight categorization in Section 5.2.2: MHA-Q/K, MHA-V, MHA-O, FFN-Mid and FFN-Out. For each of them, we compare the performance gap between quantizing and not quantizing that part (e.g., MHA-V), while leaving the rest parts all quantized (e.g., MHA-Q/K, MHA-O, FFN-Mid and FFN-Out). Similarly, for each Transformer layer, we quantize all layers but leave the layer under investigation un-quantized, and calculate the performance gain compared with the fully quantized baseline. The performance gain of both Transformer parts and layers is shown in Figure 5.5. As can be seen, for Transformer parts, the FFN-Mid and MHA-Q/K rank in the first and second place. In terms of Transformer layers, shallower layers are more sensitive to quantization than the deeper ones.

However, the absolute performance gain may not truly reflect the quantization sensitivity directly, since Transformer parts have different number of parameters. Therefore, we further divide the performance gain by the number of parameters in that part or layer, so as to obtain the parameter-wise performance gain. We are thus able to measure the quantization sensitivity of the ith Transformer part in the jth

Transformer layer by summing their parameter-wise performance gain together. We also apply the same procedure to word embedding and the pooler layer to obtain their sensitivity scores.

Formulation. Given the sensitivity vector $\mathbf{u}$, adaptive splitting can be formulated as a combinatorial optimization problem. Specifically, the splitting assignment can be represented as a binary vector $\mathbf{s} \in \{0,1\}^Z$, where $s_z = 1$ means to ternarize the z-th weight matrix, and vice versa. We also denote $\mathbf{c} \in \mathbb{R}_+^Z$ as the cost vector, which stores the additional increase of parameter or FLOPs of each ternary weight matrix against a binary choice. The optimal assignment $\mathbf{s}^*$ can thus be solved from the following constrained problem:

$$\begin{aligned} \max_{\mathbf{s}} \quad & \mathbf{u}^\top \mathbf{s} \ , \\ \text{s.t.} \quad & \mathbf{c}^\top \mathbf{s} \leq \mathcal{C} - \mathcal{C}_0, \ \ \mathbf{s} \in \{0,1\}^Z, \end{aligned} \tag{5.12}$$

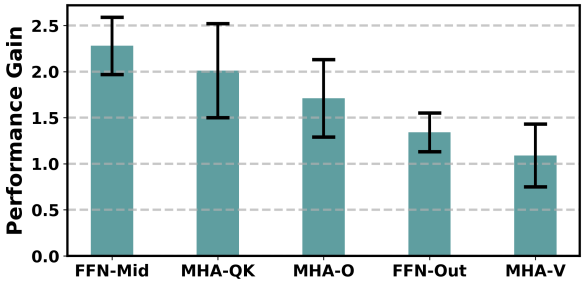
where $\mathcal{C}$ is the pre-specified resource constraint. $\mathcal{C}_0$ is the baseline efficiency of the half-sized binary network, which is of the minimal resource consumption. Equation (5.12) can be solved by dynamic programming as a knapsack problem, where the constraint $\mathcal{C} - \mathcal{C}_0$ is the volume of the knapsack, and the sensitivity scores $\mathbf{u}$ are the item values.

5.4 Experiments

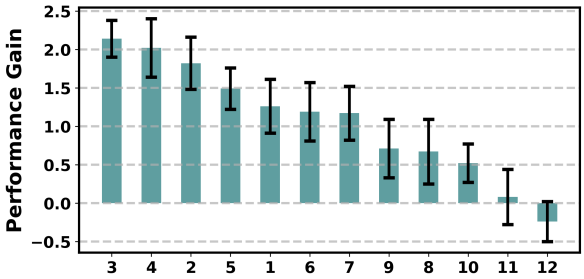
In this section, we empirically verify our proposed approach on the GLUE [187] and SQuAD [188, 191] benchmarks. We first introduce the experimental setup in Section 5.4.1, and then present the main experimental results on both benchmarks in Section 5.4.2. We compare with other state-of-the-arts in Section 5.4.3, and finally provide more discussions on the proposed methods in Section 5.4.4.

5.4.1 Experimental Setup

Dataset and Metrics. The GLUE benchmark contains multiple natural language understanding tasks. We follow [11] to evaluate the performance on these tasks: Matthews correlation for CoLA, Spearman correlation for STS-B and accuracy for the rest tasks: RTE, MRPC, SST-2, QQP, MNLI-m (matched) and MNLI-mm (mismatched). For machine reading comprehension on SQuAD, we report the EM (exact match) and F1 score.



(a) Transformer Parts.



(b) Transformer Layers.

Figure 5.5: The performance gain of different Transformer parts and layers in descending order. All numbers are averaged by 10 random runs with standard deviations reported.

Aside from the task performance, we also report the model size (MB) and computational FLOPs at inference. For quantized operations, we follow [29, 47, 32] to count the bit-wise operations, i.e., the multiplication between an m-bit number and an n-bit number approximately takes $mn/64$ FLOPs for a CPU with the instruction size of 64 bits.

Implementation. We take DynaBERT [41] sub-networks as backbones as they offer both half-sized and full-sized models for easy comparison. We start with a ternary model of width 0.5× by conducting two-stage knowledge distillation introduced in Section 5.3.1, i.e., intermediate-layer distillation (Int. Dstil.) and prediction-layer distillation (Pred. Dstil.). Then we perform ternary weight splitting to obtain a full-sized binary model, followed by fine-tuning (Split Ft.) with only prediction-layer distillation. The initial learning rate is set as 5×10^{-5} for the intermediate-layer distillation, and 2×10^{-5} for the prediction-layer distillation, both of which linearly decay to 0 at the end of training. We conduct experiments on GLUE tasks both without and with data augmentation (DA) except for MNLI and QQP due to their limited performance gain. The running epochs for MNLI and QQP are set to 3, and 6 for the rest tasks if without DA and 1 otherwise. For the rest

Table 5.1: Hyper-parameters for training BinaryBERT on the GLUE benchmark at different stages.

	BinaryBERT		
	Int. Dstil. (Ternary)	Pred. Dstil. (Ternary)	Split Ft. (Binary)
Batch Size	32	32	32
Sequence Length	128	128	128
Learning rate (LR)	5e-5	2e-5	2e-5
LR Decay	Linear	Linear	Linear
Warmup portion	0.1	0.1	0.1
Weight Decay	1e-2	1e-2	1e-2
Gradient Clipping	1	1	1
Dropout	0.1	0.1	0.1
Epochs w/o DA -other datasets	6	6	6
Epochs w DA -other datasets	1	1	1
Epochs w/o DA -MNLI, QQP	3	3	3

hyper-parameters, we follow the default setting in [11]. The detailed hyper-parameters are summarized in Table 5.1.

We verify our ternary weight splitting (**TWS**) against vanilla binary training (**BWN**), the latter of which doubles training epochs to match the overall training time in TWS for a fair comparison.

Activation Quantization. While BinaryBERT focuses on weight binarization, we also explore activation quantization in our implementation, which is beneficial for reducing the computation burden on specialized hardwares [97, 29, 6]. Aside from 8-bit uniform quantization [6, 177] in past efforts, we further pioneer to study 4-bit activation quantization. We find that uniform quantization can hardly deal with outliers in the activation. Thus we use Learned Step-size Quantization (LSQ) [31] to directly learn the quantized values, which empirically achieves better quantization performance.

5.4.2 Experimental Results

Table 5.2: Results on the GLUE development set. "#Bits (W-E-A)" represents the bit number for weights of Transformer layers, word embedding, and activations. "DA" is short for data augmentation. "Avg." denotes the average results of all tasks including MNLI-m and MNLI-mm. The higher results in each block are bolded.

#	Quant	#Bits (W-E-A)	Size (MB)	FLOPs (G)	DA	MNLI -m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
1	-	*full-prec.*	417.6	22.5	-	84.9/85.5	91.4	92.1	93.2	59.7	90.1	86.3	72.2	83.9
2	BWN	1-1-8	13.4	3.1	✗	84.2/84.0	91.1	90.7	92.3	46.7	86.8	82.6	68.6	80.8
3	TWS	1-1-8	16.5	3.1	✗	**84.2/84.7**	**91.2**	**91.5**	**92.6**	**53.4**	**88.6**	**85.5**	**72.2**	**82.7**
4	BWN	1-1-4	13.4	1.5	✗	83.5/83.4	90.9	90.7	**92.3**	34.8	84.9	79.9	**65.3**	78.4
5	TWS	1-1-4	16.5	1.5	✗	**83.9/84.2**	**91.2**	**90.9**	**92.3**	**44.4**	**87.2**	**83.3**	**65.3**	**79.9**
6	BWN	1-1-8	13.4	3.1	✓	84.2/84.0	91.1	91.2	92.7	54.2	88.2	**86.8**	70.0	82.5
7	TWS	1-1-8	16.5	3.1	✓	**84.2/84.7**	**91.2**	**91.6**	**93.2**	**55.5**	**89.2**	86.0	**74.0**	**83.3**
8	BWN	1-1-4	13.4	1.5	✓	83.5/83.4	90.9	91.2	92.5	51.9	87.7	85.5	70.4	81.9
9	TWS	1-1-4	16.5	1.5	✓	**83.9/84.2**	**91.2**	**91.4**	**93.7**	**53.3**	**88.6**	**86.0**	**71.5**	**82.6**

Table 5.3: Results on the GLUE test set scored using the GLUE evaluation server.

#	Quant	#Bits (W-E-A)	Size (MB)	FLOPs (G)	DA	MNLI -m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
1	-	*full-prec.*	417.6	22.5	-	84.5/84.1	89.5	91.3	93.0	54.9	84.4	87.9	69.9	82.2
2	BWN	1-1-8	13.4	3.1	✗	83.3/83.4	88.9	**90.1**	92.3	38.1	81.2	**86.1**	63.1	78.5
3	TWS	1-1-8	16.5	3.1	✗	**84.1/83.6**	**89.0**	90.0	**93.1**	**50.5**	**83.4**	86.0	**65.8**	**80.6**
4	BWN	1-1-4	13.4	1.5	✗	83.5/82.5	**89.0**	**89.4**	92.3	26.7	78.9	84.2	59.9	76.3
5	TWS	1-1-4	16.5	1.5	✗	**83.6/82.9**	**89.0**	89.3	**93.1**	**37.4**	**82.5**	**85.9**	**62.7**	**78.5**
6	BWN	1-1-8	13.4	3.1	✓	83.3/83.4	88.9	**90.3**	91.3	48.4	**83.2**	**86.3**	66.1	80.1
7	TWS	1-1-8	16.5	3.1	✓	**84.1/83.5**	**89.0**	89.8	**91.9**	**51.6**	82.3	85.9	**67.3**	**80.6**
8	BWN	1-1-4	13.4	1.5	✓	83.5/82.5	**89.0**	**89.9**	92.0	45.0	81.9	85.2	64.1	79.2
9	TWS	1-1-4	16.5	1.5	✓	**83.6/82.9**	**89.0**	89.7	**93.1**	**47.9**	**82.9**	**86.6**	**65.8**	**80.2**

Table 5.4: Development set results (EM/F1) on SQuAD.

Quant	#Bits (W-E-A)	Size (MB)	FLOPs (G)	SQuAD v1.1	SQuAD v2.0
-	*full-prec.*	417.6	22.5	82.6/89.7	75.1/77.5
BWN	1-1-8	13.4	3.1	79.2/86.9	**73.6/76.6**
TWS	1-1-8	16.5	3.1	**80.8/88.3**	73.6/76.5
BWN	1-1-4	13.4	1.5	77.5/85.8	71.9/75.1
TWS	1-1-4	16.5	1.5	**79.3/87.2**	**72.5/75.4**

Results on the GLUE Benchmark

The main results on the development set are shown in Table 5.2. For results without data augmentation (row #2-5), our ternary weight splitting method outperforms BWN with a clear margin [1]. For instance, on CoLA, ternary weight splitting achieves 6.7% ↑ and 9.6% ↑ with 8-bit and 4-bit activation quantization, respectively. While data augmentation (row 6-9) mostly improves each entry, our approach still overtakes BWN consistently. Furthermore, 4-bit activation quantization empirically benefits more from ternary weight splitting (row 4-5 and 8-9) compared with 8-bit activations (row 2-3 and 6-7), demonstrating the potential of our approach in extremely low bit quantized models.

In Table 5.3, we also provide the results on the test set of GLUE benchmark. Similar to the observation in Table 5.2, our approach achieves consistent improvement on both 8-bit and 4-bit activation quantization compared with BWN.

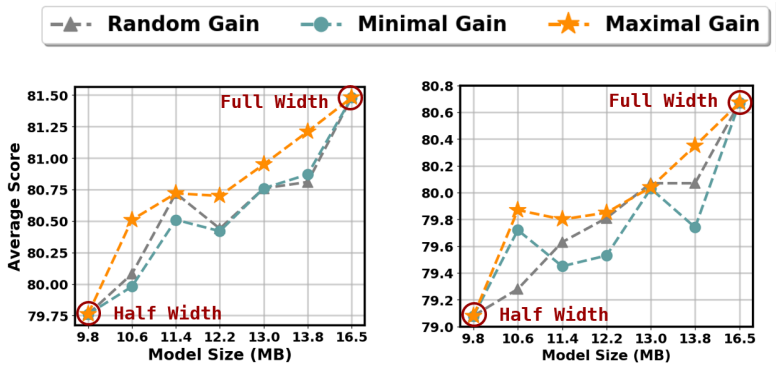
Results on SQuAD Benchmark

The results on the development set of SQuAD v1.1 and v2.0 are shown in Table 5.4. Our proposed ternary weight splitting again outperforms BWN w.r.t both EM and F1 scores on both datasets. Similar to previous observations, 4-bit activation enjoys a larger gain in performance from the splitting approach. For instance, our approach improves the EM score of 4-bit activation by 1.8% and 0.6% on SQuAD v1.1 and v2.0, respectively, both of which are higher than those of 8-bit activation.

Adaptive Splitting

The adaptive splitting in Section 5.3.2 supports the conversion of mixed ternary and binary precisions for more-fine-grained configurations. To verify its advantages, we name our approach as Maximal Gain according

[1] Note that DynaBERT only squeezes width in the Transformer layers but not the word embedding layer, thus the split binary model has a slightly larger size than BWN.



(a) 8-bit Activation. (b) 4-bit Activation.

Figure 5.6: The average performance over six GLUE tasks of adaptive splitting strategies.

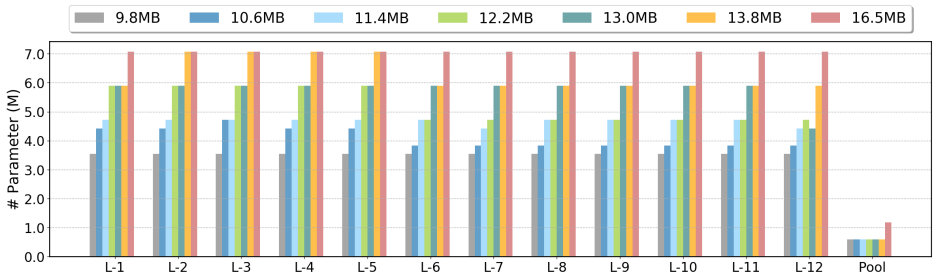


Figure 5.7: The architecture visualization for adaptive splitting on MRPC. The y-axis records the number of parameters split in each layer instead of the storage.

to Equation (5.12), and compare it with two baseline strategies i) Random Gain that randomly selects weight matrices to split; and ii) Minimal Gain that splits the least important modules according to sensitivity. We report the average score over six tasks (QNLI, SST-2, CoLA, STS-B, MRPC and RTE) in Figure 5.6. The end-points of 9.8MB and 16.5MB are the half-sized and full-sized BinaryBERT, respectively. As can be seen, adaptive splitting generally outperforms the other two baselines under varying model sizes, indicating the effectiveness of maximizing the gain in adaptive splitting.

As an illustration example, we further visualize the architectures after adaptive splitting on MRPC in Figure 5.7. For clear presentation, we merge all splittable parameters in each Transformer layer. As the baseline, 9.8MB refers to no splitting, while 16.5MB refers to splitting all splittable parameters in the model. According to Figure 5.7, with the increasing model size, shallower layers are more preferred for splitting than deeper layers.

Table 5.5: Comparison with other state-of-the-art methods on development set of SQuAD v1.1 and MNLI-m.

Method	#Bits (W-E-A)	Size (MB)	Ratio (↓)	SQuAD v1.1	MNLI -m
BERT-base	*full-prec.*	418	1.0	80.8/88.5	84.6
DistilBERT	*full-prec.*	250	1.7	79.1/86.9	81.6
LayerDrop-6L	*full-prec.*	328	1.3	-	82.9
LayerDrop-3L	*full-prec.*	224	1.9	-	78.6
TinyBERT-6L	*full-prec.*	55	7.6	79.7/87.5	82.8
ALBERT-E128	*full-prec.*	45	9.3	82.3/89.3	81.6
ALBERT-E768	*full-prec.*	120	3.5	81.5/88.6	82.0
Quant-Noise	PQ	38	11.0	-	83.6
Q-BERT	2/4-8-8	53	7.9	79.9/87.5	83.5
Q-BERT	2/3-8-8	46	9.1	79.3/87.0	81.8
Q-BERT	2-8-8	28	15.0	69.7/79.6	76.6
GOBO	3-4-32	43	9.7	-	83.7
GOBO	2-2-32	28	15.0	-	71.0
TernaryBERT	2-2-8	28	15.0	79.9/87.4	83.5
BinaryBERT	**1-1-8**	**17**	**24.6**	**80.8/88.3**	**84.2**
BinaryBERT	**1-1-4**	**17**	**24.6**	**79.3/87.2**	**83.9**

5.4.3 Comparison with State-of-the-arts

Now we compare our proposed approach with a variety of state-of-the-art counterparts, including Q-BERT [177], GOBO [116], Quant-Noise [189] and TernaryBERT [6]. Aside from quantization, we also compare with other general compression approaches such as DistillBERT [171], LayerDrop [170], TinyBERT [51], and ALBERT [174]. The results are taken from the original papers, respectively. From Table 5.5, our proposed BinaryBERT has the smallest model size with the best performance among all quantization approaches. Compared with the full-precision model, our BinaryBERT retains competitive performance with a significant reduction of model size and computation. For example, we achieve more than **24**× compression ratio compared with BERT-base, with only 0.4% ↓ and 0.0%/0.2% ↓ drop on MNLI-m on SQuAD v1.1, respectively.

5.4.4 Discussion

Further Improvement after Splitting

We now demonstrate the performance gain by refining the binary model on the new architecture. We evaluate the performance gain after splitting from a half-width ternary model ($\text{TWN}_{0.5\times}$) to the full-sized model ($\text{TWN}_{1.0\times}$) on the development set of SQuAD v1.1, MNLI-m, QNLI and MRPC. The results are shown in Table 5.6. As can be seen, further fine-tuning brings consistent improvement on both 8-bit and 4-bit activation.

Table 5.6: The performance gain by fine-tuning the binary model after splitting. 0.5× and 1.0× denote the half-sized and full-sized models, respectively.

Quant	#Bits (W-E-A)	SQuAD v1.1	MNLI -m	QNLI	MRPC
$\text{TWN}_{0.5\times}$	2-2-8	80.3/87.9	84.1	91.3	85.7
$\text{TWS}_{1.0\times}$	1-1-8	**80.8/88.3**	**84.2**	**91.6**	**86.0**
$\text{TWN}_{0.5\times}$	2-2-4	78.0/86.4	83.7	90.9	85.5
$\text{TWS}_{1.0\times}$	1-1-4	**79.3/87.2**	**83.9**	**91.4**	**86.0**

Table 5.7: Comparison with other binarization methods.

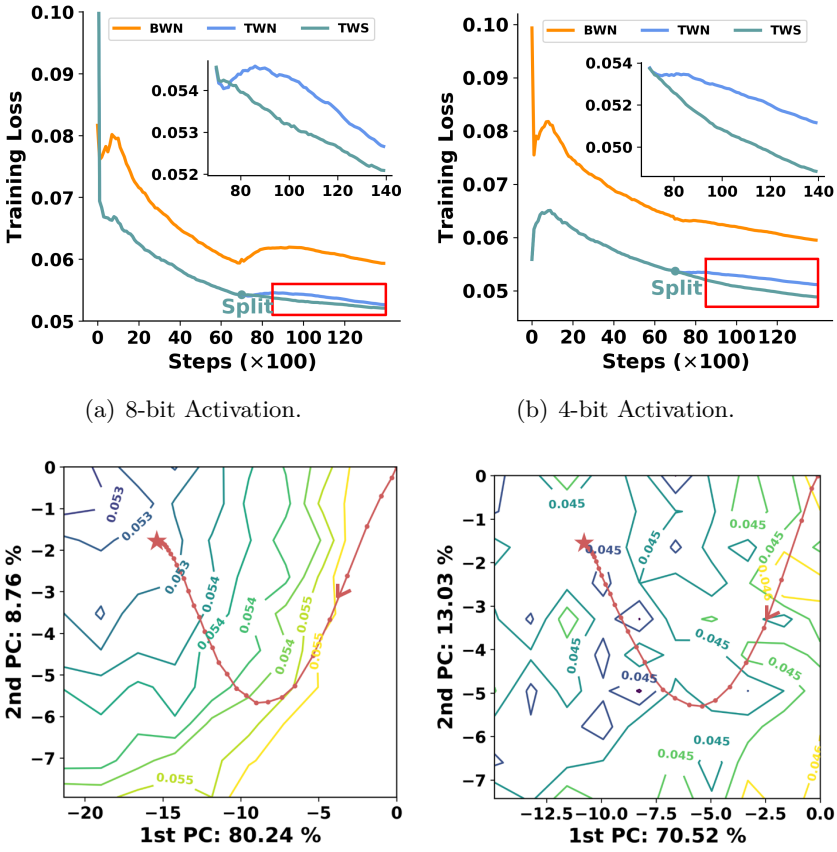
Quant	#Bits (W-E-A)	SQuAD v1.1	MNLI -m	QNLI	SST-2
BWN	1-1-8	79.2/86.9	**84.2**	91.2	92.7
LAB	1-1-8	79.0/87.0	83.6	91.5	92.8
BiReal	1-1-8	79.4/87.1	83.9	91.4	92.5
BWN†	1-1-8	79.4/87.3	**84.2**	91.3	92.8
BWN‡	1-1-8	79.6/87.2	83.5	91.2	92.9
TWS	1-1-8	**80.8/88.3**	**84.2**	**91.6**	**93.2**
BWN	1-1-4	77.5/85.8	83.5	91.2	92.5
LAB	1-1-4	76.7/85.5	83.3	91.3	92.9
BiReal	1-1-4	76.9/85.4	83.4	91.0	92.8
BWN†	1-1-4	78.2/86.2	83.6	91.3	92.9
BWN‡	1-1-4	78.3/86.5	83.1	90.9	92.9
TWS	1-1-4	**79.3/87.2**	**83.9**	**91.4**	**93.7**

Training Curves. Furthermore, we plot the training loss curves of BWN, TWN and our TWS on MRPC with data augmentation in Figures 5.8(a) and 5.8(b). Since TWS cannot inherit the previous optimizer due to the architecture change, we reset the optimizer and learning rate scheduler of BWN, TWN and TWS for a fair comparison, despite the slight increase of loss after splitting. We find that our TWS attains much lower training loss than BWN, and also surpasses TWN, verifying the advantages of fine-tuning on the wider architecture.

Optimization Trajectory. We also follow [192, 193] to visualize the optimization trajectory after splitting in Figures 5.8(c) and 5.8(d). We calculate the first two principal components of parameters in the final BinaryBERT, which are the basis for the 2-D plane. The loss contour is thus obtained by evaluating each grid point in the plane. It is found that the binary models are heading towards the optimal solution for both 8/4-bit activation quantization on the loss contour.

Exploring More Binarization Methods

We now study if there are any improved binarization variants that can directly bring better performance. Aside from BWN, we compare with LAB [194] and BiReal [47]. Meanwhile, we compare with gradual



(a) 8-bit Activation.

(b) 4-bit Activation.

(c) 8-bit Activation.

(d) 4-bit Activation.

Figure 5.8: (a) and (b) show the training curves on MRPC under different activation bits. The red box is enlarged in the sub-figure. (c) and (d) visualize the fine-tuning trajectories after splitting, on the 2-D loss contour of BinaryBERT.

quantization, i.e., BWN training based on a ternary model, denoted as BWN†. Furthermore, we also try the same scaling factor of BWN with TWN to make the precision change smooth, dubbed as BWN‡. From Table 5.7, we find that our TWS still outperforms various binarization approaches in most cases, suggesting the superiority of splitting in finding better minima than direct binary training.

5.5 Conclusion

In this chapter, we propose BinaryBERT, pushing BERT quantization to the limit. As a result of the steep and complex loss landscape, we find directly training a BinaryBERT is hard with a large performance drop. We thus propose a ternary weight splitting that splits a trained ternary BERT to initialize BinaryBERT, followed by fine-tuning for further

refinement. Our approach also supports adaptive splitting that can tailor the size of BinaryBERT based on the edge device constraints. Empirical results show that our approach significantly outperforms vanilla binary training, achieving state-of-the-art performance on BERT compression. The proposed approach also enjoys the potential to extend to other neural networks, as splitting ternary parameters to binary branches is agnostic to the network architecture. We leave this as our future work.

□ **End of chapter.**

Chapter 6

Revisiting Parameter Sharing for Neural Architecture Search

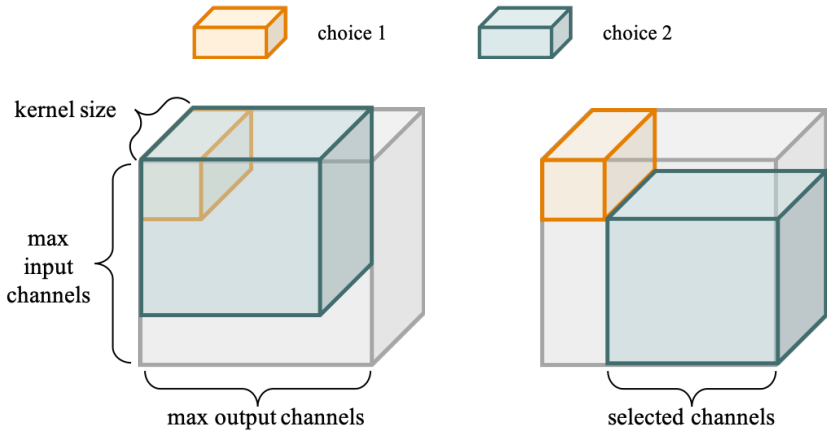
In this chapter, we shift the gear to neural architecture search (NAS), an orthogonal direction to establish efficient deep learning models. Unlike network compression and architecture splitting with manual criteria in previous chapters, NAS constructs network structures automatically from the pre-defined search space. However, the efficiency of NAS algorithms has been a major concern. Parameter sharing is widely applied to improve the NAS efficiency, which reuses parameters among different network configurations. Nevertheless, it is unclear how parameter sharing affects the searching process. Towards that end, in this chapter, we aim at a better understanding and exploitation of parameter sharing. Our analysis is based on channel number search (CNS), a fundamental problem in NAS. Specifically, we propose affine parameter sharing (APS) as a general formulation to unify and quantitatively analyze existing channel search algorithms. It is found that with parameter sharing, weight updates of one architecture can simultaneously benefit other candidates. However, it also results in less confidence in choosing good architectures. We thus propose a new strategy of parameter sharing towards a better balance between training efficiency and architecture discrimination. Extensive analysis and experiments demonstrate the superiority of the proposed strategy in channel configuration against many state-of-the-art counterparts on benchmark datasets.

6.1 Introduction

Convolutional neural networks (CNNs) have achieved great success in various areas, but substantial computational overhead limits their applications on resource-constrained platforms, *e.g.* mobile devices. To design light-weighted CNNs, neural architecture search (NAS) has been broadly adopted for channel number search (CNS) in CNNs [62, 58]. As NAS generally consumes extensive computation resources [3, 65], parameter sharing [61] is widely applied to improve the searching efficiency.

In the context of CNS, parameter sharing refers to reusing convolutional kernels of multiple network architectures. While this is intuitively believed to accelerate network training during searching [63, 58], no analysis is conducted to study its underlying mechanism. Existing parameter sharing methods largely rely on hand-crafted heuristics. For instance, as shown in Figure 6.1, ordinal selection takes channels ordinally from a shared super kernel to construct different candidates of that convolutional layer [58, 63, 64, 160, 132, 195]. Independent selection, as another common practice, instantiates different candidates of a layer as distinct trainable variables [196, 197, 198]. Though these heuristics are widely applied, it is still not well understood *how parameter sharing benefits the searching process*, and *what the potential drawback is*.

In this chapter, we formally investigate these questions. We first establish affine parameter sharing (APS), which unifies previous heuristics as applying different affine transformations on network parameters. The unified formulation facilitates quantitative analysis of the effect of parameter sharing. We thus define a metric to measure how much parameters are shared (*a.k.a. sharing level*), which is based on the cross-covariance matrix between different candidate kernels in APS. It is theoretically found that previous heuristics of ordinal sharing and independent sharing attain the maximum and minimum of the defined metric respectively. We show that a higher level of sharing accelerates the searching process (i.e. faster accuracy rise) by better aligning the gradients of different candidates. However, this also results in coupled optimization among different candidates, making architectures less discriminative. On the contrary, a lower level of sharing can better distinguish architectures but requires more iterations for searching. Therefore it remains a trade-off between the searching efficiency and architecture discrimination. Towards a better balance between the two



(a) Ordinal Selection. (b) Independent Selection.

Figure 6.1: Previous parameter sharing heuristics. The orange and green rectangles represent two different channel choices.

aspects, we propose a transitionary strategy for APS. Specifically, the sharing level is initialized at maximum such that network parameters can be rapidly optimized in the early stage. Then it is gradually annealed during the searching process such that good architectures can be better distinguished.

We conduct extensive experiments to study the effects of parameter sharing on channel number search. Besides, the transitionary sharing strategy is shown to achieve a better balance between efficient searching and architecture discrimination. Experimental results on both CIFAR-10 and ImageNet datasets show that our approach outperforms a number of competitive counterparts.

6.2 Preliminaries

6.2.1 Problem Setup

Channel number search (CNS) refers to the problem of finding the optimal channel numbers of convolutional neural networks within the computational overhead constraint. Prevalent CNS algorithms adopt a controller $\pi(\theta)$ parameterized by θ for architecture selection, as well as a super-net containing all possible architectures parameterized by $\mathbf{w}$. For a L-layer neural network, layerwise channel number decisions are sampled from $\pi(\theta)$, i.e. $a = [a_1, ..., a_L] \sim \pi(\theta)$, where $a_l \in \mathcal{A} = \{1, 2, ..., A\}$, and $\mathcal{A}$ represents the index set of channel number choices $\mathcal{C} = \{c_1, c_2, ..., c_A\}$. Here we formulate CNS based on reinforcement learning (RL) [61, 57] and

take $\pi(\theta)$ as an LSTM controller, while other NAS approaches such as gradient-based formulation [131, 58] can be similarly established. Given channel decision a and its associated parameter $\mathbf{w}(a)$, the CNS problem can be formulated as the bi-level optimization problem mentioned in Section 2.2.2, which aims to maximize the expected reward function $\mathcal{R}\Big(a, \mathbf{w}^*(\alpha)\Big)$ (e.g. the accuracy on the validation set) as follows:

$$\begin{aligned} &\max_{\theta} \ \mathbb{E}_{a\sim\pi_\theta}\mathcal{R}\Big(a, \mathbf{w}^*(a)\Big), \\ &\text{s.t.} \ \ \mathbf{w}^*(a) = \arg\min_{\mathbf{w}(a)} \mathcal{L}\Big(a, \mathbf{w}(a)\Big) \\ &\qquad \mathcal{B}\Big(\mathbf{w}(a)\Big) \leq B, \end{aligned} \tag{6.1}$$

where $\mathcal{L}\Big(a, \mathbf{w}(a)\Big)$ is the training objective such as cross-entropy, $\mathcal{B}\Big(\mathbf{w}(a)\Big)$ is the network budget function (e.g. FLOPs) and B is the budget constraint. The controller $\pi(\theta)$ is updated with policy gradient [140]. As searching with explicit constraint is infeasible, one can adopt the objective developed in [57] and penalize computation-intensive models softly in the reward function.

6.2.2 Parameter Sharing for CNS

Equation (6.1) forms a typical bi-level optimization problem. To avoid training the associated parameter $\mathbf{w}(a)$ to exact convergence before evaluating the architecture, parameter sharing [61] is widely applied in various efficient CNS algorithms. Below we summarize two commonly used sharing heuristics in CNS, which are outlined in Figure 6.1.

Ordinal Selection. [58, 63, 64, 160, 132] maintains a super kernel with a sufficiently large width for each layer. For layer l, parameters of different decisions a_l are obtained by ordinally selecting the top c_{a_l} channels from that kernel. Thus channels with lower indices are multiplexed across different width decisions.

Independent Selection. [196, 197, 198] instantiates independent convolutional kernels for each candidate $a_l \in \mathcal{A}$ in layer l. It is assumed that different channel configurations should be treated individually. Channels of different candidates are non-multiplexed in independent selection scheme.

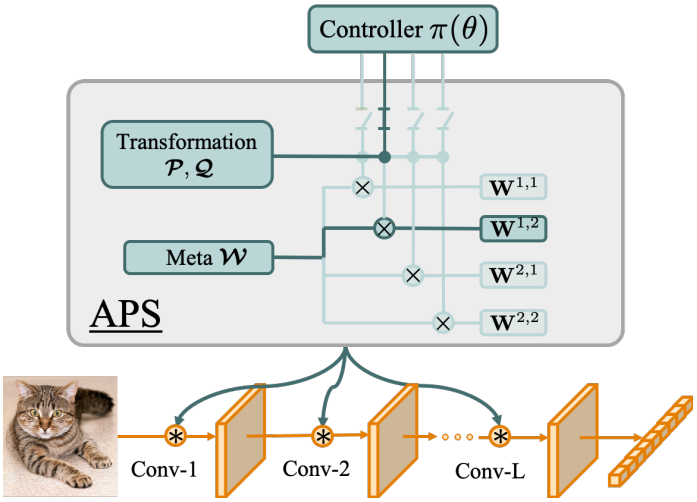


Figure 6.2: The overall framework of the proposed affine parameter sharing (APS) for CNS. We take the channel number choices $\mathcal{A} = \{1, 2\}$ for illustration.

6.3 Methodology

To investigate the role of parameter sharing for CNS, we first establish affine parameter sharing (APS), a general framework that unifies previous heuristics. Within APS, we quantitatively evaluate the level of parameter sharing, and demonstrate how it affects the searching dynamics. Based on our findings, we propose transitionary APS, a new strategy that dynamically adjusts the trade-off between efficient training and architecture discrimination. In the following discussion, we follow the standard notations: For a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{a}_x \in \mathbb{R}^m$ is the x-th column; and $a_{x,y}$ is the (x, y)-th element of $\mathbf{A}$. $\|\cdot\|_F$ refers to the Frobenius norm, and $\|\cdot\|_2$ is the l_2-norm. We denote the range of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ as $\mathcal{R}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} \,|\, \mathbf{x} \in \mathbb{R}^n\}$.

6.3.1 Affine Parameter Sharing

To facilitate the analysis of parameter sharing, a first step is to unify previous parameter sharing heuristics. Towards that end, we propose affine parameter sharing (APS), a general framework that allows flexible parameter sharing. Specifically, for each convolutional layer we maintain a meta-weight $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{c \times c \times k \times k}$ as the shared parameter pool for all candidates, where c, k are the number of filters and kernel size respectively. To transform $\boldsymbol{\mathcal{W}}$ to different sizes, we keep two sets of transformation matrices $\boldsymbol{\mathcal{P}} = \left\{\mathbf{P}^1, ..., \mathbf{P}^A\right\}$ and $\boldsymbol{\mathcal{Q}} = \left\{\mathbf{Q}^1, ..., \mathbf{Q}^A\right\}$, where

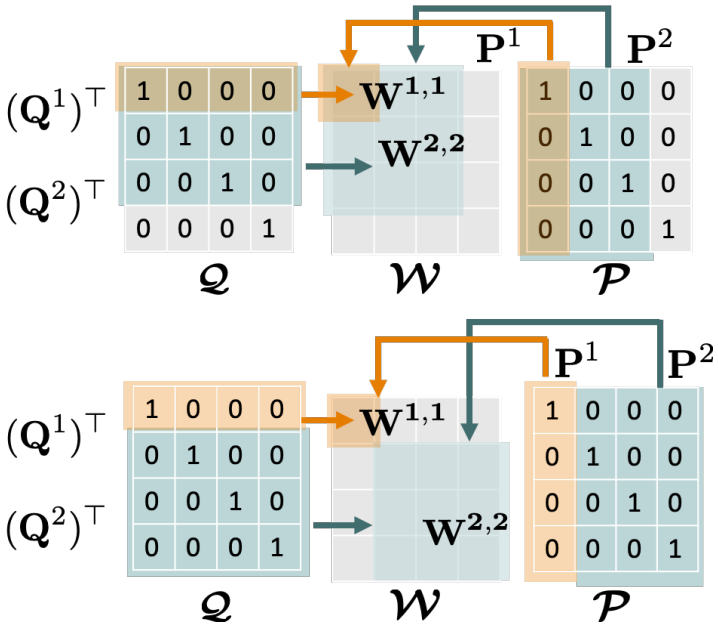


Figure 6.3: A 2-dimensional illustration of affine parameter sharing with ordinal selection (up) and independent selection (down). The candidate kernel is constructed by transforming the meta-weights into proper shapes with two transformation matrices.

$\mathbf{P}^a, \mathbf{Q}^a \in \mathbb{R}^{c \times c_a}$ $(c \geq c_a)$ are designed to be semi-orthogonal [1] such that the distinctiveness in c_a-dimensional space is maximally preserved. Given decisions on input and output width $i, o \in \mathcal{A}$, the candidate parameter $\mathbf{W}^{i,o} \in \mathbb{R}^{c_i \times c_o \times k \times k}$ can be obtained by affine transformation as:

$$\mathbf{W}^{i,o} = (\mathbf{Q}^i)^\top \times_2 \boldsymbol{\mathcal{W}} \times_1 \mathbf{P}^o, \tag{6.2}$$

where $\times_d$ denotes mode d multiplication [127], i.e, the matrix multiplication along the d-th dimension. The scheme of affine parameter sharing is visualized in Figure 6.2.

Remark. APS can be easily reduced to previous parameter sharing heuristics with different $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{Q}}$. Suppose $\{\mathbf{e}_j\}_{j=1}^c$ are standard basis in $\mathbb{R}^c$. For $\forall o, i \in \mathcal{A}$, APS is reduced to **ordinal selection** [58, 62, 63, 64, 160, 132] by choosing $\mathbf{P}^o = [\mathbf{e}_1, ..., \mathbf{e}_{c_o}]$ and $\mathbf{Q}^i = [\mathbf{e}_1, ..., \mathbf{e}_{c_i}]$. On the other hand, by taking disjoint sets of $\{\mathbf{e}_j\}_{j=1}^c$ in $\mathbf{P}^o$ or $\mathbf{Q}^i$, APS is equivalent to **independent selection** [196, 197, 198]. A 2-dimensional illustration is presented in Figure 6.3.

[1] $\mathbf{A} \in \mathbb{R}^{m \times n}$ is semi-orthogonal if $\mathbf{A}^\top \mathbf{A} = \mathbf{I}_n$ for $m > n$.

6.3.2 Quantitative Measurement

Given the formulation of APS, we are able to quantitatively measure the level of parameter sharing. For notation simplicity, we treat meta weight $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{C\times C}$ as a 2-D matrix and perform matrix multiplication.

Definition 6.3.1. *Assuming each element of meta weight $\boldsymbol{\mathcal{W}}$ follows the standard normal distribution, the **level of affine parameter sharing** of two candidate decisions (i, o) and $(\tilde{i}, \tilde{o})$ is defined as the Frobenius norm of cross-covariance matrix[2] between candidate parameters $\mathbf{W}^{i,o}$ and $\mathbf{W}^{\tilde{i},\tilde{o}}$, i.e. $\phi(i, o; \tilde{i}, \tilde{o}) = \left\| \mathrm{Cov}\left(\mathbf{W}^{i,o}, \mathbf{W}^{\tilde{i},\tilde{o}}\right) \right\|_F^2$.*

In other words, the sharing level can be quantitatively reflected by the squared sum of pairwise correlations of two candidate parameters. A large sharing level indicates high coupling between two candidates and vice versa. Taking the entire search space $\mathcal{C}$ into consideration, we are interested in the overall level of parameter sharing $\Phi = \sum_{i\leq\tilde{i}} \sum_{o\leq\tilde{o}} \phi(i, o; \tilde{i}, \tilde{o})$, as well as its maximal and minimal conditions. Without loss of generality, assuming $c_i \leq c_j$ for $i < j$, we have the following theorem:

Theorem 6.3.1. *For $\forall i \leq \tilde{i}$ and $\forall o \leq \tilde{o}$, the overall level Φ of APS is maximized if $\mathcal{R}\left(\mathbf{Q}^i\right) \subseteq \mathcal{R}\left(\mathbf{Q}^{\tilde{i}}\right)$ and $\mathcal{R}\left(\mathbf{P}^o\right) \subseteq \mathcal{R}\left(\mathbf{P}^{\tilde{o}}\right)$. Φ is minimized if $\mathcal{R}\left(\mathbf{Q}^i\right) \subseteq \mathcal{R}^{\perp}\left(\mathbf{Q}^{\tilde{i}}\right)$ and $\mathcal{R}\left(\mathbf{P}^o\right) \subseteq \mathcal{R}^{\perp}\left(\mathbf{P}^{\tilde{o}}\right)$.*

The theorem connects the level of parameter sharing with the range of transformation matrices $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{Q}}$. Notably, previous heuristics of **ordinal selection** and **independent selection** attain maximum and minimum Φ respectively. With various designs of $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{\mathcal{Q}}$, APS allows more flexible patterns of parameter sharing.

We provide a proof sketch to Thereom 6.3.1 below. We first show the case of two candidate decision (i, o) and $(\tilde{i}, \tilde{o})$, after which we can combine the pairwise optimal conditions together to yield Theorem 6.3.1. Without loss of generality, suppose $c_{\tilde{i}} > c_i$ and $c_{\tilde{o}} > c_o$, we have the following lemma:

Lemma 3. *Given candidate decisions (i, o) and $(\tilde{i}, \tilde{o})$, $\phi(i, o; \tilde{i}, \tilde{o})$ is maximized if $\mathcal{R}(\mathbf{Q}^i) \subseteq \mathcal{R}(\mathbf{Q}^{\tilde{i}})$ and $\mathcal{R}(\mathbf{P}^o) \subseteq R(\mathbf{P}^{\tilde{o}})$; $\phi(i, o; \tilde{i}, \tilde{o})$ is*

[2]The cross-covariance matrix between $\mathbf{X} \in \mathbb{R}^{m\times n}$ and $\mathbf{Y} \in \mathbb{R}^{\tilde{m}\times\tilde{n}}$ is defined as $\mathrm{Cov}(\mathbf{X}, \mathbf{Y}) = \mathbb{E}[(\mathbf{X} - \mathbb{E}(\mathbf{X})) \otimes (\mathbf{Y} - \mathbb{E}(\mathbf{Y}))^\top] \in \mathbb{R}^{m\times n\times\tilde{m}\times\tilde{n}}$, where $\otimes$ is the Kronecker product.

minimized if $\mathcal{R}(\mathbf{Q}^i) \subseteq \mathcal{R}^{\perp}(\mathbf{Q}^{\tilde{i}})$ *or* $\mathcal{R}(\mathbf{P}^o) \subseteq R^{\perp}(\mathbf{P}^{\tilde{o}})$.

Proof. As $\phi(i, o; \tilde{i}, \tilde{o})$ is defined as the squared sum of pairwise correlation between $\mathbf{W}^{i,o}$ and $\mathbf{W}^{\tilde{i},\tilde{o}}$, we can explicitly write it out as:

$$\begin{aligned}\phi(i, o; \tilde{i}, \tilde{o}) &= \sum_{x=1}^{c_i}\sum_{y=1}^{c_o}\sum_{\tilde{x}=1}^{c_{\tilde{i}}}\sum_{\tilde{y}=1}^{c_{\tilde{o}}}\left[\mathrm{Cov}_{x,y,\tilde{x},\tilde{y}}(\mathbf{W}^{i,o}, \mathbf{W}^{\tilde{i},\tilde{o}})\right]^2 \\ &= \sum_{x=1}^{c_i}\sum_{y=1}^{c_o}\sum_{\tilde{x}=1}^{c_{\tilde{i}}}\sum_{\tilde{y}=1}^{c_{\tilde{o}}}\left[\mathbb{E}\Big(W^{i,o}_{x,y}W^{\tilde{i},\tilde{o}}_{\tilde{x},\tilde{y}}\Big) - \mathbb{E}\Big(W^{i,o}_{x,y}\Big)\mathbb{E}\Big(W^{\tilde{i},\tilde{o}}_{\tilde{x},\tilde{y}}\Big)\right]^2.\end{aligned} \tag{6.3}$$

Note that the second term can be removed since $\mathbb{E}\Big(W^{i,o}_{x,y}\Big) = \mathbb{E}\Big((\mathbf{q}^i_x)^\top \boldsymbol{\mathcal{W}}\mathbf{p}^o_y\Big) = (\mathbf{q}^i_x)^\top\mathbb{E}\Big(\boldsymbol{\mathcal{W}}\Big)\mathbf{p}^o_y = 0$ and similarly $\mathbb{E}\Big(W^{\tilde{i},\tilde{o}}_{\tilde{x},\tilde{y}}\Big) = 0$. Therefore $\phi(i, o; \tilde{i}, \tilde{o})$ can be simplified as

$$\begin{aligned}\phi(i, o; \tilde{i}, \tilde{o}) &= \sum_{x=1}^{c_i}\sum_{y=1}^{c_o}\sum_{\tilde{x}=1}^{c_{\tilde{i}}}\sum_{\tilde{y}=1}^{c_{\tilde{o}}}\mathbb{E}^2\left[W^{i,o}_{x,y}\cdot W^{\tilde{i},\tilde{o}}_{\tilde{x},\tilde{y}}\right] \\ &= \sum_{x,y}\sum_{\tilde{x},\tilde{y}}\mathbb{E}^2\left[(\mathbf{q}^i_x)^\top\boldsymbol{\mathcal{W}}\mathbf{p}^o_y\cdot(\mathbf{q}^{\tilde{i}}_{\tilde{x}})^\top\boldsymbol{\mathcal{W}}\mathbf{p}^{\tilde{o}}_{\tilde{y}}\right] \\ &= \sum_{x,y}\sum_{\tilde{x},\tilde{y}}\mathbb{E}^2\left[(\mathbf{q}^i_x)^\top\boldsymbol{\mathcal{W}}\mathbf{p}^o_y\cdot(\mathbf{p}^{\tilde{o}}_{\tilde{y}})^\top\boldsymbol{\mathcal{W}}^\top\mathbf{q}^{\tilde{i}}_{\tilde{x}}\right] \\ &= \sum_{x,y}\sum_{\tilde{x},\tilde{y}}\left((\mathbf{q}^i_x)^\top\mathbb{E}\left[\boldsymbol{\mathcal{W}}\mathbf{p}^o_y\ (\mathbf{p}^{\tilde{o}}_{\tilde{y}})^\top\boldsymbol{\mathcal{W}}^\top\right]\mathbf{q}^{\tilde{i}}_{\tilde{x}}\right)^2.\end{aligned} \tag{6.4}$$

Expanding the expectation $\mathbb{E}[\boldsymbol{\mathcal{W}}\mathbf{p}^o_y(\mathbf{p}^{\tilde{o}}_{\tilde{y}})^\top\boldsymbol{\mathcal{W}}^\top]$ elementwisely, we have

$$\mathbb{E}\begin{bmatrix}\boldsymbol{w}_1\mathbf{p}^o_y(\mathbf{p}^{\tilde{o}}_{\tilde{y}})^\top\boldsymbol{w}_1^\top & \cdots & \boldsymbol{w}_1\mathbf{p}^o_y(\mathbf{p}^{\tilde{o}}_{\tilde{y}})^\top\boldsymbol{w}_c^\top \\ \vdots & \ddots & \vdots \\ \boldsymbol{w}_c\mathbf{p}^o_y(\mathbf{p}^{\tilde{o}}_{\tilde{y}})^\top\boldsymbol{w}_1^\top & \cdots & \boldsymbol{w}_c\mathbf{p}^o_y(\mathbf{p}^{\tilde{o}}_{\tilde{y}})^\top\boldsymbol{w}_c^\top\end{bmatrix} = \begin{bmatrix}(\mathbf{p}^o_y)^\top\mathbf{p}^{\tilde{o}}_{\tilde{y}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (\mathbf{p}^o_y)^\top\mathbf{p}^{\tilde{o}}_{\tilde{y}}\end{bmatrix} \tag{6.5}$$

where we have used the fact that $\mathbb{E}\Big[\boldsymbol{w}_m\mathbf{p}^o_y(\mathbf{p}^{\tilde{o}}_{\tilde{y}})^\top\boldsymbol{w}_n^\top\Big] = (\mathbf{p}^o_y)^\top\mathbb{E}\Big[\boldsymbol{w}_m^\top\boldsymbol{w}_n\Big]\mathbf{p}^{\tilde{o}}_{\tilde{y}} = (\mathbf{p}^o_y)^\top\mathbf{p}^{\tilde{o}}_{\tilde{y}}$ if $m = n$, and 0 otherwise. With Equation (6.5), Equation (6.4) can be simplified to

$$\begin{aligned}\phi(i, o; \tilde{i}, \tilde{o}) &= \sum_{x,\tilde{x}}\sum_{y,\tilde{y}}\left[(\mathbf{q}^i_x)^\top\mathbf{q}^{\tilde{i}}_{\tilde{x}}\cdot(\mathbf{p}^o_y)^\top\mathbf{p}^{\tilde{o}}_{\tilde{y}}\right]^2 \\ &= \sum_{x,\tilde{x}}\sum_{y,\tilde{y}}\left[\sum_{m=1}^{c}q^i_{m,x}q^{\tilde{i}}_{m,\tilde{x}}\cdot\sum_{n=1}^{c}p^o_{n,y}p^{\tilde{o}}_{n,\tilde{y}}\right]^2.\end{aligned} \tag{6.6}$$

Without loss of generality, we take $\mathbf{Q}^{\tilde{i}}$ and $\mathbf{P}^{\tilde{o}}$ as standard orthogonal basis, i.e. $q^{\tilde{i}}_{\tilde{x},\tilde{x}} = 1$ and $q^{\tilde{i}}_{m,x} = 0$ for $m \neq \tilde{x}$ and $\tilde{x} \in \{1, ..., c_{\tilde{i}}\}$, and

similarly for $\mathbf{P}^{\tilde{o}}$. Thus Equation (6.6) can be further reduced to

$$\sum_{x,\tilde{x}}\sum_{y,\tilde{y}}\left[q^i_{\tilde{x},x}\cdot p^o_{\tilde{y},y}\right]^2 = \sum_{x=1}^{c_i}\sum_{y=1}^{c_o}\left(\sum_{\tilde{x}=1}^{c_{\tilde{i}}}(q^i_{\tilde{x},x})^2\right)\cdot\left(\sum_{\tilde{y}=1}^{c_{\tilde{o}}}(p^o_{\tilde{y},y})^2\right) \leq \sum_x\sum_y 1 = c_i c_o. \tag{6.7}$$

The equality holds if $q^i_{m,x} = 0$ for $m > c_{\tilde{i}}$ and $p^o_{n,y} = 0$ for $n > c_{\tilde{o}}$. Therefore the maximum is attained when orthogonal basis of $\mathbf{Q}^i$ and $\mathbf{P}^o$ lie in the span of those in $\mathbf{Q}^{\tilde{i}}$ and $\mathbf{P}^{\tilde{o}}$ respectively, i.e, $\mathcal{R}(\mathbf{Q}^i) \subseteq \mathcal{R}(\mathbf{Q}^{\tilde{i}})$ and $\mathcal{R}(\mathbf{P}^o) \subseteq R(\mathbf{P}^{\tilde{o}})$.

Conversely, minimum for Equation (6.6) is attained if $q^i_{m,x} = 0$ for $m \leq c_{\tilde{i}}$ or $p^o_{n,y} = 0$ for $n \leq c_{\tilde{o}}$, which is equivalent to $\mathcal{R}(\mathbf{Q}^i) \subseteq \mathcal{R}^{\perp}(\mathbf{Q}^{\tilde{i}})$ or $\mathcal{R}(\mathbf{P}^o) \subseteq \mathcal{R}^{\perp}(\mathbf{P}^{\tilde{o}})$. □

Finally, to prove Theorem 6.3.1, we only need to extend Lemma 3 to the case of multiple candidate decisions. The maximum and minimum of $\Phi = \sum_{i,\tilde{i}}\sum_{o,\tilde{o}}\phi(i,o;\tilde{i},\tilde{o})$ can be achieved when each $\phi(i,o;\tilde{i},\tilde{o})$ attains its maximum and minimum respectively. This corresponds to the intersection of optimal conditions in Lemma 3, which is exactly Theorem 6.3.1.

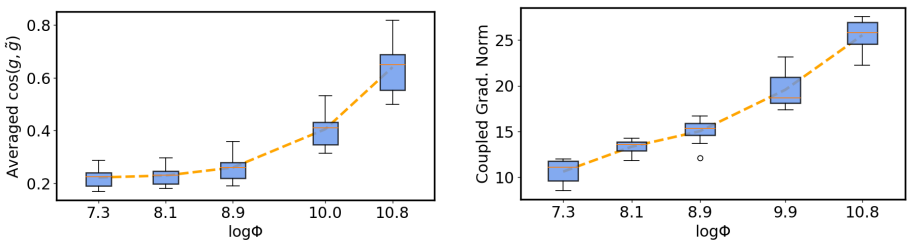
6.3.3 Parameter Sharing and the Searching Dynamics

The effects of parameter sharing can be reflected by the impact of different parameter sharing level Φ in the searching process. Specifically, we have the following observations:

Parameter Sharing Benefits Efficient Searching. We first investigate the impact of sharing level Φ on searching efficiency. Given two candidates (i,o) and $(\tilde{i},\tilde{o})$, we check the relationship between Φ and their gradients alignment on meta-weights $\boldsymbol{\mathcal{W}}$, which is computed by the cosine similarity:

$$\cos(\mathbf{g},\tilde{\mathbf{g}}) = \frac{\mathbf{g}^\top\tilde{\mathbf{g}}}{\|\mathbf{g}\|_2\cdot\|\tilde{\mathbf{g}}\|_2}, \tag{6.8}$$

where $\mathbf{g} = \nabla_{\boldsymbol{\mathcal{W}}}\mathcal{L}\left(\mathbf{W}^{i,o}\right)$ and $\tilde{\mathbf{g}} = \nabla_{\boldsymbol{\mathcal{W}}}\mathcal{L}\left(\mathbf{W}^{\tilde{i},\tilde{o}}\right)$. A positive cosine value indicates that $\nabla_{\boldsymbol{\mathcal{W}}}\mathcal{L}\left(\mathbf{W}^{i,o}\right)$ stands in the same side with $\nabla_{\boldsymbol{\mathcal{W}}}\mathcal{L}\left(\mathbf{W}^{\tilde{i},\tilde{o}}\right)$, thus the gradient update on candidate (i,o) is also a descent direction for the other configuration $(\tilde{i},\tilde{o})$. We plot the sharing level Φ against the averaged cosine similarity on a 20-layer residual network, shown in Figure 6.4(a), It can be observed that a larger Φ gives more alignment of



(a) Φ v.s. Gradient Cosines Similarity. (b) Φ v.s. Norm of Coupled Gradients.

Figure 6.4: The variation of Φ against the cosine similarity (left) and the norm of coupled gradients (right). We take the training of a 20-layer residual network for demonstration.

gradients. In other words, each gradient update simultaneously benefits multiple architectures with parameter sharing, and thus accelerates the searching process in the sense of accuracy raise.

Parameter Sharing Couples Architecture Optimization. As a side effect of efficient searching, sharing parameters inevitably couples the update of multiple architectures. We temporarily clean the notation by abbreviating the forwarding kernel $\mathbf{W}^t = \mathbf{W}^{i_t,o_t}$, transformation matrices $\mathbf{P}^t = \mathbf{P}^{o_t}$ and $\mathbf{Q}^t = \mathbf{Q}^{i_t}$ at time step t. Given decisions (i_t, o_t), the forwarding kernel can be updated as $\mathbf{W}^t = (\mathbf{Q}^t)^\top \boldsymbol{\mathcal{W}}^{t-1}\mathbf{P}^t - \eta(\mathbf{Q}^t)^\top\mathbf{Q}^{t-1}\Big(\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}^{t-1})\Big)(\mathbf{P}^{t-1})^\top\mathbf{P}^t$. For $i_t = i_{t-1}$ and $o_t = o_{t-1}$, the semi-orthogonality constraints on $\mathbf{P}_t$ and $\mathbf{Q}_t$ reduce the update to the gradient descent on the same architecture. However, when $i_t \neq i_{t-1}$ or $o_t \neq o_{t-1}$, the update from other candidates interfere the current candidate. By expanding all historical updates and re-arranging them properly, we have:

$$\begin{aligned}\mathbf{W}^t = (\mathbf{Q}^t)^\top\boldsymbol{\mathcal{W}}^0\mathbf{P}^t \\ \underbrace{-\eta\sum_{\substack{i_{\tilde{t}}=i_t\\ o_{\tilde{t}}=o_t}}(\mathbf{Q}^t)^\top\mathbf{Q}^{\tilde{t}}\Big(\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}^{\tilde{t}})\Big)(\mathbf{P}^{\tilde{t}})^\top\mathbf{P}^t}_{\text{Normal updates on the current candidate}} \\ \underbrace{-\eta\sum_{\substack{i_{\tilde{t}}\neq i_t,\text{or}\\ o_{\tilde{t}}\neq o_t}}(\mathbf{Q}^t)^\top\mathbf{Q}^{\tilde{t}}\Big(\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}^{\tilde{t}})\Big)(\mathbf{P}^{\tilde{t}})^\top\mathbf{P}^t}_{\text{Coupled updates from other candidates}}.\end{aligned} \tag{6.9}$$

In Figure 6.4(b), we compare the sharing level Φ against the Frobenius norm of coupled gradient (i.e, the third term) on the same 20-layer residual network. A larger Φ gives rise to more coupling among candi-

dates, which could make the controller less discriminative to distinguish different architectures.

6.3.4 Transitionary Strategy

With the above analysis, we see that a higher level of sharing accelerates the searching process but couples the optimization of different candidates, making them less discriminative to the controller. It is thus critical to balance these two aspects. Towards that end, we propose a transitionary strategy for APS. We initialize $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$ with ordinal selection, where Φ attains its maximum and gradually anneal it. A large Φ in the early stages quickly warms up the network, while the annealed Φ later on decouples the optimization and thus gives higher confidence to good architectures. The transition can be realized by minimizing the APS sharing level Φ with regard to $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$ as follows:

$$\begin{aligned}\min_{\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{Q}}} \ \Phi &\triangleq \sum_{i\le\tilde{i}}\sum_{o\le\tilde{o}} \left\| \operatorname{Cov}\left(\mathbf{W}^{i,o}, \mathbf{W}^{\tilde{i},\tilde{o}}\right) \right\|_F^2 \\ &= \sum_{i\le\tilde{i}}\sum_{o\le\tilde{o}} \|{\mathbf{Q}^i}^\top \mathbf{Q}^{\tilde{i}}\|_F^2 \cdot \|{\mathbf{P}^o}^\top \mathbf{P}^{\tilde{o}}\|_F^2, \\ \text{s.t. } &\left\|\mathbf{p}_x^o\right\|_2^2 = 1, \ \left\|\mathbf{q}_y^i\right\|_2^2 = 1, \\ &x \in \{1,...,c_o\}, y \in \{1,...,c_i\} \text{ and } i, o \in \mathcal{A},\end{aligned} \tag{6.10}$$

where the unit length constraints prevent the trivial zero solution. Note that the original semi-orthogonality constraints on $\mathbf{P}^o, \mathbf{Q}^i$ lead to a Stiefel manifold optimization [199] problem, which is computationally expensive. Instead, Equation (6.10) provides an feasible reformulation to the problem. We apply projected gradient descent to update $\mathbf{P}^o$ and $\mathbf{Q}^i$ as:

$$\mathbf{p}_x^o \leftarrow \Pi_{\mathcal{U}}(\mathbf{p}_x^o - \tau \nabla_{\mathbf{p}_x^o}\Phi), \ \ \mathbf{q}_y^i \leftarrow \Pi_{\mathcal{U}}(\mathbf{q}_y^i - \tau \nabla_{\mathbf{q}_y^i}\Phi), \tag{6.11}$$

where $\mathcal{U} = \{\mathbf{u} \in \mathbb{R}^C | \ \|\mathbf{u}\|_2 = 1\}$. The learning rate τ controls the transition rate of Φ. Therefore, $\boldsymbol{\mathcal{P}}$, $\boldsymbol{\mathcal{Q}}$ control the transition of parameter sharing, and $\boldsymbol{\mathcal{W}}$ is updated for the task-specific loss, both of which are optimized in a decoupled way. Note that direct optimization of the sharing level Φ could be computationally expensive. Instead, we alternatively update $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$ to minimize Φ with lower frequencies. For instance, we fix $\boldsymbol{\mathcal{P}}$ to optimize $\boldsymbol{\mathcal{Q}}$ for ten iterations, where the product terms of $\boldsymbol{\mathcal{P}}$ are computed in advance and can be reused them for multiple turns. In this way, the computational burden can be effectively reduced.

Algorithm 4 RL-based CNS algorithm with Transitionary APS.

Input:
Training data $\mathcal{D}_{tr}$, validation data $\mathcal{D}_{val}$
Base network with meta weights $\boldsymbol{\mathcal{W}}$, transformation matrices $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$
RL controller $\pi(\theta)$ and channel search space $\mathcal{C}$
Output:
Optimal channel configurations

▷ *Stage 1: fast optimization of meta-weights* $\boldsymbol{\mathcal{W}}$
Initialize $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$ with maximal Φ
for $t = 1, ..., T_1$ **do**
  Sample the architecture a uniformly from $\mathcal{C}$
  Update $\boldsymbol{\mathcal{W}}$ via gradient descent with a on $\mathcal{D}_{tr}$
end for
▷ *Stage 2: transitionary affine parameter sharing*
for $t = 1, ..., T_2$ **do**
  Sample the architecture from controller $a \sim \pi(\theta)$
  Update $\boldsymbol{\mathcal{W}}$ via gradient descent with a on $\mathcal{D}_{tr}$
  Update $\theta^{t+1} = \theta^t + \eta \mathbb{E}_a \left[\nabla_\theta \log p(a) \mathcal{R}\right]$ on $\mathcal{D}_{val}$
  Anneal Φ by updating $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$ in Equation (6.11)
end for

6.3.5 Overall Workflow

An overall workflow is shown in Algorithm 4, which consists of two stages. In the first stage, we fix the controller $\pi(\theta)$ and initialize Φ of affine parameter sharing with maximal value, so that meta weights $\boldsymbol{\mathcal{W}}$ can be efficiently optimized. During this stage, architectures are uniformly sampled from the search space $\mathcal{C}$ and are thus equally updated. In the second stage, we gradually anneal Φ via Equation (6.11) so as to transit the sharing level Φ. We alternatively update meta weights $\boldsymbol{\mathcal{W}}$ and controller parameter θ based on architectures sampled from the controller.

6.4 Experiments

In this section, we first demonstrate the effect of parameter sharing and the advantage of the proposed transitionary strategy for CNS in Section 6.4.2. Then we compare to state-of-the-art algorithms in Section 6.4.3. Code is available at https://github.com/haolibai/APS-channel-search.

6.4.1 Experimental Setup

We conduct experiments on CIFAR-10 [200] and ImageNet 2012 [7], following standard data pre-processing techniques in [109, 130]. A brief summarization of experimental setup is introduced below.

CIFAR-10 Experiments. For CIFAR-10, we take ResNet [109] as base models similar to [58, 65]. To be consistent with [58], the total searching epoch is set to 600, which can be finished within 6.9 hours for ResNet-20 and 8.6 hours for ResNet-56 on a single NVIDIA Tesla-P40. The first 200 epochs are used for warm-up training with fixed $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$, and candidate architectures are uniformly sampled from $\mathcal{C}$. The rest 400 epochs are left for the transition and training of the RL controller. We set $\mathcal{C} = \{16, 32, 64, 96\}$ for the analysis of parameter sharing in Section 6.4.2 and 100% FLOPs search, and $\mathcal{C} = \{4, 8, 16, 32, 64\}$ when searching for more compact model to compare to other baselines in Section 6.4.3. Note that in CIFAR-10 experiments all convolutional layers share the same search space, which is free from domain expertise on the search space design.

ImageNet Experiments. For ImageNet experiments, we choose ResNet-18 and MobileNet-v2 as base models. For memory efficiency, we increase candidate channels after each down-sampling layer according to default expansion rates of base models. The initial candidates $\mathcal{C}$ are set to $\{32, 48, 64, 80\}$ for ResNet18 and $\{8, 12, 16, 20\}$ for MobileNet-v2 respectively. We search for 160 epochs where the first 80 epochs are for warm-up training. The whole searching process can be finished within 24 hours for ResNet-18 and 48 hours for MobileNet-v2 on four NVIDIA Tesla-P40s.

Implementation Details. We follow the RL-based NAS algorithm for channel number search, which is previously introduced in Section 2.2. We adopt the budget-regularized reward in Equation (2.21), and use the REINFORCE [180] algorithm to update the RL controller. Following [61], we take a two-layer recurrent neural network for controller network, where the layer-wise dependencies of channel configurations can be incorporated into LSTM cells.

The detailed hyper-parameters for the RL-based CNS algorithms on both CIFAR-10 and ImageNet datasets are shown in Table 6.1. Given the searched architecture, we follow the default settings of the base models

to train stand-alone models from scratch to obtain the final performance.

Table 6.1: Hyper-parameters for different base models on CIFAR-10 and ImageNet.

Hyper-parameters	CIFAR-10		ImageNet	
	ResNet-20	ResNet-56	ResNet-18	MobileNet-v2
Channel Number $\mathcal{C}$	[4,8,16,32,64]	[4,8,16,32,64]	[32,48,64,80]	[8,12,16,20]
Width Multipliers	1	1	2	default
Max Channel Width c	208	208	128	32
Batch Size Per GPU	256	256	64	64
Init. Learning Rate of $\boldsymbol{\mathcal{W}}$	1e-1	1e-1	1e-1	5e-2
Learning Rate Decay	Stepwise	Stepwise	Cosine	Cosine
Optimizer	SGD	SGD	SGD	SGD
Momentum	0.9	0.9	0.9	0.9
Nestrov	False	False	True	True
Learning Rate of $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$	1e-3	1e-3	1e-3	1e-3
Learning Rate of θ	1.6e-4	1.6e-4	1.6e-4	1.6e-4
FLOPs Penalty α, β	0, -0.1	0, -0.06	0, -0.1	0,-0.1
Entropy Regularization	4e-3	4e-3	5e-1	4e-1
Weight Decay	2e-4	2e-4	1e-4	4e-5
Warmup Epochs	200	200	80	80
Max Epochs	600	600	160	160

6.4.2 The Effect of Parameter Sharing

We use ResNet-20 on CIFAR-10 for illustration. We denote the transitionary strategy as APS-T, and compare it against APS-O (ordinal selection) and APS-I (independent selection), which attain maximum and minimum of Φ respectively. Given no FLOPs constraint, the oracle architecture is supposed to attain the maximum channel capacity. Therefore, we perform the search without FLOPs constraint to compare how close the searched architecture is to this optimal oracle solution.

Network Optimization. To inspect the optimization dynamics, we plot the accuracy curvature of training, evaluation, the variation of averaged alignment of gradients ($\cos(\mathbf{g}, \tilde{\mathbf{g}})$) as well as the parameter sharing level Φ in Figure 6.5. It can be found that the accuracy of both ASP-O and ASP-T raises much faster than ASP-I. This indicates that the maximized parameter sharing can indeed accelerate the optimization especially during the warm-up period (first 200 epochs). From the rightmost figure, the alignment of gradients decreases with the annealed sharing level Φ when APS-T is applied. Nevertheless, it does not affect the accuracy much in late stages when APS-O, APS-I, and APS-T are mostly overlapped.

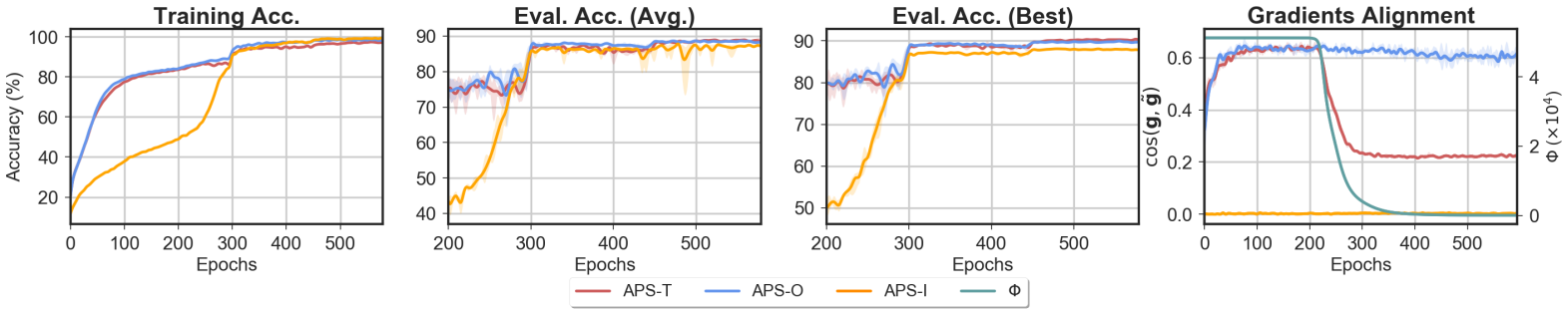


Figure 6.5: The left three figures show the Accuracy curvature with APS-O, APS-I and APS-T. For evaluation, we sample 20 architectures and report average and maximal accuracy. The rightmost figure shows the averaged alignment of gradients $(\cos(\mathbf{g}, \tilde{\mathbf{g}}))$ of APS-O, APS-I and APS-T, as well as the change of the corresponding parameter sharing level Φ for APS-T along the searching trajectory.

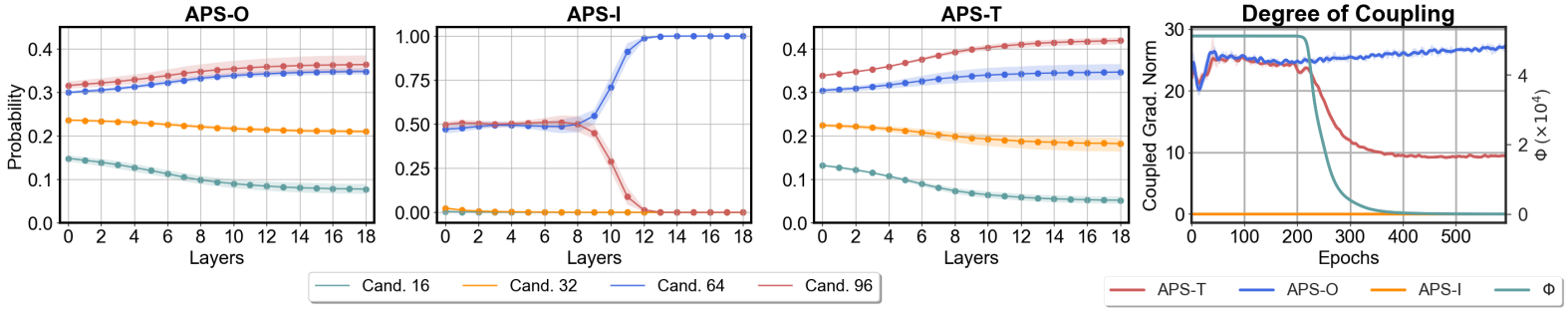


Figure 6.6: The left three figures show the layer-wise decision probabilities of controller π on ResNet-20 over the last 100 training epochs. The solid line denotes the expected logit values, and shadowed areas are 95% confidence intervals. The rightmost figure shows the averaged norm of coupled gradients of APS-O, APS-I and APS-T respectively, as well as the change of the corresponding parameter sharing level Φ for APS-T along the searching trajectory.

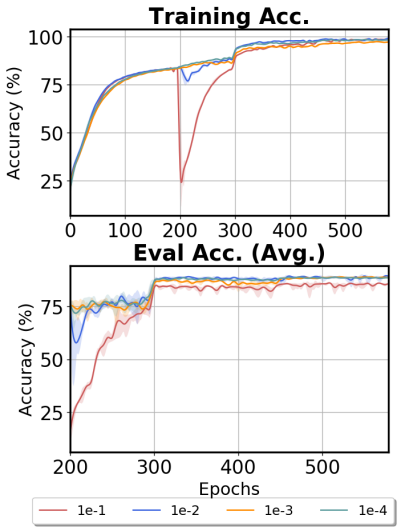


Figure 6.7: Accuracies with different learning rates of $\mathcal{P}, \mathcal{Q}$.

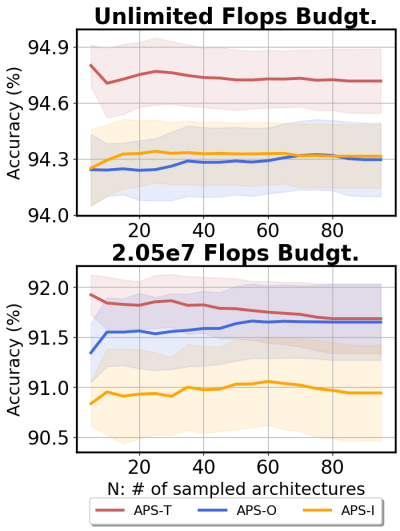


Figure 6.8: Accuracies of the top-N likely sampled models.

Architecture discrimination. The architecture discrimination can be reflected by the probabilities of layer-wise channel decisions from the controller after searching. From Figure 6.6, APS-O cannot effectively distinguish the best choice (96) and the second best choice (64) throughout all layers. APS-I separates different candidates by a large margin especially in deep layers but is sometimes stuck in incorrect local optima. This is possibly due to the incorrect reward from the insufficiently trained meta weights to the controller, such that the controller cannot explore better solution space. Finally, our APS-T confidently separates each candidate, and the optimal solution of maximum channel capacity can be stably attained across different runs. We also plot the variation of the averaged norm of coupled gradients and parameter sharing level Φ in the rightmost of Figure 6.6. The transitionary strategy successfully decreases the norm of coupled gradients that facilitates better architecture discrimination.

Transition Rate. We study the learning rate τ of $\mathcal{P}, \mathcal{Q}$ that controls the speed of the transition. We enumerate over $\tau \in \{$1e-1, 1e-2, 1e-3, 1e-4$\}$, and the converged values of Φ are: 2.78e4, 2.20e3, 2.70e2, 5.05e4 respectively. We first plot accuracy curvatures in Figure 6.7. It can be found that a large τ (e.g. 1e-1) makes the transition non-smooth, which leads to a sudden drop of accuracy. Then we plot the probabilities of layerwise channel decisions under different learning rates in Figure 6.9. We see that either large or small τ cannot separate different architectures with insufficiently optimized Φ, while only a proper learning rate (e.g.

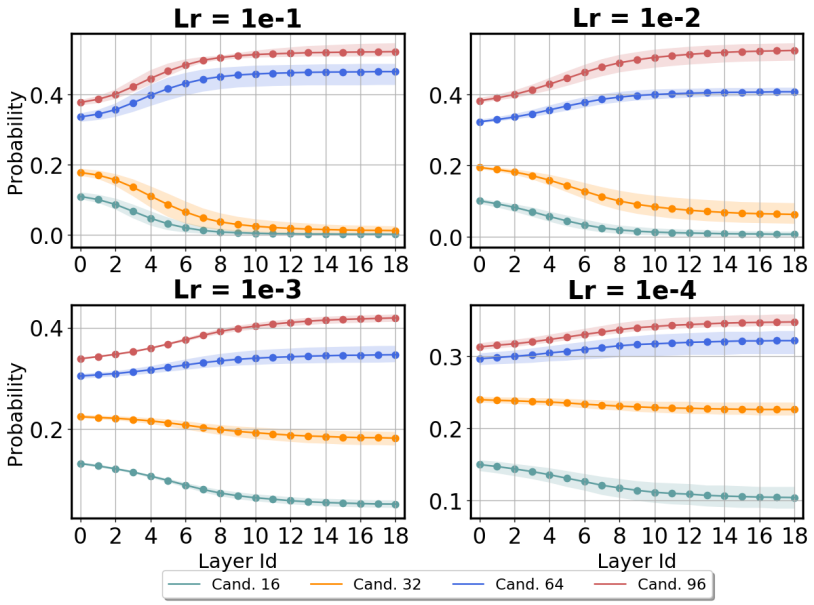


Figure 6.9: Probabilities of layer-wise channel decisions with different learning rates of $\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{Q}}$.

1e-3) produces more discriminative candidates with lower Φ.

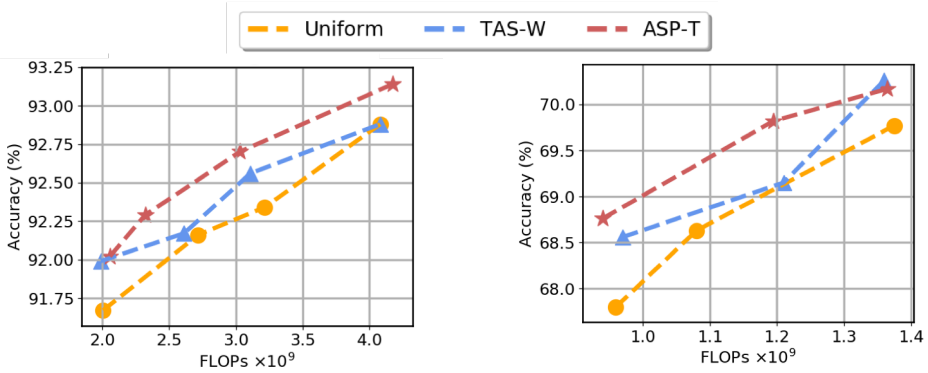
Sampled Architectures. For practical deployment, it is necessary to check the overall qualities of sampled architectures. We generate top-N architectures via beam search according to the sequence likelihood by the controller, and train them from scratch to obtain the final performance. We conduct both unlimited FLOPs budget search and 2.05e7 FLOPs constraint (∼50% of the original ResNet-20) search, and plot the accumulated accuracies of top-N likely models in Figure 6.8. Both mean and standard deviation are reported. It can be found that given no FLOPs constraint, APS-T clearly outperforms APS-O and APS-I, as the top-N models of APS-T distribute around the optimal solution. Under FLOPs constraint, APS-T still outperforms the other two. The improvement decreases as N increases, indicating that with APS-T, better architectures can be picked with higher probabilities.

6.4.3 Comparisons with state-of-the-arts

Finally, we compare APS-T against many state-of-the-art methods ranging from hand-crafted (HC) channel pruning [25, 201, 202], to automatic (Auto) channel search [58, 62, 63]. We perform the search under different FLOPs constraints, and then apply beam search to generate top-N architectures according to the sequence likelihood from

Table 6.2: Comparison of different algorithms for ResNet-20 and ResNet-56 on CIFAR10. Drops↓ denotes the decrease of accuracy comparing to base models, and Ratio↓ is the reduction of FLOPs. - stands for unavailable records. * denotes the results reported with knowledge distillation/depth search and † indicates results reported with pre-trained model, both of which are absent in our model.

Methods	Types	ResNet-20				ResNet-56			
		Accuracy	Drop↓	FLOPs	Ratio↓	Accuracy	Drop↓	FLOPs	Ratio↓
Original	-	92.78%	-	40.8M	0.0%	94.28%	-	126.0M	0.0%
CP [25]	*HC*	-	-	-	-	91.80%	1.00%	62.9M	50.0%
LCCL [201]	*HC*	91.68%	1.06%	26.1M	36.0%	92.81%	1.54%	78.1M	37.9%
SFP [202]	*HC*	90.83%	1.37%	24.3M	42.2%	93.35%	0.24%	59.4M	52.6%
FPGM [203]	*HC*	91.09%	1.11%	24.3M	42.2%	92.93%	0.66%	59.4M	52.6%
FPGM† [203]	*HC*	-	-	-	-	93.49%	0.10%	59.4M	52.6%
AMC [65]	*Auto*	-	-	-	-	91.90%	0.90%	62.9M	50.0%
TAS-W [58]	*Auto*	91.99%	0.89%	19.9M	51.3%	92.87%	1.59%	63.1M	49.9%
TAS-W* [58]	*Auto*	92.31%	0.57%	19.9M	51.3%	93.69%	0.77%	59.5M	52.7%
APS-O	*Auto*	91.61%	1.17%	19.1M	53.4%	92.93%	1.35%	57.7M	53.9%
APS-I	*Auto*	91.24%	1.54%	21.5M	47.5%	92.85%	1.43%	57.8M	53.8%
APS-T	*Auto*	92.02%	0.76%	20.6M	49.6%	93.42%	0.86%	60.3M	51.8%
APS-T	*Auto*	93.14%	-0.36%	41.7M	-2.3%	94.54%	-0.26%	122.5M	2.7%



(a) ResNet-20@CIFAR-10. (b) ResNet-18@ImageNet.

Figure 6.10: Comparison under different FLOPs constraint.

the RL controller. We choose the architecture closest to the target FLOPs from the top-N candidates, and train it from scratch to obtain the final performance. The results on CIFAR-10 and ImageNet are shown in Table 6.2 and Table 6.3 respectively.

From Table 6.2, APS-T discovers architectures with better or comparable performance under various FLOPs constraints. Note that this is achieved with all layers sharing the same set of candidate widths, without domain expertise on the design of search space. Moreover, searching with 100% FLOPs of ResNet-20 and ResNet-56 increase 0.36% and 0.26% accuracy respectively comparing to base models. Finally, APS-T also outperforms APS-I and APS-O by a clear margin, demonstrating the

Table 6.3: Comparison for ResNet-18 and MobileNet-V2 on ImageNet. * denotes original results with knowledge distillation.

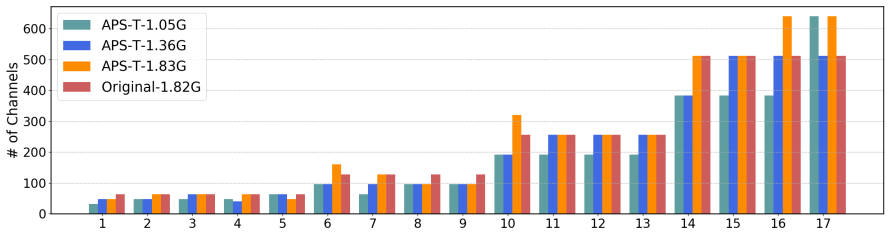
Methods	Types	Top-1 Acc	Top-5 Acc	FLOPs	Ratio↓
Resnet-18 [109]	-	69.76%	89.08%	1.82G	0.0%
LCCL [201]	*HC*	66.33%	86.94%	1.19G	34.6%
SFP [202]	*HC*	67.10%	87.78%	1.06G	41.8%
FPGM [203]	*HC*	68.41%	88.48%	1.06G	41.8%
TAS [58]	*Auto*	69.15%	89.19%	1.21G	33.3%
APS-O	*Auto*	68.60%	88.44%	1.04G	42.9%
APS-I	*Auto*	68.32%	88.21%	1.05G	41.8%
APS-T	*Auto*	69.34%	88.89%	1.05G	41.8%
APS-T	*Auto*	70.17%	89.59%	1.36G	24.9%
APS-T	*Auto*	71.67%	90.36%	1.83G	-0.9%
MobileNet-V2 [130]	-	71.80%	91.00%	314M	0.0%
×0.65 scaling	*HC*	67.20%	-	140M	55.4%
MetaPrune [62]	*Auto*	68.20%	-	140M	53.3%
MetaPrune [62]	*Auto*	72.70%	-	300M	4.4%
AutoSlim [63]	*Auto*	72.49%	90.50%	305M	2.9%
AutoSlim* [63]	*Auto*	74.20%	-	305M	2.9%
APS-O	*Auto*	72.58%	90.76%	316M	-0.6%
APS-I	*Auto*	72.38%	90.54%	311M	1.0%
APS-T	*Auto*	68.96%	88.48%	156M	50.3%
APS-T	*Auto*	72.83%	90.75%	314 M	0.0%

effectiveness of the proposed transitionary method.

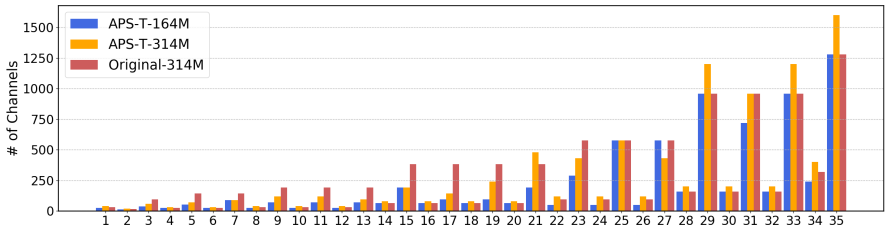
For ResNet-18 on ImageNet, APS-T consistently outperforms baselines under similar computational FLOPs. For instance, it achieves 69.34% top-1 accuracy given 41.8% reduction of FLOPs, surpassing the most competitive TAS by 0.19%. For MobileNet-v2, it achieves more than 0.7% gain of accuracy with only 3% more FLOPs comparing to MetaPrune, and higher accuracy to MetaPrune and AutoSlim without knowledge distillation under 314M FLOPs. Meanwhile, APS-T surpasses APS-I and APS-O under similar FLOPs constraints on both models.

To further demonstrate APS-T, we draw the accuracy curvature of ResNet-20 and ResNet-18 against different FLOPs constraints in Figure 6.10. It can be found that our APS-T mostly outperforms TAS as well as uniform scaling strategy on both base networks.

Visualization of Channel Configurations. Finally, we visualize the searched channel configurations under different FLOPs constraints of ResNet-18 and MobileNet-v2 in Figure 6.11. Note that for MobileNet-v2, we omit the output channels of the depth-wise convolution in each block since it equals the input channels. It can be observed that for both ResNet-18 and MobileNet-v2 under various FLOPs constraints, APS-T tends to find configurations with fewer channels in front layers and more



(a) ResNet-18.



(b) MobileNet-v2.

Figure 6.11: Channel configurations of ResNet-18 and MobileNet-v2 under different FLOPs constraint.

channels in deep layers.

6.5 Conclusion

In this chapter, we pioneer to provide analysis of the effect of parameter sharing in automatic channel number search, and conduct preliminary research on exploiting the strength of parameter sharing and avoiding its weakness. We first propose a general framework named affine parameter sharing (APS) to unify previous parameter sharing heuristics. Then we quantitatively measure APS and demonstrate how it affects the searching process. Empirical results show that parameter sharing brings efficient searching of network parameters but also results in less discrimination of architectures. Thus we are motivated to propose the transitionary APS strategy, such that a proper balance between searching efficiency and architecture discrimination can be achieved. Extensive experiments and analysis are conducted to demonstrate the effectiveness of our strategy.

□ **End of chapter.**

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we explore to develop efficient deep learning models by network compression and neural architecture search. Specifically, our works are spanned by the three common challenges in the paradigm of efficient deep learning, as introduced in Chapter 1. The first two works study network pruning and quantization given limited training resources such as data and computation devices. The third work, on the other hand, explores the limit of quantization under adequate training resources. In the last work, we shift to neural architecture search, and analyze the role of parameter sharing in prevalent NAS methods. Detailed summaries of each chapter are presented in the following:

- In Chapter 3, we present cross distillation for few-shot network pruning, a novel layer-wise knowledge distillation approach for pruning when only a limited number of training instances are available. The proposed method is especially suitable for situations when data security and privacy are of high priority. By reducing estimation errors between the student network and teacher network, cross distillation can bring a more powerful and generalizable student network. Extensive experiments on benchmark datasets demonstrate the superiority of our method against various competitive baselines.

- In Chapter 4, we study post-training quantization for pre-trained language models under limited training resources. We show that existing quantization-aware training solutions suffer from slow training, huge memory consumption, and privacy issues of accessing

the full training set. To mitigate these issues, we propose module-wise reconstruction error minimization, an efficient solution to quantize PLMs. MREM can be conducted either sequentially or in parallel, where the parallel training can achieve speed up close to the theoretical limit without apparent performance degradation. Experimental results show that our proposed solution can achieve comparable results to QAT, but significantly reduces the training time and memory overhead with only thousands of training instances.

- In Chapter 5, we propose BinaryBERT, pushing BERT quantization to the limit. As a result of the steep and complex loss landscape, we find directly training a BinaryBERT is hard with a large performance drop. We thus propose a ternary weight splitting that splits a trained ternary BERT to initialize BinaryBERT, followed by fine-tuning for further refinement. Our approach also supports adaptive splitting that can tailor the size of BinaryBERT based on the edge device constraints. Empirical results show that our approach significantly outperforms vanilla binary training, achieving state-of-the-art performance on BERT compression.
- In Chapter 6, we study the effect of parameter sharing, which is an orthogonal direction to develop efficient deep learning models. Based on channel number search problems, we conduct preliminary research on the strength and weaknesses of parameter sharing. We first propose a general framework named affine parameter sharing (APS) to unify previous parameter sharing heuristics. Then we quantitatively measure APS and demonstrate how it affects the searching process. Empirical results show that parameter sharing brings efficient searching of network parameters but also results in less discrimination of architectures. Thus we are motivated to propose the transitionary APS strategy, such that a proper balance between searching efficiency and architecture discrimination can be achieved. Extensive experiments and analysis are conducted to demonstrate the effectiveness of our strategy.

7.2 Future Work

Network compression and neural architecture search are still popular research directions that are actively studied in recent years. For network compression, we believe that compressing models with limited training

resources will always be practical and necessary in the industry. At the same time, with the growing size of deep learning models, it is also urgent to extend network compression techniques at larger scales. In terms of neural architecture search, while numerous research works focus on developing better searching algorithms, there are relatively fewer efforts paid to the fundamental challenges in performance estimation and design of search space, both of which also play important roles in NAS systems. More concretely, we plan to explore the following directions in the near future:

- **Network Compression with Domain Adaptation.** An interesting direction would be combine domain adaptation [204, 205, 125, 206] for network compression under limited training resources. In domain adaptation, a source domain with abundant training data can provide auxiliary information for the target domain where the model is trained and compressed. This can be helpful when access to the target domain is restricted due to security issues. Apart from that, domain adaptation can also benefit model compression from the pipeline of pre-training and then fine-tuning, where downstream tasks generally have fewer training instances, while the pre-training domain contains oceans of irrelevant training instances and is thus computationally expensive. This is especially true for recent pre-trained language models applied for natural language understanding tasks [11, 207, 208]. Hence it would be promising if a related domain with abundant training data can be found to assist the compression. For now, there are few preliminary attempts in this direction [209, 210], and the problem is far from being solved. For instance, it still remains space to design better strategies to fuse knowledge from different domains. Meanwhile, the role of the uncompressed model is also seldom studied in domain adaptation based network compression.

- **Network Compression for Trillion-scale Models.** The fast development of very recent deep learning models have scaled up to billion to trillion parameters, such as DeBERTA [208] for natural language understanding, GPT-3 [18] and PanGu-α [20] for language generation, as well as cross-modal networks such as CLIP [19] and DALL·E [18]. Despite the universal power of these gigantic models, these models suffer greatly from the high response latency and oceans of computing infrastructure, which is even hard for training and deployment on the cloud. Consequently, it remains

urgent to apply model compression techniques to these trillion-scale models. However, the trillion-scale models fundamentally challenge the conventional compression pipeline. For instance, both the forward and backward pass of trillion scale models rely on the combination of various parallel strategies [184, 185, 211, 212], which should be considered in the compression algorithms. In the meanwhile, the trillion parameters together with the pre-training corpus make it prohibited to fine-tune the compressed network again, which should be done in normal-sized models. Network compression for trillion-scale models is still an open challenge for the research community for now.

- **Architecture Search with Improved Performance Estimation.** To design efficient deep learning models with neural architecture search, it is a fundamental yet under-explored direction to improve the performance estimation strategy in NAS algorithms. As analyzed in Chapter 6, existing NAS approaches rely on parameter sharing to improve the searching efficiency but at the cost of less architecture discrimination. Inspired by recent progress in few-shot learning [213, 214], we wonder whether any fast training for the architecture can make itself better distinguished. For instance, a model-specific proxy dataset can be constructed at each searching iteration, which only contains a few labeled training instances for fast-training of the model. The trained architecture is then evaluated, providing more informative signals to the controller optimization. Recently, few-shot NAS is proposed [215] to adopt multiple supernets to partition the search space, such that the co-adaption in one-shot NAS can be alleviated. This also opens the door for more works that explore the balance between accurate performance estimation and searching efficiency.

- **Architecture Search with Refined Search Space.** Finally, we highlight that the design of search space also plays a key role in NAS algorithms. The widely used search space in prevalent approaches [61, 2, 59] still contains up to 1.3×10^{11} candidates, which challenges the training of the controller. On the contrary, the NAS algorithm can be greatly simplified when armed with a well-designed search space. For instance, empirical observations find that a good search space can be constructed based on a linear function, where any regular network (a.k.a. RegNet) performs reasonably well [70]. Another potential solution to simplify the

search space is via progressive searching [60, 41]. One may first search for the types of operations and connections, followed by the network width, and finally the network depth. The greedy procedure can better identify the ideal configuration on each dimension of the network architecture, and their combination thus may lead to more powerful structures.

- **Further Combination of Network Compression and Architecture Search.** Most existing efforts along this direction exploit the layer-wise sensitivity to pruning or quantization and search for the best compression configuration [65, 68, 63, 216]. However, there is still room for further research. First, as network compression mostly starts from a well-trained model, it remains explorable to design parameter sharing for efficient searching that does not break the knowledge within the original model. Second, as NAS algorithms usually train the architecture from scratch after searching, it is against the pipeline of "compression and then fine-tuning" in network compression. Therefore it would be beneficial to unify both searching and fine-tuning in a single round when combining NAS with compression. Third, current research mostly simulates the network redundancy by either computational FLOPs or the latency on proxy devices [68, 216]. However, these measurement can be inaccurate when deployed on edge devices [217]. Hence it would be promising to search for hard-ware aware neural architectures directly on the edge.

Chapter 8

Publications during Ph.D. Study

8.1 Published Conference Papers

1. **Haoli Bai**, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, Irwin King. *BinaryBERT: Pushing the Limit of BERT Quantization.* In Proceedings of the 59th Conference of the Association for Computational Linguistics, Long Paper, 2021.

2. Xianghong Fang*, **Haoli Bai***, Jian Li, Zenglin Xu, Michael Lyu, Irwin King. *Discrete Auto-regressive Variational Attention Models for Text Modeling.* In Proceedings of International Joint Conference on Neural Networks, Long Paper, 2021. * indicates equal contributions.

3. Jiaxing Wang*, **Haoli Bai***, Jiaxiang Wu, Xupeng Shi, Junzhou Huang, Irwin King, Michael Lyu, Jian Cheng. *Revisiting Parameter Sharing for Automatic Neural Channel Number Search.* Advances in Neural Information Processing Systems, 2020. * indicates equal contributions.

4. **Haoli Bai**, Jiaxiang Wu, Irwin King, Michael Lyu. *Few Shot Network Compression via Cross Distillation.* In Proceedings of the 34th AAAI Conference on Artificial Intelligence, 2020.

5. **Haoli Bai**, Zhuangbin Chen, Irwin King, Michael Lyu, Zenglin Xu. *Neural Relational Topic Models for Scientific Article Analysis.* In Proceedings of the 27th International Conference on Information

and Knowledge Management, Long Paper, 2018.

8.2 Preprints

1. **Haoli Bai**, Lu Hou, Lifeng Shang, Irwin King, Michael Lyu. *Towards Efficient Post-training Quantization of Pre-trained Language Models*. In preparation to Transactions of the Association for Computational Linguistics, 2021.

2. **Haoli Bai**, Jiaxiang Wu, Mingyang Yi, Irwin King, Michael Lyu. *Cross Distillation: A Unified Approach for Few-shot Network Compression*. In preparation to IEEE Transactions on Neural Networks and Learning Systems, 2021.

3. Chung Yiu Yau, **Haoli Bai**, Michael Lyu, Irwin King. *DAP-BERT: Differentiable Architecture Pruning of BERT*. Submitted to the 28th International Conference on Neural Information Processing, 2021.

□ **End of chapter.**

Bibliography

[1] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, pp. 1–21, 2019.

[2] Hanxiao Liu, Karen Simonyan, and Yiming Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[3] Barret Zoph and Quoc V. Le, "Neural architecture search with reinforcement learning," in *Proceedings of the International Conference on Learning Representations*, 2017.

[4] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan, "A survey on evolutionary neural architecture search," Preprint arXiv:2008.10937, 2020.

[5] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang, "Slimmable neural networks," in *International Conference on Representation Learning*, 2019.

[6] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu, "Ternarybert: Distillation-aware ultra-low bit bert," in *Conference on Empirical Methods in Natural Language Processing*, 2020.

[7] Deng Jia, Dong Wei, Socher Richard, Li Li-Jia, Li Kai, and Fei-Fei Li, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248—-255.

[8] 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.

[9] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2014.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin,

"Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.

[12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[13] Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Ekaterina Gonina *et al.*, "State-of-the-art speech recognition with sequence-to-sequence models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2018, pp. 4774–4778.

[14] Hao Wang, Naiyan Wang, and Dit-Yan Yeung, "Collaborative deep learning for recommender systems," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 1235–1244.

[15] Hao Wang, Xingjian Shi, and Dit-Yan Yeung, "Relational deep learning: A deep latent variable model for link prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[16] Haoli Bai, Zhuangbin Chen, Michael R Lyu, Irwin King, and Zenglin Xu, "Neural relational topic models for scientific article analysis," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 27–36.

[17] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, 2020.

[18] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever, "Zero-shot text-to-image generation," Preprint arXiv:2102.12092, 2021.

[19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark *et al.*, "Learning transferable visual models from natural language supervision," *arXiv preprint arXiv:2103.00020*, 2021.

[20] Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang *et al.*, "Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation," Preprint arXiv:2104.12369, 2021.

[21] Ji Wang, Bokai Cao, Philip Yu, Lichao Sun, Weidong Bao, and Xiaomin Zhu, "Deep learning towards mobile applications," in *International Conference on Distributed Computing Systems*, 2018, pp. 1385–1393.

[22] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017.

[23] Jie Tang, Dawei Sun, Shaoshan Liu, and Jean-Luc Gaudiot, "Enabling deep learning on iot devices," *Computer*, vol. 50, no. 10, pp. 92–96, 2017.

[24] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2722–2730.

[25] 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. 1398–1406.

[26] Jianhao Luo, Jianxin Wu, and Weiyao Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the International Conference on Computer Vision*, 2017, pp. 5058–5066.

[27] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 875–886.

[28] 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.

[29] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," Preprint arXiv:1606.06160, 2016.

[30] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," Tech. Rep. arXiv:1805.06085, 2018.

[31] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha, "Learned step size quantization," in *International Conference on Learning Representations*, 2019.

[32] Yuhang Li, Xin Dong, and Wei Wang, "Additive powers-of-two quantization: A non-uniform discretization for neural networks," in *International Conference on Learning Representations*, 2020.

[33] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio, "Fitnets: Hints for thin deep nets," in *International Conference on Learning Representations*, 2015.

[34] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," Preprint arXiv:1503.02531, 2015.

[35] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1943–1955, 2015.

[36] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *International Conference on Learning Representations*, 2016.

[37] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations*, 2016.

[38] Antonio Polino, Razvan Pascanu, and Dan Alistarh, "Model compression via distillation and quantization," in *International Conference on Learning Representations*, 2018.

[39] Michael Zhu and Suyog Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," in *International Conference on Learning Representations workshop*, 2018.

[40] Paul Michel, Omer Levy, and Graham Neubig, "Are sixteen heads really better than one?" in *Advances in Neural Information Processing Systems*, 2019.

[41] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu, "Dynabert: Dynamic bert with adaptive width and

depth," in *Advances in Neural Information Processing Systems*, 2020.

[42] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proceedings of the International Conference on Machine Learning*, 2017.

[43] Xin Dong, Shangyu Chen, and Sinno Jialin Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 2017, pp. 4857–4867.

[44] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry, "Scalable methods for 8-bit training of neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 5145–5153.

[45] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Conference on European Conference on Computer Vision*, 2016.

[46] Xiaofan Lin, Cong Zhao, and Wei Pan, "Towards accurate binary convolutional neural network," in *Advances in Neural Information Processing Systems*, 2017.

[47] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng, "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," in *Conference on European Conference on Computer Vision*, 2018.

[48] Hyungjun Kim, Kyungsu Kim, Jinseok Kim, and Jae-Joon Kim, "Binaryduo: Reducing gradient mismatch in binary activation network by coupling binary activations," in *International Conference on Learning Representations*, 2019.

[49] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng, "Reactnet: Towards precise binary neural network with generalized activation functions," in *European Conference on Computer Vision*, 2020, pp. 143–159.

[50] Tianhong Li, Jianguo Li, Zhuang Liu, and Changshui Zhang, "Knowledge distillation from few samples," Preprint arXiv:1812.01839, 2018.

[51] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu, "Tinybert: Distilling bert for natural language understanding," in *Findings of Empirical Methods in Natural Language Processing*, 2020.

[52] Sergey Zagoruyko and Nikos Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," in *International Conference on Representation Learning*, 2017.

[53] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi, "Knowledge transfer via distillation of activation boundaries formed by hidden neurons," in *Proceedings of the AAAI conference on Artificial Intelligence*, 2019.

[54] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu, "Learning compact recurrent neural networks with block-term tensor decomposition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9378–9387.

[55] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," in *International Conference on Representation Learning*, 2015.

[56] Guangxi Li, Jinmian Ye, Haiqin Yang, Di Chen, Shuicheng Yan, and Zenglin Xu, "Bt-nets: Simplifying deep neural networks via block term decomposition," Preprint arXiv:1712.05689, 2017.

[57] 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.

[58] Xuanyi Dong and Yi Yang, "Network pruning via transformable architecture search," in *Advances in Neural Information Processing Systems*, 2019, pp. 760–771.

[59] Han Cai, Ligeng Zhu, and Song Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *Proceedings of the International Conference of Representation Learning*, 2019.

[60] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *International Conference on Representation Learning*, 2020.

[61] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean, "Efficient neural architecture search via parameter sharing," in *Proceedings of the Proceedings of the International Conference on Machine Learning*, 2018, pp. 4092–4101.

[62] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, 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, pp. 3296–3305.

[63] Jiahui Yu and Thomas S. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," Preprint arXiv:1903.11728, 2019.

[64] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12 962–12 971, 2020.

[65] 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 *Proceedings of the European Conference on Computer Vision*, 2018, pp. 815–832.

[66] Jiaxiang Wu, Yao Zhang, Haoli Bai, Huasong Zhong, Jinlong Hou, Wei Liu, and Junzhou Huang, "Pocketflow: An automated framework for compressing and accelerating deep neural networks," in *Advances in Neural Information Processing Systems, Workshop on Compact Deep Neural Networks with Industrial Applications*, 2018.

[67] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2078–2087.

[68] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[69] Dms: Differentiable dimension search for binary neural networks, "Dms: Differentiable dimension search for binary neural networks," in *International Conference on Learning Representations, 1st Workshop on Neural Architecture Search*, 2020.

[70] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár, "Designing network design spaces," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 428–10 436.

[71] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin, "SNAS: stochastic neural architecture search," in *Proceedings of the International Conference on Learning Representations*, 2019.

[72] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2017, pp. 2902–2911.

[73] Hanxiao Liu, Andrew Brock, Karen Simonyan, and Quoc V Le, "Evolving normalization-activation layers," Preprint arXiv:2004.02967, 2020.

[74] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2014, pp. 387–395.

[75] SN Sivanandam and SN Deepa, "Genetic algorithms," in *Introduction to genetic algorithms*. Springer, 2008, pp. 15–37.

[76] William B Langdon and Riccardo Poli, *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[77] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[78] Gabriel Bender, "Understanding and simplifying one-shot architecture search," in *Proceedings of the Proceedings of the International Conference on Machine Learning*, 2019, pp. 549–558.

[79] Yann LeCun, John S Denker, and Sara A Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

[80] Babak Hassibi, David G Stork, and Gregory J Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*, 1993, pp. 293–299.

[81] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representations*, 2017.

[82] Jonathan Frankle and Michael Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proceedings of the International Conference on Machine Learning*, 2018.

[83] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov, "Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned," in *Annual Meeting of the Association for Computational Linguistics*, 2019.

[84] Bin Dai, Chen Zhu, Baining Guo, and David Wipf, "Compressing neural networks using the variational information bottleneck," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2018, pp. 1135–1144.

[85] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," in *Advances in Neural Information Processing Systems*, 2017.

[86] Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez, "Structured variational learning of bayesian neural networks with horseshoe priors," in *Proceedings of the International Conference on Machine Learning*, 2018, pp. 1744–1753.

[87] Jiaxing Wang, Haoli Bai, Jiaxiang Wu, and Jian Cheng, "Bayesian automatic model compression," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 727–736, 2020.

[88] Fanxu Meng, Hao Cheng, Ke Li, Huixiang Luo, Xiaowei Guo, Guangming Lu, and Xing Sun, "Pruning filter in filter," in *Advances in Neural Information Processing Systems*, 2020.

[89] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang, "Collaborative channel pruning for deep networks," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2019, pp. 5113–5122.

[90] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner, "Data-free knowledge distillation for deep neural networks," Preprint arXiv:1710.07535, 2017.

[91] Hanting Chen, Yunhe Wang, Chang Xu, Zhaohui Yang, Chuanjian Liu, Boxin Shi, Chunjing Xu, Chao Xu, and Qi Tian, "Dafl: Data-free learning of student networks," in *Proceedings of the International Conference on Computer Vision*, 2019, pp. 3514–3522.

[92] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer, "Zeroq: A novel zero shot quantization framework," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 169–13 178.

[93] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz, "Dreaming to distill: Data-free knowledge transfer via deepinversion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8715–8724.

[94] Jian-Hao Luo and Jianxin Wu, "Neural network pruning with residual-connections and limited-data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1458–1467.

[95] Fengfu Li, Bo Zhang, and Bin Liu, "Ternary weight networks," Preprint arXiv:1605.04711, 2016.

[96] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally, "Trained ternary quantization," in *International Conference on Learning Representations*, 2017, pp. 4299–4307.

[97] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016.

[98] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[99] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang, "Improving neural network quantization without retraining using outlier channel splitting," in *Proceedings of the International Conference on Machine Learning*, 2019.

[100] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak, "Lsq+: Improving low-bit quantization through learnable offsets and better initialization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 696–697.

[101] Karen Ullrich, Edward Meeds, and Max Welling, "Soft weight-sharing for neural network compression." in *International Conference on Learning Representations*, 2017.

[102] Daisuke Miyashita, Edward H Lee, and Boris Murmann, "Convolutional neural networks using logarithmic data representation," Tech. Rep. arXiv:1603.01025, 2016.

[103] Jingyong Cai, Masashi Takemoto, and Hironori Nakajo, "A deep look into logarithmic quantization of model parameters in neural networks," in *Proceedings of the 10th International Conference on Advances in Information Technology*, 2018, pp. 1–8.

[104] Lu Hou and James T Kwok, "Loss-aware weight quantization of deep networks," in *International Conference on Learning Representations*, 2018.

[105] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 365–382.

[106] Yuhang Li, Xin Dong, Sai Qian Zhang, Haoli Bai, Yuanpeng Chen, and Wei Wang, "Rtn: Reparameterized ternary network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4780–4787.

[107] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan, "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4852–4861.

[108] Yinghao Xu, Xin Dong, Yudian Li, and Hao Su, "A main/subsidiary network framework for simplifying binary neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7154–7162.

[109] 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.

[110] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos, "Training binary neural networks with real-to-binary convolutions," in *International Conference on Learning Representations*, 2020.

[111] Adrian Bulat, Brais Martinez, and Georgios Tzimiropoulos, "High-capacity expert binary networks," Preprint arXiv:2010.03558, 2020.

[112] 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.

[113] Yu Bai, Yu-Xiang Wang, and Edo Liberty, "Proxquant: Quantized neural networks via proximal operators," in *International Conference on Learning Representations*, 2019.

[114] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat, "Q8bert: Quantized 8bit bert," Preprint arXiv:1910.06188, 2019.

[115] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H Hassoun, "Post-training piecewise linear quantization for deep neural networks," in *European Conference on Computer Vision*, 2020, pp. 69–86.

[116] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," Preprint arXiv:2005.03842, 2020.

[117] Yury Nahshan, Brian Chmiel, Chaim Baskin, Evgenii Zheltonozhskii, Ron Banner, Alex M Bronstein, and Avi Mendelson, "Loss aware post-training quantization," Preprint arXiv:1911.07190, 2019.

[118] Peisong Wang, Qiang Chen, Xiangyu He, and Jian Cheng, "Towards accurate post-training network quantization via bit-split

and stitching," in *Proceedings of the International Conference on Machine Learning*, 2020, pp. 9847–9856.

[119] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort, "Up or down? adaptive rounding for post-training quantization," in *Proceedings of the International Conference on Machine Learning*, 2020, pp. 7197–7206.

[120] Denny Zhou, Mao Ye, Chen Chen, Tianjian Meng, Mingxing Tan, Xiaodan Song, Quoc Le, Qiang Liu, and Dale Schuurmans, "Go wide, then narrow: Efficient training of deep thin networks," in *Proceedings of the International Conference on Machine Learning*, 2020, pp. 11 546–11 555.

[121] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[122] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4133–4141.

[123] Pengpeng Liu, Michael Lyu, Irwin King, and Jia Xu, "Selflow: Self-supervised learning of optical flow," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4571–4580.

[124] Pengpeng Liu, Irwin King, Michael R Lyu, and Jia Xu, "Ddflow: Learning optical flow with unlabeled data distillation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 8770–8777.

[125] Kuo Zhong, Ying Wei, Chun Yuan, Haoli Bai, and Junzhou Huang, "Translider: Transfer ensemble learning from exploitation to exploration," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 368–378.

[126] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, 2014, pp. 675–678.

[127] Tamara G Kolda and Brett W Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[128] 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.

[129] 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," Preprint arXiv:1704.04861, 2017.

[130] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," Preprint arXiv:1801.04381, 2018.

[131] Hanxiao Liu, Karen Simonyan, and Yiming Yang, "Darts: Differentiable architecture search," in *Proceedings of the International Conference of Representation Learning*, 2019.

[132] 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*, 2020, pp. 544–560.

[133] Jiaxing Wang, Jiaxiang Wu, Haoli Bai, and Jian Cheng, "M-NAS: meta neural architecture search," in *Proceedings of the AAAI conference on Artificial Intelligence*, 2020, pp. 6186–6193.

[134] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju, "Fastbert: a self-distilling bert with adaptive inference time," in *Annual Meeting of the Association for Computational Linguistics*, 2020.

[135] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin, "Deebert: Dynamic early exiting for accelerating bert inference," in *Annual Meeting of the Association for Computational Linguistics*, 2020.

[136] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei, "Bert loses patience: Fast and robust inference with early exit," in *Advances in Neural Information Processing Systems*, 2020.

[137] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin, "Dsnas: Direct neural architecture search without parameter retraining," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 084–12 092.

[138] David So, Quoc Le, and Chen Liang, "The evolved transformer," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2019, pp. 5877–5886.

[139] Benoît Colson, Patrice Marcotte, and Gilles Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, no. 1, pp. 235–256, 2007.

[140] Ronald J Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[141] 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, no. 01, 2019, pp. 4780–4789.

[142] Jian Ren, Zhe Li, Jianchao Yang, Ning Xu, Tianbao Yang, and David J Foran, "Eigen: Ecologically-inspired genetic approach for neural network structure searching from scratch," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9059–9068.

[143] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen, "Completely automated cnn architecture design based on blocks," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 4, pp. 1242–1254, 2019.

[144] Aditya Rawal and Risto Miikkulainen, "From nodes to networks: Evolving recurrent neural networks," Preprint arXiv:1803.04439, 2018.

[145] Xiaoqi Jiao, Huating Chang, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu, "Improving task-agnostic bert distillation with layer mapping search," Preprint arXiv:2012.06153, 2020.

[146] James Kennedy and Russell Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[147] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *International joint conference on artificial intelligence*, 2015.

[148] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter, "Learning curve prediction with bayesian neural networks," in *International Conference on Learning Representations*, 2016.

[149] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik, "Accelerating neural architecture search using performance prediction," in *Advances in Neural Information Processing Systems, Workshop on Meta-Learning*, 2017.

[150] 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.

[151] Dahyun Kim, Kunal Pratap Singh, and Jonghyun Choi, "Learning architectures for binary networks," in *European Conference on Computer Vision*, 2020, pp. 575–591.

[152] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens, "Net2net: Accelerating learning via knowledge transfer," in *International Conference on Learning Representations*, 2016.

[153] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter, "Simple and efficient architecture search for convolutional neural networks," in *Advances in Neural Information Processing Systems, Workshop on Meta-Learning*, 2017.

[154] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu, "Path-level network transformation for efficient architecture search," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2018, pp. 678–687.

[155] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang, "Efficient architecture search by network transformation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[156] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *International Conference on Representation Learning*, 2019.

[157] Qiang Liu, Lemeng Wu, and Dilin Wang, "Splitting steepest descent for growing neural architectures," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[158] Dilin Wang, Meng Li, Lemeng Wu, Vikas Chandra, and Qiang Liu, "Energy-aware neural architecture optimization with fast splitting steepest descent," Preprint arXiv:1910.03103, 2019.

[159] Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu, "Firefly neural architecture descent: a general approach for growing neural networks," in *Advances in Neural Information Processing Systems*, 2020.

[160] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu, "Single-path nas: Designing hardware-efficient convnets in less than 4 hours," Preprint arXiv:1904.02877, 2019.

[161] Liangjiang Wen, Xuanyang Zhang, Haoli Bai, and Zenglin Xu, "Structured pruning of recurrent neural networks through neuron selection," *Neural Networks*, pp. 134–141, 2020.

[162] Kartikeya Bhardwaj, Naveen Suda, and Radu Marculescu, "Dream distillation: A data-independent model compression framework," Preprint arXiv:1905.07072, 2019.

[163] Shangyu Chen, Wenya Wang, and Sinno Jialin Pan, "Deep neural network quantization via layer-wise optimization using limited training data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 3329–3336.

[164] Michael P Friedlander and Paul Tseng, "Exact regularization of convex programs," *SIAM Journal on Optimization*, vol. 18, no. 4, pp. 1326–1350, 2007.

[165] Neal Parikh, Stephen Boyd *et al.*, "Proximal algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[166] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell, "Rethinking the value of network pruning," in *International Conference on Representation Learning*, 2019.

[167] Hou Pong Chan, Wang Chen, Lu Wang, and Irwin King, "Neural keyphrase generation via reinforcement learning with adaptive rewards," in *Annual Meeting of the Association for Computational Linguistics*, 2019.

[168] Wang Chen, Hou Pong Chan, Piji Li, and Irwin King, "Exclusive hierarchical decoding for deep keyphrase generation," in *Annual Meeting of the Association for Computational Linguistics*, 2020.

[169] Wenxiang Jiao, Xing Wang, Zhaopeng Tu, Shuming Shi, Michael R Lyu, and Irwin King, "Self-training sampling with monolingual data uncertainty for neural machine translation," in *Annual Meeting of the Association for Computational Linguistics*, 2021.

[170] Angela Fan, Edouard Grave, and Armand Joulin, "Reducing transformer depth on demand with structured dropout," in *International Conference on Learning Representations*, 2019.

[171] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," Preprint arXiv:1910.01108, 2019.

[172] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu, "Patient knowledge distillation for bert model compression," in *Conference on Empirical Methods in Natural Language Processing*, 2019.

[173] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser, "Universal transformers," in *International Conference on Learning Representations*, 2019.

[174] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut, "Albert: A lite bert for self-supervised learning of language representations," in *International Conference on Learning Representations*, 2020.

[175] Jiaxing Wang, Haoli Bai, Jiaxiang Wu, Xupeng Shi, Junzhou Huang, Irwin King, Michael Lyu, and Jian Cheng, "Revisiting parameter sharing for automatic neural channel number search," in *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[176] Zhiqi Huang, Lu Hou, Lifeng Shang, Xin Jiang, Xiao. Chen, and Qun Liu, "Ghostbert: Generate more features with cheap operations for bert," in *Annual Meeting of the Association for Computational Linguistics*, 2021.

[177] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer, "Q-bert: Hessian based ultra low precision quantization of bert," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[178] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King, "Binarybert: Pushing the limit of bert quantization," in *Annual Meeting of the Association for Computational Linguistics*, 2021.

[179] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu, "Brecq: Pushing the limit of post-training quantization by block reconstruction," in *International Conference on Learning Representations*, 2021.

[180] Ronald J Williams and David Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[181] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry, "Improving post training neural quantization: Layer-wise calibration and integer programming," in *Proceedings of the International Conference on Machine Learning*, 2021.

[182] Qirong Ho, James Cipar, Henggang Cui, Jin Kyu Kim, Seunghak Lee, Phillip B Gibbons, Garth A Gibson, Gregory R Ganger, and Eric P Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in Neural Information Processing Systems*, 2013, p. 1223.

[183] Haoli Bai, Jiaxiang Wu, Irwin King, and Michael Lyu, "Few shot network compression via cross distillation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3203–3210.

[184] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in neural information processing systems*, 2018.

[185] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[186] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2018.

[187] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," Preprint arXiv:1804.07461, 2018.

[188] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang, "Squad: 100,000+ questions for machine comprehension of text," Preprint arXiv:1606.05250, 2016.

[189] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin, "Training with quantization noise for extreme fixed-point compression," Preprint arXiv:2004.07320, 2020.

[190] Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song, "A tensorized transformer for language modeling," in *Advances in Neural Information Processing Systems*, 2019.

[191] Pranav Rajpurkar, Robin Jia, and Percy Liang, "Know what you don't know: Unanswerable questions for squad," Preprint arXiv:1806.03822, 2018.

[192] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein, "Visualizing the loss landscape of neural nets," in *Advances in Neural Information Processing Systems*, 2018.

[193] Yaru Hao, Li Dong, Furu Wei, and Ke Xu, "Visualizing and understanding the effectiveness of BERT," in *Conference on Empirical Methods in Natural Language Processing*, 2019.

[194] Lu Hou, Quanming Yao, and James T Kwok, "Loss-aware binarization of deep networks," in *International Conference on Learning Representations*, 2017.

[195] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu, "Single-path NAS: designing hardware-efficient convnets in less than 4 hours," in *Machine Learning and Knowledge Discovery in Databases - European Conference*, 2019, pp. 481–497.

[196] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang, "Densely connected search space for more flexible neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 625–10 634.

[197] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang, "Blockwisely supervised neural architecture search with knowledge distillation," Preprint arXiv:1911.13053, 2019.

[198] Junran Peng, Ming Sun, ZHAO-XIANG ZHANG, Tieniu Tan, and Junjie Yan, "Efficient neural architecture transformation search in channel-level for object detection," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 290–14 299.

[199] Zaiwen Wen and Wotao Yin, "A feasible method for optimization with orthogonality constraints," *Mathematical Programming*, vol. 142, no. 1-2, pp. 397–434, 2013.

[200] Alex Krizhevsky and Geffery Hinton, "Learning multiple layers of features from tiny images," in *Technical report*, 2009.

[201] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan, "More is less: A more complicated network with less inference complexity," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5840–5848.

[202] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018, pp. 2234–2240.

[203] 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.

[204] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang, "Domain adaptation via transfer component analysis," *IEEE transactions on neural networks*, vol. 22, no. 2, pp. 199–210, 2010.

[205] Yaroslav Ganin and Victor Lempitsky, "Unsupervised domain adaptation by backpropagation," in *Proceedings of the International Conference on Machine Learning*, 2015, pp. 1180–1189.

[206] Xianghong Fang, Haoli Bai, Ziyi Guo, Bin Shen, Steven Hoi, and Zenglin Xu, "Dart: domain-adversarial residual-transfer networks for unsupervised cross-domain image classification," *Neural Networks*, vol. 127, pp. 182–192, 2020.

[207] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," Preprint arXiv:1907.11692, 2019.

[208] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen, "Deberta: Decoding-enhanced bert with disentangled attention," in *International Conference on Representation Learning*, 2021.

[209] Shangyu Chen, Wenya Wang, and Sinno Jialin Pan, "Cooperative pruning in cross-domain deep neural network compression," in *Proceedings of the International Joint Conference on Artificial Intelligence.*, 2019, pp. 2102–2108.

[210] Jianfei Yang, Han Zou, Shuxin Cao, Zhenghua Chen, and Lihua Xie, "Mobileda: Toward edge-domain adaptation," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6909–6918, 2020.

[211] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[212] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[213] Chelsea Finn, Pieter Abbeel, and Sergey Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the Proceedings of the International Conference on Machine Learning*. PMLR, 2017, pp. 1126–1135.

[214] Jake Snell, Kevin Swersky, and Richard S. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4077–4087.

[215] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo, "Few-shot neural architecture search," in *Proceedings of the International Conference on Machine Learning*, 2021.

[216] Yuhang Li, Wei Wang, Haoli Bai, Ruihao Gong, Xin Dong, and Fengwei Yu, "Efficient bitwidth search for practical mixed precision neural network," Preprint arXiv:2003.07577, 2020.

[217] Yuhang Li, Mingzhu Shen, Jian Ma, Yan Ren, Mingxin Zhao, Qi Zhang, Ruihao Gong, Fengwei Yu, and Junjie Yan, "Mqbench: Towards reproducible and deployable model quantization benchmark," in *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.