Enabling Deep Learning at Edge:

From Efficient and Dynamic Inference to On-Device Learning

by

Li Yang

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved June 2023 by the
Graduate Supervisory Committee:

Deliang Fan, Chair
Jae-sun Seo
Junshan Zhang
Yu Cao

ARIZONA STATE UNIVERSITY

August 2023

©2023 Li Yang

All Rights Reserved

ABSTRACT

In recent years, Artificial Intelligence (AI) (e.g., Deep Neural Networks (DNNs), Transformer) has shown great success in real-world applications due to its superior performance in various cognitive tasks. The impressive performance achieved by AI models normally accompanies the cost of enormous model size and high computational complexity, which significantly hampers their implementation on resource-limited Cyber-Physical Systems (CPS), Internet-of-Things (IoT), or Edge systems due to their tightly constrained energy, computing, size, and memory budget. Thus, the urgent demand for enhancing the **Efficiency** of DNN has drawn significant research interests across various communities. Motivated by the aforementioned concerns, this doctoral research has been mainly focusing on **Enabling Deep Learning at Edge: From Efficient and Dynamic Inference to On-Device Learning**.

Specifically, from the inference perspective, this dissertation begins by investigating a hardware-friendly model compression method that effectively reduces the size of AI model while simultaneously achieving improved speed on edge devices. Additionally, due to the fact that diverse resource constraints of different edge devices, this dissertation further explores dynamic inference, which allows for real-time tuning of inference model size, computation, and latency to accommodate the limitations of each edge device. Regarding efficient on-device learning, this dissertation starts by analyzing memory usage during transfer learning training. Based on this analysis, a novel framework called “Reprogramming Network” (Rep-Net) is introduced that offers a fresh perspective on the on-device transfer learning problem. The Rep-Net enables on-device transfer learning by directly learning to reprogram the intermediate features of a pre-trained model. Lastly, this dissertation studies an efficient continual learning algorithm that facilitates learning multiple tasks without the risk of forgetting

previously acquired knowledge. In practice, through the exploration of the task correlation, an interesting phenomenon is observed that the intermediate features are highly correlated between tasks with the self-supervised pre-trained model. Building upon this observation, a novel approach called progressive task-correlated layer freezing is proposed to gradually freeze a subset of layers with the highest correlation ratios for each task leading to training efficiency.

DEDICATION

This thesis is dedicated to my father & mother.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Deliang Fan at Arizona State University for its high-quality advising and mentoring. I am grateful for his invaluable suggestions and ideas, which helped form my research structure and directions. I would commend his patience and help me develop as a researcher through continuous encouragement and appreciation. In addition, I sincerely thank the graduate advisory committee Dr. Jae-sun Seo, Dr.Junshan Zhang, and Dr. Yu (Kevin) Cao, for their support and valuable suggestions. Also, thanks to all of my co-workers and their contributions to forming this thesis. I want to thank Zhezhi, Adnan, and my other colleagues who have given their valuable expertise and suggestions in helping me throughout my Ph.D.

This Ph.D. work is supported in part by the National Science Foundation under Grant No.2003749, No.1931871, and No. 2144751

TABLE OF CONTENTS

Page

LIST OF TABLES . . . x

LIST OF FIGURES . . . xiii

CHAPTER

1 INTRODUCTION . . . 1

1.1 Overview . . . 1

1.2 Statement of the Main Problems . . . 3

1.3 Contributions . . . 5

1.3.1 Contribution 1: Hardware-friendly Deep Neural Network Compression . . . 6

1.3.2 Contribution 2: Dynamic Inference with Run-Time Tuning of Model Size, Computation and Latency . . . 8

1.3.3 Contribution 3: On-Device Learning for Task Adaption . . . 9

1.3.4 Contribution 4: Efficient Continual Learning via Progressive Layer Freezing . . . 10

1.4 Dissertation Structure . . . 12

2 HARDWARE-FRIENDLY MODEL COMPRESSION AND ACCELERATION . . . 14

2.1 Basics of Model Compression . . . 15

2.1.1 Weight Quantization . . . 15

2.1.2 Pruning . . . 16

2.2 Proposed Model Compression Method . . . 19

2.2.1 Problem Formalization . . . 19

2.2.2 Weight Penalty Clipping with Self-adapting Threshold . . . 20

2.2.3 PE-wise Structured Pruning ... 22
2.3 Experiments ... 24
2.3.1 Experiment Setup ... 24
2.3.1.1 Dataset and Network Structure ... 24
2.3.1.2 Experiment Platform ... 25
2.3.1.3 Compression Rate ... 25
2.3.2 Experimental Results ... 26
2.3.2.1 CIFAR-10 Experiment ... 26
2.3.2.2 ImageNet Experiment ... 27
2.4 Ablation Study and Discussion ... 29
2.4.1 Evolution of Clipping Threshold ... 30
2.4.2 PE Capacity vs. Structured Sparsity Ratio ... 30
2.4.3 Relation to the Norm-based Criterion ... 31
2.4.4 FPGA Implementation ... 32
2.5 Summary ... 32
3 DYNAMIC DEEP NEURAL NETWORK WITH RUN-TIME TUNING OF ACCURACY AND LATENCY ... 34
3.1 Bascis of Sparse Training ... 35
3.1.1 In-training Sparsification ... 35
3.2 What is Dynamic Inference ... 38
3.3 Sparse Train More at Once ... 39
3.3.1 Rationale of Alternating Sparse Training (AST) ... 40
3.3.2 AST with Gradient Correction ... 41
3.3.3 Sparse Sub-net Training ... 43

CHAPTER Page

3.4 Experiments 46

3.4.1 Experimental Setup 46

3.4.2 Experimental Results 47

3.4.2.1 CIFAR-10/100 47

3.4.2.2 ImageNet 48

3.4.2.3 Extend to Structured Fine-grained Sparsity 49

3.5 Ablation Study and Discussion 50

3.5.1 AST-GC with More Sub-nets 50

3.5.2 AST Sub-nets with Different Sparsity Differences 51

3.5.3 Extended AST Training Efforts 52

3.6 Summary 53

4 ON-DEVICE LEARNING FOR TASK ADAPTION 54

4.1 Preliminaries 54

4.1.1 Memory Efficient Learning 54

4.1.2 Transfer Learning 56

4.1.2.1 Transfer Learning via Fine-tuning 56

4.1.2.2 Transfer Learning via Input Reprogramming 57

4.2 Training Memory Analysis 58

4.2.1 Fine-tuning and Adaptor-based Methods 58

4.2.2 Mask-based Learning Method 59

4.2.3 Training Memory Usage Analysis. 60

4.3 On-device Learning via Feature Reprogramming 64

4.3.1 Architecture Overview 64

4.3.2 Activation Connector 66

CHAPTER Page

4.3.3 Reprogramming Step 67

4.3.4 Design Intuition 67

4.3.4.1 Design Intuition 1 68

4.3.4.2 Design Intuition 2 68

4.3.4.3 Design Intuition 3 70

4.4 Experiments 73

4.4.1 Experimental Setup 73

4.4.1.1 Datasets and Networks. 73

4.4.1.2 Training details 73

4.4.1.3 Evaluation Metric 73

4.4.2 Main Results 74

4.4.2.1 Comparison with Input Reprogramming Methods 74

4.4.2.2 Comparison with State-of-the-art Methods 75

4.5 Ablation Study and Discussion 75

4.5.1 Does Rep-Net Transfer Better by Using Better ImageNet Models? 75

4.5.2 Can Rep-Net Combine with Other Transfer Learning Methods for Efficient Inference? 77

4.6 Summary 78

5 EFFICIENT CONTINUAL LEARNING VIA TASK-CORRELATED PROGRESSIVE LAYER FREEZING 79

5.1 Related Works and Background 80

5.1.1 Self-supervised Learning 80

5.1.2 Continual Learning 82

CHAPTER Page

5.1.3 Layer Freezing 84

5.2 Efficient Self-supervised Continual Learning 84

5.2.1 Problem Formulation 84

5.2.2 Progressive Task-correlated Layer Freezing 85

5.2.2.1 Overview 85

5.2.2.2 Layer Freezing via Task Correlation 86

5.2.3 Subspace Construction via Memory Replay Data. 88

5.2.3.1 Progressive Task-correlated Freezing 89

5.3 Experimental Results 91

5.3.1 Experimental Setup 91

5.3.2 Main Results 93

5.4 Ablation Study and Discussion 96

5.4.1 Task-correlated Layer Freezing vs. Ascending Order Layer Freezing 96

5.4.2 Self-supervised Layer Decision 97

5.5 Summary 98

6 CONCLUSION AND OUTLOOK 99

REFERENCES 101

LIST OF TABLES

Table Page

1. Inference Accuracy (%) of Resnets on Cifar-10. 26
2. Simulation Result of Structured Pruning for Alexnet on Imagenet Dataset. FP. Indicates Full-precision (32-bit Floating-point) and Tern. Indicates Ternary Weights. Note That, Acc. And Comp. Are the Abbreviation of Accuracy and Compression. 27
3. Ablation Study on CIFAR-10 28
4. Inference Accuracy (%) of ResNet18 on ImageNet 28
5. Ablation Study of ResNet-18 [41] on ImageNet. 30
6. PE Capacity Versus Structured Sparsity Ratio of ResNet-20 Trained on CIFAR-10 Dataset. 31
7. Convolutional Layer Implementation Setup and Speed Up on FPGA 32
8. Different Relationships Between Three Subnets G_i, G_j, and G_k: Completely Subset (CS) and Non-disjoint (ND). 43
9. Averaged Accuracy and Standard Deviations Between Different AST Training Schemes on CIFAR-10 Dataset (Three Times Experiments Each). 44
10. CIFAR-10/100 Accuracy and Training Cost Comparison with SoTA Works on Wide ResNet-32. 47
11. ImageNet Accuracy and Training Cost Comparison with SoTA Works on ResNet-50. 49
12. Inference Acceleration and Negligible Accuracy Drop of the Proposed AST Algorithm with Structured Fine-grained Sparsity on ResNet-18 Model. 49
13. AST with Extended Training Effort on CIFAR-10 with Wide ResNet-32 [127]. 51

Table Page

14. ResNet-18 Training Results of AST (160 Epochs) with Various Sparsity Values and Numbers of Sub-nets for CIFAR-10 Dataset. 51

15. ImageNet Accuracy with Different Sparsity Combinations on ResNet-50. . . . 52

16. Summary of The Parameters and Activation Memory Consumption of Different Layers. 61

17. The Ablation Study to Validate the Four Design Choices. 'adv Rep + Last' Is the Adversarial Reprogramming Combining with Re-training Last Classifier. The 6^{Th} Row Shows the Results for Proposed Rep-Net. The Last Row (7^{Th}) Shows the Result for Using Dual Connector at All the Convolution Layer for the Backbone Model. We Use the Imagenet Pre-trained Proxylessnas-mobile as the Backbone Model. 69

18. Comparison with Input Reprogramming Works. 'adv. Rep' Is the Original Adversarial Reprogramming Work; 'adv Rep + Last' Is the Improved Adversarial Reprogramming Work That Further Re-train the Last Classifier. 72

19. Comparison with Previous State-of-the-art (Sota) Transfer Learning Methods Using Different Backbone Neural Networks, Where'i-v3' Is Inception-v3; 'n-a' Is Nasnet-a Mobile; 'm2-1.4' Is Mobilenetv2-1.4; 'r-50' Is Resnet-50; 'pm' Is Proxylessnas-mobile. In This Table, We Show Our Improvements in Comparison to Best Existing Transfer Learning Scheme Tinytl. 74

20. Combining the Rep-Net with Learnable Binary Mask-based Method for Efficient Inference. 'tinytl-last' Means Only Re-training the Last Classifier on the Tinytl Imagenet Pre-trained Model. The Inference Flops Is Reported on Flowers Dataset. 74

Table Page

21. Accuracy and Forgetting of the Learned Representations on Split Cifar-10, Split CIFAR-100, and Split Tiny-imagenet on Resnet-18 Architecture with KNN Classifier. All the Values Are Measured by Computing Mean and Standard Deviation Across Three Trials. Note That, We Use the Layer Freezing Ratio As 0.4 by Default for All Our Results. 94
22. Training Time (Measured Time in Nvidia A4000 GPU), Memory Cost, And Computation Flops of the Learned Representations on Split CIFAR-10, Split Cifar-100 and Split Tiny-imagenet. 95
23. Accuracy, Forgetting, Training Time, and Training Memory Cost of the Learned Representations on Imagenet-100 with Linear Evaluation by Using Barlowtwin and Mocov2 Respectively. 95
24. The Ablation Study on the Proposed Method in Comparison to Layer Freezing in Ascending Layer Index (I.E., "Top Layer") Order on Both Split Cifar-10 and Cifar-100 Datasets by Using Barlowtwin as Backbone Method. 96

LIST OF FIGURES

Figure | Page

1. General Flow of Enabling Deep Learning at The Edge. 2
2. The Development Trend of The Deep Learning Models. 3
3. Hardware Resources of Various Devices. 4
4. Running Times of Same Model on Various Android Phones. 5
5. Stitched Ternarized Kernel (Each Kernel Is in 3×3) of the First Layer of Resnet-20 on Cifar-10, with (Left) Non-structured and (Right) Structured Pruning. Patterns in {White,Grey,Black} Denotes {-1,0,+1} Respectively. 17
6. Sample Weight Distribution of Resnet-20 Layer under Different Training Configurations, with Mean (μ) and Standard Deviation (σ). Baseline: Full-precision Model Without Compression; Group-lasso Prune: Structured Weight Pruning Utilizing Group Lasso Technique; Weight Tern.: Weight Ternarization; Naive Combine: Naively Combining the Aforementioned Pruning and Ternarization; This Work: Combining Pruning and Ternarization with Proposed Weight Penalty Clipping Technique. 19
7. The Overview of Weight Penalty Clipping with Self-adapting Threshold. 21
8. PE-wise Pruning. 22
9. Evolution of Clipping Threshold and Sparsity 29
10. Norm-based Criterion on ResNet20 for CIFAR-10 dataset. (Left) Is the Conv8 Layer and (Right) Is Conv15 Layer. 31
11. Alternating Sparse Training (AST): The Subset Network (Sub-net) are Iteratively Activated Throughout the Training. The Model Only Learns the Active Connections of Each Sub-net, Leading to the One-time-Trained Multiple Sub-nets. 39

Figure Page

12. The Overview of the AST Process With Gradient Correction on Consecutive Inner-group Sub-nets (AST-GC) . . . 41
13. The Ratio of Negative and Positive Inner Product of Two Sub-nets During the AST Process on CIFAR-10 by Using Wide ResNet-32. . . . 42
14. Layer-wise Sparsity and the Connection Dissimilarity Between Two ResNet-32 Sub-nets Trained by the Non-disjoint (ND) AST Scheme. . . . 45
15. An Example of Adapting ResNet50 (Pre-trained on ImageNet dataset) to Flower Dataset. *Top:* Model Parameters and Activation Memory of Three Different Methods. *Bottom:* Training Time of One Epoch on Two Different Platforms: One Powerful GPU (Quadro RTX 5000) and One Edge GPU (Jetson Nano) . . . 60
16. The Workflow of Adversarial Reprogramming (a) and the Rep-Net (b). Adversarial Reprogramming Reprograms the Input by Introducing An Additive Learnable Parameter. Differently, Rep-Net Takes the Input Data Directly and Learns to Reprogram the Intermediate Activation Features. . . 63
17. The Overview of Reprogramming Network (Rep-Net). The Proposed Rep-Net Model Learns Directly From the Input. The input Image is Directly Ded Into Both Rep-Net and the Backbone Model Parallelly. In Addition, Rep-Net Consists of A Small Number of Layers Positioned at Specific Locations Where the Backbone Model Observes a Feature Resolution Reduction. 65
18. Design Choices for Feature Exchange Operation in Activation Connector. . . 70
19. The Test Accuracy Vs Learning Epochs under Four Difference Design Choices. 71
20. The Accuracy Comparison Between Pre-trained Imagenet and Transfer to CIFAR10, CUB, Aircraft and Flowers on Four Different Models. . . . 76

Figure Page

21. The Overview of Our Proposed Method Which Progressively Freezes Partial Layers During the Whole Training Process for Each Task. 86

22. The Final Selection of Updated Layer for Each Task. The Freezing Ratios Are 0.3 And 0.5 Respectively. Note That, Each Blue Point Means the Index of the Updated Layer. For Split CIFAR-100 20 Tasks Setup, We Show the First Five Tasks for Simplification.. 96

Chapter 1

INTRODUCTION

1.1 Overview

In the last decades, Artificial intelligence (AI), especially deep learning, has shown great success in various applications, such as Image classification, object detection/tracking, machine translation, and the recent popular ChatGPT, etc. Moreover, the development of edge devices pushes AI to the edge. Nowadays, the number of edge devices is growing rapidly which has the market size in billions. They receive and then process massive amounts of new data in our daily life. Thus the need for intelligent, personalized experiences powered by AI is ever-growing, for example, the AI applications on smartphones, smart cities, smart homes, and autonomous vehicles. In the current market, there are probably less than 100 million cold servers in the world. But there are already 3 billion mobile phones, 12 billion IoT devices and 150 billion micro-controllers. In the long term, these small, low-power devices will consume the most deep-learning applications.

Generally, the overflow of enabling deep learning at edge mainly can be divided into three steps as shown in Figure 1:

1. Offline training: given a dataset, the initial deep learning model (e.g., ViT, DNN) will be trained on cloud-based systems, for example by using powerful GPU servers.
2. On-device inference: after this offline training, the well-trained model will be

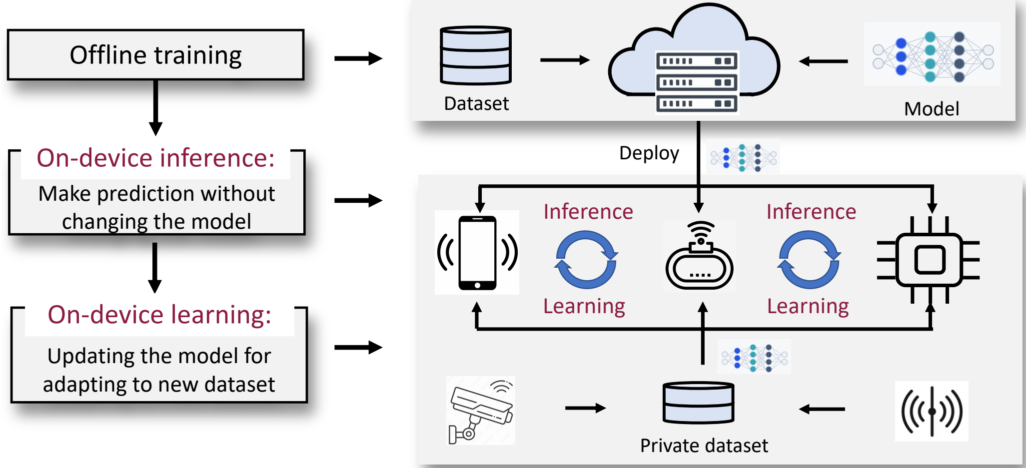


Figure 1. General Flow of Enabling Deep Learning at The Edge.

deployed on various edge devices to perform inference. Inference means that the well-trained model will be used to make predictions on edge devices without any modification to the model.

3. On-device learning: the users usually will collect their own data. They will further learn the model in order to transfer the pre-trained model to their own dataset.

Such edge AI/deep learning is becoming increasingly essential. Firstly, it can enable real-time decision-making, which is critical in applications such as autonomous vehicles. Compared to could computing, it has low latency and can avoid bandwidth competition. Second, as the data generated by different devices continues to grow, it is becoming impractical and inefficient to transmit all of the data to centralized cloud-based systems. Such edge deep learning can achieve data decentralization which allows for the processing of data closer to the source, reducing communication and improving response time. Lastly, edge deep learning can also enhance the privacy of data and deep learning models by collecting and processing the locally. This is particularly

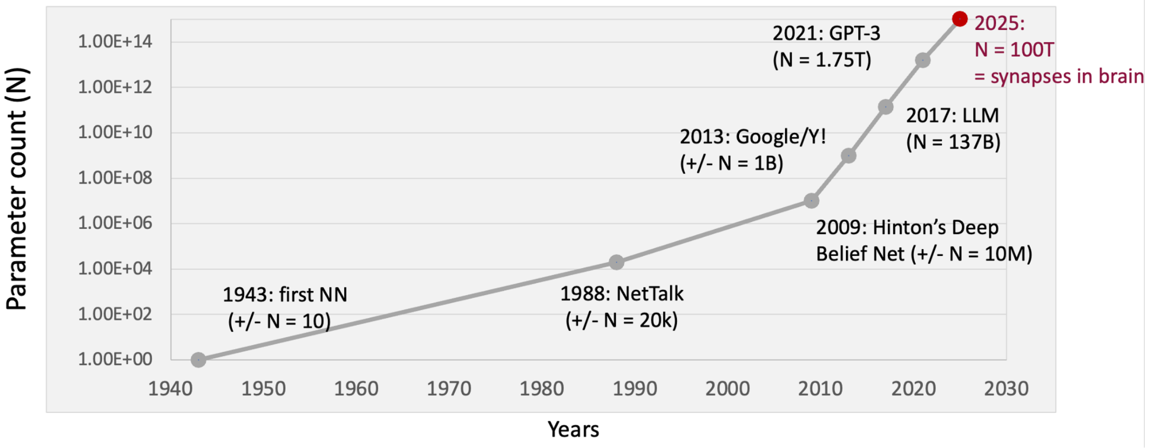


Figure 2. The Development Trend of The Deep Learning Models.

important in the applications such as healthcare and finance, where data privacy is of the utmost importance. However, there remains a huge gap between powerful deep learning models and edge devices.

1.2 Statement of the Main Problems

Deep learning models are quite expensive. Figure 2 shows the trends of the deep learning models. If looking over the years, the amount of parameters in deep learning models has increased exponentially. If this trend keeps up, we will have around 100 trillion parameters that reach the number of synapses in human brains. But the energy consumption of the human brain is 100,000 $\times$ more efficient[1]. Thus we could see that AI is powered by the explosive growth of deep learning models. A more powerful model with higher accuracy implies a larger model size, higher computing, and memory workload.

[1]Source from Qualcomm Technologies Inc.

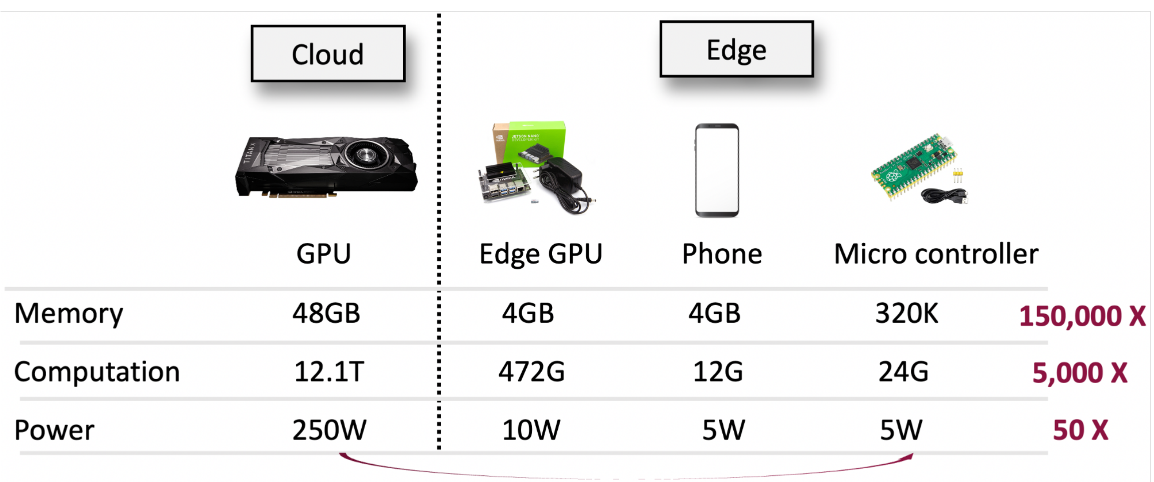


Figure 3. Hardware Resources of Various Devices.

In contrast, edge devices are diverse and resource-limited. Specifically, as shown in Figure 3, it's clear that different devices have diverse hardware resources. In addition, compared to cloud GPU, edge devices are memory, computation, and energy limited. For example, comparing microcontrollers to could GPUs, the memory size is 150,000$\times$ smaller, the computation capacity is 5000$\times$ and the power consumption is 50$\times$ smaller. In addition, Figure 4 shows the running time of the deep learning application (i.e., MobileNet V1 model [49] for ImageNet classification dataset) on various Android phones. It's clear to see that different phones have diverse running times for the same deep learning models[2]. Moreover, even for one single phone, there are many other factors that will affect running time, such as low battery, overheating, and heavy storage usage.

In summary, such a huge gap leads to several challenges in enabling deep learning at the edge as shown below:

[2]Source from AI-Benchmark

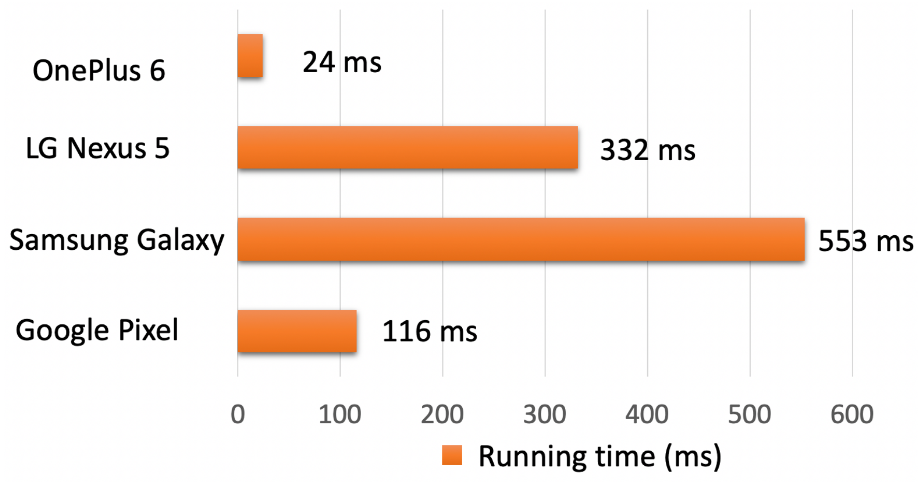


Figure 4. Running Times of Same Model on Various Android Phones.

- Edge devices suffer from large computing and memory costs, which make it difficult to be deployed on edge devices directly.
- Different devices have different hardware resources. Furthermore, even for one single device, it usually will be used on dynamic environments, which have different hardware resource constraints.
- For the on-device learning, updating the pre-trained model on their own dataset will make it forget prior learned knowledge on the cloud.

1.3 Contributions

To tackle the issues, the main objective of the Ph.D. study is to **enable deep learning at edge: from efficient and dynamic inference to on-device learning.**. In practice, the contributions of this dissertation can be summarized into four major parts: i) Hardware-friendly model compression to compress the model size for efficient inference [110, 85]; ii) Considering the dynamic hardware resource and environment, the run-time dynamic inference is further explored [117, 121, 115]; iii) compute and

memory-efficient algorithms to enable on-device learning [113, 120, 112]; iv) efficient continual learning methods to continually learn multiple tasks without forgetting prior learnd knowledge [116, 67, 65].

1.3.1 Contribution 1: Hardware-friendly Deep Neural Network Compression

In recent years, Deep Neural Networks (DNNs) have evolved into more complex model structures, characterized by deeper layers, and larger model sizes. Despite these advancements, the deployment of DNNs on lightweight hardware remains a challenge due to the substantial costs associated with computation and storage. A notable example is VGG-16 [103] from ImageNet (ILSVRC 2014), which necessitates 552MB parameters and 30.8 GFLOP per image, making it impractical to implement on mobile systems with limited resources. Many recent works have put forth various techniques to compress large DNNs, which mainly encompass network quantization [50], low-rank approximation [21], weight pruning [38] and knowledge distillation [46].

Weight pruning is a technique that reduces the size of a model by setting certain weights to zero. Previous research on weight pruning can be broadly categorized into two groups: non-structured pruning [38, 37] and structured pruning [58, 69, 108]. The primary difference between these two approaches lies in the regularity of the sparse weight patterns achieved after pruning. Non-structured pruning aims to generate highly irregular sparse weight patterns, striving for the highest possible sparsity. However, this irregularity poses challenges in efficiently encoding sparse weights due to sparse indexing. While non-structured pruning generally exhibits minimal degradation in compression accuracy due to its higher pruning flexibility, its effectiveness in hardware deployment is limited. In contrast, structured pruning

introduces weight sparsification in a regular manner, such as on a kernel-wise or channel-wise basis [108], making it more compatible with hardware deployment.

Moreover, weight quantization involves discretizing the weights of a DNN into multiple discrete levels, typically represented by limited-bit-width binary strings. Unlike weight pruning, which requires specialized hardware for efficient DNN inference, weight quantization is comparatively straightforward and easier to implement. In this study, we adopt a ternary value quantization scheme $(-1, 0, +1)$ to simplify the complexity of convolution computations from multiply-accumulate (MAC) operations to only additions and subtractions, thereby reducing the model size. Previous research [37] has explored the combination of pruning and quantization, demonstrating minimal accuracy degradation when using moderate compression rates (e.g., 8-bit quantization bit-width) on highly redundant DNN architectures like AlexNet. However, these previous works do not provide detailed insights for more aggressive compression methods applied to state-of-the-art DNN architectures.

To tackle these challenges, our goal is to combine structured weight pruning and ternarization techniques to enhance DNN inference performance on hardware platforms while minimizing accuracy loss. To address various technical concerns and ensure the effectiveness of our approach, we adopt a hardware/software co-design methodology. This allows us to thoroughly address and optimize various aspects of the system, taking into account both hardware and software considerations.

1.3.2 Contribution 2: Dynamic Inference with Run-Time Tuning of Model Size, Computation and Latency

As mentioned in Section 1.3.1, the model compression methods [49, 100, 37, 50, 21, 108, 46, 110] can effectively reduce the model size to achieve efficient inference. However, different hardware platforms have varying available resources, necessitating different levels of compression while maintaining similar latency requirements. Furthermore, even within a specific hardware platform, real-world scenarios introduce dynamic variations. For instance, smartphones may experience overheating or low battery, resulting in varying allocated computing resources for DNN computations and consequently affecting throughput and latency. In such cases, retraining and reloading DNN models to accommodate different or dynamic requirements can be prohibitively expensive or even impractical. This situation presents a new challenge: *How can we develop an adaptive DNN model that can dynamically adjust its computing complexity, model size, and accuracy on-the-fly to meet the changing application requirements and workload without the need for reloading new models?*

To overcome this challenge, our approach involves constructing a dynamic DNN structure composed of multiple subnets using a novel sub-network sampling method based on non-uniform channel selection. This method is inspired by the observation that different layers in parametric DNNs (such as convolution or fully-connected layers) have varying sensitivity to capacity reduction, as demonstrated in model pruning [64, 88, 108] and NAS (Neural Architecture Search) works [136, 70, 71]. This new dynamic DNN can adjust the number of involved convolution channels (i.e., model size and computing load) at runtime during the inference stage without the need for retraining. This allows for dynamic trade-offs between computing complexity (power

and speed) and accuracy. Unlike previous approaches with uniform structures, our dynamic DNN, referred to as a supernet model, comprises multiple subnets, each possessing a non-uniform structure to achieve optimal efficiency.

1.3.3 Contribution 3: On-Device Learning for Task Adaption

The proliferation of IoT devices has led to a significant increase in their utilization, with around 250 billion microcontrollers being deployed worldwide today[3]. These devices collect vast amounts of data across various domains and tasks, sparking researchers' interest in on-device AI capabilities. In addition to performing inference, there is a growing focus on enabling on-device training or transferring pre-trained models to new data. This learning paradigm aligns with the concept of transfer learning [13], where well-trained deep learning models are transferred from a primary source task to a new task, giving rise to the emerging field of on-device transfer learning. Compared to the conventional approach of training deep learning models in the cloud and performing inference on devices, on-device transfer learning offers several advantages. It eliminates the need for communication between cloud and edge devices, addressing concerns related to data privacy. Moreover, on-device learning enables real-time adaptation to changing conditions and data, enhancing the autonomy and responsiveness of IoT/edge devices. However, the memory requirements of the training process pose a significant challenge for memory-constrained IoT/edge devices. Recent studies on memory-efficient learning [15, 11, 111] have revealed that the storage of intermediate activations, rather than parameters, is the main bottleneck in training memory consumption. Therefore, reducing the memory footprint during training,

[3]https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/

particularly the storage of activation data from pre-trained models, becomes crucial for enabling on-device transfer learning. Unfortunately, existing transfer learning methods either suffer from high training memory consumption or have limited transfer capacity, hindering their practicality and effectiveness in on-device scenarios.

To address the limitations and challenges of previous approaches, we introduce a novel framework called "Reprogramming Network" (Rep-Net) that offers a fresh perspective on the on-device transfer learning problem. In Rep-Net, we focus on the concept of "intermediate feature reprogramming" to facilitate on-device transfer learning. The principle working mechanism of this design is to reprogram the fixed backbone model from the input data via the proposed activation connector that enables feature exchange between the backbone and Rep-Net at regular intervals. Such feature exchange uses an additive operation that not only helps both the backbone model and Rep-Net model to update and improve their features.

1.3.4 Contribution 4: Efficient Continual Learning via Progressive Layer Freezing

As described in Contribution 3, on-device learning aims directly train deep learning models on edge devices directly. A further step is to continually learn multiple tasks instead of a single one. Building upon the achievements of Self-supervised learning (SSL) in extracting visual representations from unlabeled data, researchers have explored its application in the realm of continual learning (CL). In CL, multiple tasks are learned sequentially, leading to the emergence of a new paradigm known as self-supervised continual learning (SSCL). SSCL has demonstrated superior performance compared to supervised continual learning (SCL) because the learned representations are more informative and resistant to catastrophic forgetting.

However, it is important to design SSCL approaches intelligently, as the training complexity can become excessively high due to the inherent cost of SSL. SSL typically requires additional computational resources and time compared to traditional supervised learning, as it involves the generation of pseudo-labels or pretext tasks to train models on unlabeled data. Thus, while SSCL holds great promise, it is crucial to develop efficient and scalable methods that strike a balance between the benefits of self-supervised learning and the practical constraints of training complexity. By addressing these challenges, SSCL can become a powerful framework for continual learning, enabling models to learn from and adapt to new tasks while preserving previously acquired knowledge.

To address the challenge of high training costs and catastrophic forgetting in self-supervised continual learning (SSCL), we propose a novel method called progressive task-correlated layer freezing (PTLF). Our approach leverages an analysis of task correlations based on gradient projection in SSCL, revealing that the intermediate representations learned through self-supervised learning exhibit high correlation and variability among tasks, in contrast to supervised continual learning (SCL). Motivated by this finding, we introduce PTLF as a means to reduce training time and memory costs in SSCL. The key idea is to selectively freeze layers during training based on the task correlations. To accomplish this, we first define a metric called the task correlation ratio, which quantifies the correlation between the current task and prior tasks using the gradient projection norm. Next, we progressively freeze the top-ranked layers with higher task correlation ratios among tasks during the self-supervised continual learning process for each task. By applying PTLF, we exploit the inherent correlations among tasks to prioritize the freezing of layers that have higher redundancy and lower task-specific information. This selective freezing strategy reduces computational and

memory requirements by effectively leveraging the shared representations learned through self-supervised learning.

1.4 Dissertation Structure

The dissertation structure is organized as follows:

- **Chapter 2** presents a novel hardware-friendly model compression method for efficient inference. It contains materials from "Harmonious coexistence of structured weight pruning and ternarization for deep neural networks" published at AAAI 2020 [110]. The dissertation author is the first author of the paper.
- **Chapter 3** presents a family of run-time dynamic inference methods. It contains materials from "Get More at Once: Alternating Sparse Training with Gradient Correction" published at NeurPS 2022 [117], "Non-uniform dnn structured subnets sampling for dynamic inference" published at DAC 2020 [121], and "A progressive subnetwork searching framework for dynamic inference" published at TNNLS 2022 [115]. The dissertation author is the first author of the papers.
- **Chapter 3** presents a memory-efficient algorithm to enable on-device learning for task adaption. It contains materials from "DA3: Dynamic Additive Attention Adaption for Memory-Efficient On-Device Multi-Domain Learning" published at CVPR-ECV 2022 [112] and "Rep-net: Efficient on-device learning via feature reprogramming" published at CVPR 2022 [113]. The dissertation author is the first author of the papers.
- **Chapter 4** presents an efficient continual learning algorithm via progressive layer freezing. It contains materials from "Efficient Self-supervised Continual

Learning with Progressive Task-correlated Layer Freezing". The dissertation author is the first author of the papers.

Chapter 2

HARDWARE-FRIENDLY MODEL COMPRESSION AND ACCELERATION

Deep convolutional neural networks (DNNs) have achieved remarkable success and are extensively employed in various computer vision tasks. However, their large model size and high computational complexity hinder their deployment in resource-limited embedded systems like FPGAs and mobile GPUs. Weight pruning and quantization are the two most widely used techniques for compressing DNN models. Weight pruning introduces weight sparsity by forcing certain weights to zero, while quantization reduces the bit-width of weights by representing them with limited precision values. Although there have been attempts to combine weight pruning and quantization, a lack of harmony between the two techniques persists, particularly when more aggressive compression schemes, such as structured pruning and low bit-width quantization, are employed. We propose a novel approach named PE-wise structured pruning, which incorporates the architecture of the Processing Elements (PE) to introduce weight sparsity. Additionally, we integrate this approach with an optimized weight ternarization technique, which quantizes weights into ternary values $(-1, 0, +1)$. By doing so, we convert the dominant convolution operations in the DNN from multiplication-and-accumulation (MAC) to addition-only, significantly reducing computational complexity. Moreover, this compression scheme compresses the original model from 32-bit floating point to a 2-bit ternary representation, achieving a compression ratio of at least 16 times. Furthermore, we address the coexistence issue between PE-wise structured pruning and ternarization by proposing a technique Weight Penalty Clipping (WPC) with a self-adapting threshold. This technique

effectively manages the trade-off between pruning and quantization, ensuring the compatibility and optimal performance of both approaches

2.1 Basics of Model Compression

2.1.1 Weight Quantization

Weight quantization had become a must-done step for deep learning models to be deployed on the hardware. The primary concept of weight quantization is to decrease the bit-width of the weight representation format, such as reducing from 32-bit floating-point to 8-bit integer [37]. Extensive research has been dedicated to this area, with the aim of compressing DNNs to lower bit-width (resulting in higher compression rates) while minimizing accuracy degradation compared to the full-precision baseline. Several research endeavors [62, 45] have focused on achieving this objective by optimizing weight quantization techniques.

We adopt the weight ternarization method proposed in [45] as our low bit-width quantization baseline. This method can be summarized as follows. For the weights $\boldsymbol{W}_l$ in the l-th layer of the DNN, the weight ternarization function can be formulated as:

$$\hat{w}_{l,i} = \text{Tern}(w_{l,i}, \Delta_l) = \alpha_l \cdot \begin{cases} +1 & w_{l,i} > \Delta_l \\ 0 & -\Delta_l \leq w_{l,i} \leq \Delta_l \\ -1 & w_{l,i} < -\Delta_l \end{cases} \tag{2.1}$$

$$\alpha_l = \mathbb{E}(|\{w_{l,i}\}|), \quad \forall \{i \big| |\boldsymbol{W}_{l,i}| > \Delta_l\} \tag{2.2}$$

where $\hat{w}_{l,i}$ is the i-th ternarized weight element in l-th layer. α_l is the layer-wise scaling factor (i.e., quantized value). The threshold $\Delta_l = 0.05 \cdot \max(|\boldsymbol{W}_l|)$. Similar to other

quantization techniques, we employ the Straight-Through Estimator (STE) method [4] to address the non-differentiability issue associated with the staircase quantization function defined in Equation (2.1). Therefore, considering a vectorized input $\boldsymbol{x}$ and target $\boldsymbol{t}$, the optimization process for the ternarized DNN can be expressed as:

$$\begin{aligned} \min_{\{\boldsymbol{W}_l\}_{l=1}^{L}} \quad & \mathcal{L}(f(\boldsymbol{x}; \{\hat{\boldsymbol{W}}_l\}_{l=1}^{L}), \boldsymbol{t}) \\ s.t.\ \{\hat{\boldsymbol{W}}\}_{l=1}^{L} &= \mathrm{Tern}(\{\boldsymbol{W}\}_{l=1}^{L}) \end{aligned} \tag{2.3}$$

where this equation uses identical notations.

2.1.2 Pruning

Pruning is another well-known technique for compressing neural networks, which removes partial weights or connections for reducing the model size and simplifying computations [38]. Depending on the sparsity pattern of the pruned weights, pruning techniques can be broadly categorized into two branches: non-structured and structured pruning [64, 77, 58, 108, 3, 44]. As shown in Figure 5, structured pruning methods lead to sparsity patterns (indicated by grey colors) with highly regular shapes. The classification of pruning methods into non-structured and structured categories is primarily driven by hardware considerations. Hardware accelerators for DNN inference, such as GPUs or FPGAs, can benefit more from the regular sparsity patterns generated by structured pruning methods (as shown on the right side of Figure 5), rather than the random sparse patterns produced by non-structured pruning methods (as shown on the left side of Figure 5). Therefore, structured pruning provides a more viable solution for hardware inference acceleration, as it enables efficient utilization of the regular sparsity patterns, resulting in improved computational efficiency and better hardware deployment.

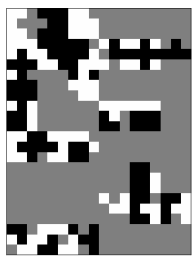


Figure 5. Stitched Ternarized Kernel (Each Kernel Is in 3×3) of the First Layer of Resnet-20 on Cifar-10, with (Left) Non-structured and (Right) Structured Pruning. Patterns in {White,Grey,Black} Denotes {-1,0,+1} Respectively.

Non-structured Pruning The study of non-structured pruning originated from the concepts of optimal brain damage and optimal brain surgeon [59, 39]. Han et al. further explored this technique in the context of deep neural networks (DNNs) [38, 37]. In their work [38, 37], Han et al. adopted a simple pruning strategy where weights below a predefined threshold are iteratively removed (i.e., set to zero), followed by fine-tuning to recover accuracy. Srinivas et al. proposed a data-free pruning algorithm in [102], which iteratively removes redundant neurons by connecting similar neurons together. The technique of variational dropout is utilized in [88] for pruning redundant weights in DNNs. In [79], Louizos et al. introduced a method where DNNs learn their sparse weights through L_0-norm regularization based on a stochastic gate. These studies represent different approaches and techniques within the realm of non-structured pruning, each with its own focus and methodology for reducing the redundancy in DNNs.

Structured Pruning In structured pruning, various sparsity patterns have been explored, including channel, kernel, and customized group patterns, in different studies [64, 77, 58, 108, 3, 44]. In [64], the authors directly prune unimportant filters based on their L_1-norm. They identify filters with small L_1-norms as less significant and remove them from the model. Liu et al. [77] introduce L_1 regularization on the

scaling coefficients of batch normalization layers as a penalty term. This regularization penalizes channels with small scaling coefficients, leading to their removal. On the other hand, the structured pruning methods proposed in [58, 108, 3, 44]share a common core technique: group Lasso. Group Lasso extends the concept of Lasso regularization to groups of weights or channels, promoting sparsity at the group level. These different approaches to structured pruning offer flexibility in choosing the sparsity pattern based on the specific requirements of the task or the characteristics of the neural network model.

Group Lasso was initially introduced in [128], and Wen et al. [108] incorporated it as an additional term in the loss function during the training of DNNs using back-propagation. The loss function can be formulated as:

$$\hat{\mathcal{L}} = \mathcal{L}\left(f(\boldsymbol{x}; \boldsymbol{W}ll = 1^L), \boldsymbol{t}\right) + \lambda \sum_{l=1}^{L} \sum_{i=1}^{G_l} \mathcal{P}(\boldsymbol{W}_{l,i}) \tag{2.4}$$

In this equation, $f(\boldsymbol{x}; \boldsymbol{W}ll = 1^L)$ computes the outputs of the DNN with parameters $\boldsymbol{W}ll = 1^L$ given the input $\boldsymbol{x}$. $\mathcal{L}(\cdot, \cdot)$ represents the objective function of the DNN, such as the cross-entropy loss. $\mathcal{P}(\boldsymbol{W}l, i) = ||\boldsymbol{W}l, i||2$ calculates the Euclidean norm of the indexed weight group $\boldsymbol{W}l, i$. The second term in Equation (2.4) is the L_1-norm of $\mathcal{P}(\boldsymbol{W}_{l,i})$, known as group Lasso [128]. It acts as a group-wise weight penalty, promoting group-wise sparsity during optimization. G_l represents the number of groups in the l-th layer, and λ is a hyperparameter that needs to be tuned based on the dataset. By incorporating group Lasso into the loss function, Wen et al. aim to encourage group-wise sparsity, leading to structured sparse weight patterns in the DNNs. This technique facilitates efficient hardware deployment and computation due to the structured nature of the sparsity.

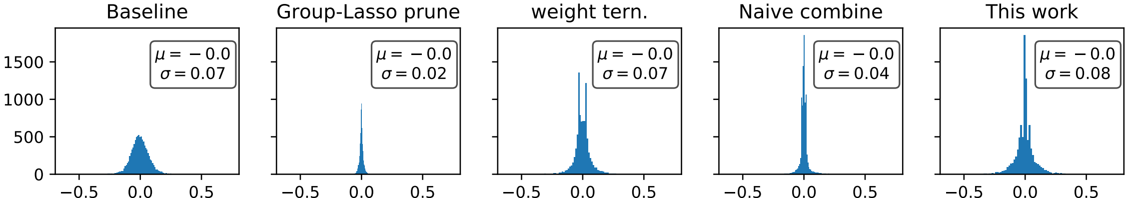


Figure 6. Sample Weight Distribution of Resnet-20 Layer under Different Training Configurations, with Mean (μ) and Standard Deviation (σ). Baseline: Full-precision Model Without Compression; Group-lasso Prune: Structured Weight Pruning Utilizing Group Lasso Technique; Weight Tern.: Weight Ternarization; Naive Combine: Naively Combining the Aforementioned Pruning and Ternarization; This Work: Combining Pruning and Ternarization with Proposed Weight Penalty Clipping Technique.

2.2 Proposed Model Compression Method

2.2.1 Problem Formalization

As discussed above, both group-lasso based structured pruning and weight ternarization have demonstrated impressive performance in compressing DNN models while preserving inference accuracy compared to the full-precision baseline. To maximize the model compression rate and leverage the advantages of both techniques, we initially combine them in a straightforward manner. The DNN training process can be formulated as minimizing the following loss function:

$$\hat{\mathcal{L}} = \mathcal{L}(f(\boldsymbol{x}; \mathrm{Tern}\{\boldsymbol{W}_l\}_{l=1}^{L}), \boldsymbol{t}) + \lambda \sum_{l=1}^{L} \sum_{i=1}^{G_l} \mathcal{P}(\boldsymbol{W}_{l,i}) \tag{2.5}$$

Nevertheless, such naive combination leads to severe accuracy degradation without significantly improving the compression rate.

To investigate the problem further, we analyze the weight distribution of different training configurations. The histograms of these weight distributions are depicted in Figure 6. The results reveal that the group Lasso-based structured pruning acts as

a regularization term, causing the weights to shift towards smaller values, indicated by a small standard deviation (σ) and a mean (μ) close to zero, compared to the full-precision baseline. On the other hand, weight ternarization has a negligible effect on the standard deviation compared to the baseline. Interestingly, when we naively combine the structured pruning and ternarization methods, we observe that the standard deviation converges to a value between that of the group Lasso and weight ternarization counterparts. This observation inspires us to investigate the potential cause of accuracy degradation when these two methods are combined.

- Is that applying the group Lasso (i.e., weight penalty) upon entire weights contradictory to the weight ternarization method?
- Whether maintaining the weight distribution close to the original ternary counterpart helpful to mitigate the accuracy degradation?

Building upon these insights, we introduce a novel technique called *Weight Penalty Clipping* (WPC) with a self-adapting threshold to address the coexistence issue between group Lasso-based structured pruning and ternarization. The WPC technique is outlined in the following subsection.

2.2.2 Weight Penalty Clipping with Self-adapting Threshold

To address the issue discussed above, we make further adjustments to the intra-group L_2-norm term in Eq.2.4. As a result, the Eq.2.5 can be reformulated as:

$$\hat{\mathcal{L}} = \mathcal{L}(f(\boldsymbol{x}; \text{Tern}\{\boldsymbol{W}_l\}_{l=1}^{L}), \boldsymbol{t}) + \lambda \sum_{l=1}^{L} \sum_{i=1}^{G_l} \underbrace{\min\big(||\boldsymbol{W}_{l,i}||_2; \delta_l\big)}_{\text{Weight Penalty Clipping}} \tag{2.6}$$

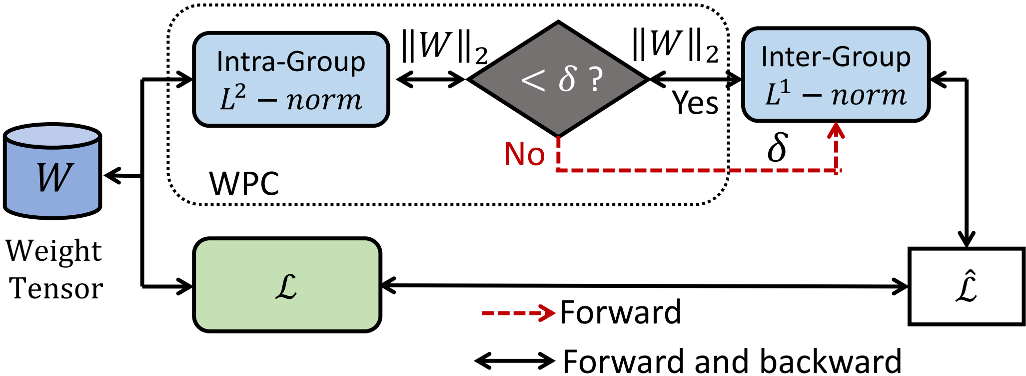


Figure 7. The Overview of Weight Penalty Clipping with Self-adapting Threshold.

$$\delta_l = a \cdot \frac{1}{G_l} \sum_{i=1}^{G_l} ||\boldsymbol{W}_{l,i}||_2 \tag{2.7}$$

where δ_l represents the layer-wise self-adapting clipping threshold. It is employed to limit the penalty of the intra-group L_2-norm term on large weights. The parameter a is a scaling coefficient. It is important to note that when the intra-group L_2-norm penalty of $\boldsymbol{W}l, i$ is clipped, the inter-group L_1-norm penalty is also clipped accordingly. We refer to this technique as Weight Penalty Clipping (WPC). Figure 7 provides an overview of the WPC process. In each training iteration, the updated weights are utilized in the loss function, as shown in Eq.2.7. Subsequently, after calculating the intra-group L_2-norm, WPC compares it with the threshold δ_l to determine whether the corresponding $||\boldsymbol{W}l, i||_2$ will be used in the loss function and contribute to the backward propagation. Considering two cases:

- When $||\boldsymbol{W}l, i||2 \geq \delta_l$, it indicates that the weights in $\boldsymbol{W}l, i$ are relatively large and important, and they should not be pruned by the group Lasso term in Eq.2.6. In this case, weight penalty clipping is performed, where the weight penalty term of $||\boldsymbol{W}l, i||_2$ in $\hat{\mathcal{L}}$ is replaced with δ_l. It is important to note that δ_l is treated as a constant and its calculation is excluded from the backward computation graph.

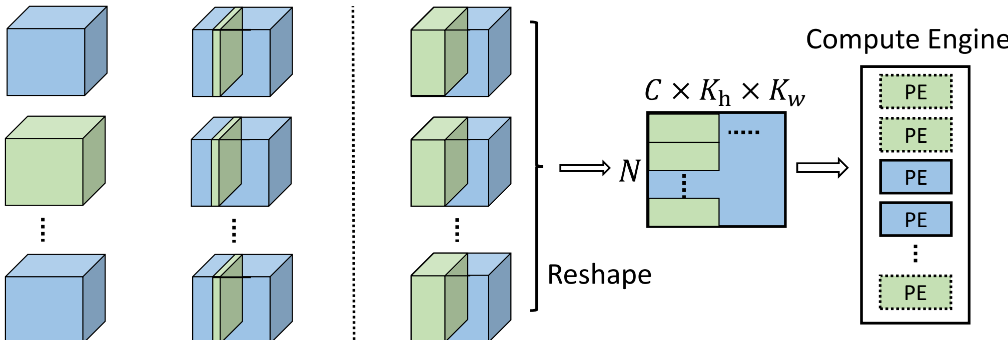


Figure 8. PE-wise Pruning.

- When $||\boldsymbol{W}_{l,i}||2 < \delta_l$, we retain the weight penalty term of $||\boldsymbol{W}l,i||2$ in its original value. This allows the group Lasso term to continue affecting $\boldsymbol{W}l,i$ and prune the weights in a group-wise fashion.

2.2.3 PE-wise Structured Pruning

The performance, such as accuracy, of a DNN pruned using a group Lasso-based method heavily relies on the defined group shape. Various group shapes, such as channel, filter, and depth, have been explored in [108]. However, using a large group capacity (i.e., a high number of weights per group) often leads to either low group sparsity or low inference accuracy. On the other hand, using a small group capacity can be beneficial for both group sparsity and accuracy. However, it may be challenging to accelerate the pruned DNN on target hardware due to the irregular sparsity pattern.

To strike a balance between DNN performance after structured pruning and inference efficiency on target hardware, we propose making the group shape identical to the Processing Element (PE) of the target hardware. The PE represents the basic computing unit in modern DNN accelerators, such as FPGAs, ASICs, or GPUs. By

aligning the group shape with the PE architecture, the DNN accelerator can efficiently utilize the PE-wise sparsity pattern, resulting in improved speed-up, as demonstrated in our later experiments.

Due to limited hardware resources, achieving fully parallelized computation is challenging. To address this issue, we propose a PE-wise structured pruning approach where the group size is defined to be equal to the capacity of one PE. In a standard convolution layer, the weights are stored in a 4D tensor $\boldsymbol{W} \in \mathbb{R}^{N \times C \times K_{\mathrm{h}} \times K_{\mathrm{w}}}$, where N, C, K_{h}, and K_{w} denote the output channel, input channel, kernel height, and kernel width in the current layer, respectively. As illustrated in Figure 8, the size of the PE-wise pruning falls between filter ($C \times K_{\mathrm{h}} \times K_{\mathrm{w}}$) pruning and channel ($N$) pruning. Furthermore, in terms of hardware deployment, the high-dimensional convolution operation is often reduced to matrix multiplication. This involves reshaping the 4D weight tensor into a 2D matrix with a size of $(N, C \times K_{\mathrm{h}} \times K_{\mathrm{w}})$. As a result, defining structured sparsity groups in terms of channels (C) and shape ($K_{\mathrm{h}} \times K_{\mathrm{w}}$) does not align well with practical hardware implementation. To address this mismatch, we choose the capacity of a PE as the group size for performing group-wise pruning, with a size of $C_{\mathrm{g}} \times K_{\mathrm{h}} \times K_{\mathrm{w}}$. Here, C_{g} is a multiple of C. With this approach, one PE can perform computations between one group of weights and the corresponding partial feature map vector with a size of $C_{\mathrm{g}} \times H \times W$. This aligns with the hardware architecture and allows for efficient computation within the PE-wise pruning scheme.

2.3 Experiments

2.3.1 Experiment Setup

2.3.1.1 Dataset and Network Structure

To evaluate the performance of our proposed technique, we conducted experiments on a classic image classification task. We used two datasets in this work: CIFAR-10 [57] and ImageNet [19]. CIFAR-10 consists of 50,000 training samples and 10,000 test samples, with each image having a size of 32×32 pixels. For CIFAR-10, we employed the ResNet-20/32/44/56 architecture [41]. The network was trained using the momentum SGD optimizer, with an initial learning rate of 0.1. The learning rate was scaled by a factor of 0.1 at epochs 80 and 120. Data augmentation techniques were applied following the configuration described in [41].

For the ImageNet dataset, it consists of 1.2 million training images and 50,000 validation images, with a total of 1,000 categories. We followed the data preprocessing scheme adopted by ResNet [41]. We employed the ResNet-18 and AlexNet architectures for our experiments on ImageNet. The networks were trained using the Adam optimizer, with an initial learning rate of 0.0001. The learning rate was scaled by a factor of 0.2 at epochs 30, 40, and 45. Similar to other quantization works such as [68, 45, 96], we kept the first and last layers in full precision, i.e., 32-bit float, while applying quantization to the intermediate layers.

2.3.1.2 Experiment Platform

The algorithm was implemented using the PyTorch deep learning framework[4] and executed on a system equipped with 4 NVIDIA Titan XP GPUs. To evaluate the performance of PE-wise structured pruning, we designed a FPGA-based DNN accelerator. The FPGA platform used was the Xilinx PYNQ-Z1 board, which is supported by the PYNQ open-source framework. The board features a Xilinx Zynq-7000 SoC, which includes an XC7Z020 FPGA and an embedded ARM Cortex-A9 processor. This FPGA-based accelerator provides hardware acceleration for DNN computations, enabling efficient execution of the pruned models.

2.3.1.3 Compression Rate

To fully leverage structured pruning from a hardware perspective, we can utilize a binary indexer to indicate which PE groups contain all zero values. With this approach, we only need to store the weight groups that have non-zero values in memory. The size of the binary indexer is negligible compared to the size of the weights. Consequently, we can formulate the compression rate C of the weights as follows:

$$C = \frac{32}{(1 - G_s) \cdot n} \tag{2.8}$$

where G_s represents the group sparsity, which is the fraction of the number of groups with all zero values over the total number of groups across layers. The term n denotes the bit-width of the model, which is set to 2 in this case as we encode the weights in binary format. The value 32 corresponds to the bit-width of full precision. It

[4]https://pytorch.org/

Table 1. Inference Accuracy (%) of Resnets on Cifar-10.

	ResNet-20	ResNet-32	ResNet-44	ResNet-56
FP	91.7	92.36	92.47	92.68
Ours	90.89	91.62	91.76	92.05
Gap	-0.81	-0.74	-0.71	-0.63

should be noted that when calculating the compression rate, we only consider the group sparsity since it is the matrix that can be realized in hardware implementation. Furthermore, all the compression rates of our method, as shown below, are defined using Eq.2.8.

2.3.2 Experimental Results

2.3.2.1 CIFAR-10 Experiment

The proposed method was evaluated on ResNet-20/32/44/56 architectures. The size of the PE group was set to $16 \times 3 \times 3$ for all layers in all network types. The results of the inference accuracy, as shown in Table.1, indicate that all four ResNet models achieved less than 1% accuracy loss compared to the floating-point baseline. It is worth noting that more compact neural networks, such as ResNet-20, are more susceptible to accuracy loss due to the aggressive compression of the model, which hampers the network's capacity.

Table 2. Simulation Result of Structured Pruning for Alexnet on Imagenet Dataset. FP. Indicates Full-precision (32-bit Floating-point) and Tern. Indicates Ternary Weights. Note That, Acc. And Comp. Are the Abbreviation of Accuracy and Compression.

Method	Weight format	Top-1 Acc. gap (%)	Sparsity metrics	Layer index						Comp. rate
				Conv1	Conv2	Conv3	Conv4	Conv5	Conv2-5	
SSL [108]	Floating-point (32-bit)	2.06	Column (%)	0.0	63.2	76.9	84.7	80.7	-	$\sim 6.4\times$
			Row (%)	9.4	12.9	40.6	46.9	0.36		
			Layer (%)	9.4	68.8	86.3	91.9	80.8	84.4	
This work	Ternary (2-bit)	3.23	Group (%)	24.3	36.5	47.3	45.0	24.2	35.46	$\sim 24.7\times$
			In-group (%)	13.2	59.5	48.6	41.7	36.1		
			Layer (%)	34.3	80.2	72.9	67.9	51.6	68.2	

2.3.2.2 ImageNet Experiment

The proposed method was evaluated on ResNet-18 and AlexNet architectures. The size of the PE group was set to $64 \times 3 \times 3$ for all convolutional layers in both networks.

For the AlexNet architecture, we conducted comparisons with both non-structured ternarization and related structured pruning methods, specifically the work by Wen *et al.* [108]. However, due to the differences in baseline accuracy (61.78% for our method and 57.4% for [108]), we reported the accuracy loss in Table 2 to provide a fair comparison. We evaluated various types of sparsity, including column sparsity (filter-wise), row sparsity (shape-wise), and layer sparsity. Additionally, we introduced PE-wise sparsity, which is based on the computation capacity of a single PE. The simulation results in Table 2 demonstrate a 1.17% accuracy degradation and a 16.2% sparsity reduction compared to [108] while achieving a much higher compression rate.

For the ResNet architecture, we compared our method with existing non-structured weight quantization works, specifically those using binarization and ternarization techniques. The inference accuracy results are shown in Table 4. When compared to ABC-Net, which utilizes multiple binarization approximation techniques, our method achieves almost the same accuracy with a much higher compression rate. Furthermore,

Table 3. Ablation Study on CIFAR-10

	Tern	Pruning	Naive combine	Ours
ResNet-20	91.62	91.1	90.01	90.89
Overall sparsity	49	61	70	50
Group sparsity	-	44	18	25
Comp. rate	$\sim 16\times$	~ 1.78	$\sim 19.5\times$	~ 21.5
ResNet-32	92.48	91.88	90.68	91.64
Overall sparsity	48	40	74	58
Group sparsity	-	34	28	43
Comp. rate	$\sim 16\times$	~ 1.5	$\sim 22.2\times$	~ 28.1
ResNet-44	92.71	92.29	91.15	91.98
Overall sparsity	55	58	80	64
Group sparsity	-	46	21	44
Comp. rate	$\sim 16\times$	~ 1.85	$\sim 20.2\times$	~ 28.6
ResNet-56	93.1	92.86	92.01	92.85
Overall sparsity	52	67	82	67
Group sparsity	-	28	42	55
Comp. rate	$\sim 16\times$	~ 1.39	$\sim 27.6\times$	~ 35.6

Table 4. Inference Accuracy (%) of ResNet18 on ImageNet

	Quan scheme	First layer	Last layer	Accuracy (top1/top/5)	Comp. rate
Baseline	-	FP	FP	69.75/89.07	$1\times$
BWN[96]	Bin.	FP	FP	60.8/83.0	$\sim 32\times$
ABC-Net[68]	Bin.	FP	FP	68.3/87.9	$\sim 6.4\times$
ADMM[62]	Bin.	FP	FP	64.8/86.2	$\sim 32\times$
TWN[62, 63]	Tern.	FP	FP	61.8/84.2	$\sim 16\times$
TTN[135]	Tern.	FP	FP	66.6/87.2	$\sim 16\times$
ADMM[62]	Tern.	FP	FP	67.0/87.5	$\sim 16\times$
[45]	Tern.	FP	FP	67.95/88.0	$\sim 16\times$
Ours	Tern.	FP	FP	**68.01/88.13**	$\sim 21.3\times$

when compared to [45], which employs the same non-structured ternarization method as our work, we achieve a 1.35x compression rate with a slight accuracy enhancement.

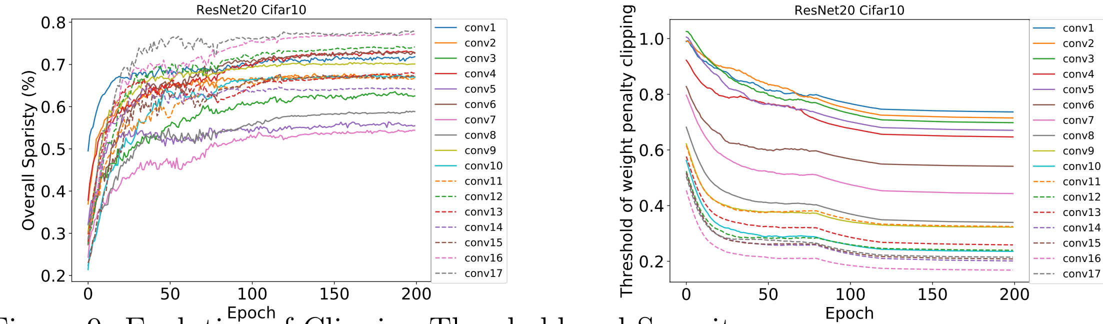


Figure 9. Evolution of Clipping Threshold and Sparsity

2.4 Ablation Study and Discussion

To evaluate the effectiveness of our proposed method, we compared it with three different cases: weight ternarization, Group-Lasso pruning, and directly combining these two techniques (naive combine) on both the CIFAR-10 and ImageNet datasets. The results are presented in Table 3 and Table 5. In these tables, overall sparsity refers to the ratio of individual zero values within the entire weight tensors, while group sparsity represents the ratio of PE-wise groups that are all zeros.

Comparing our method with the naive combine, we observed that our method achieves smaller overall sparsity but larger useful group sparsity. This result demonstrates the effectiveness of the proposed weight penalty clipping with self-adapting threshold. By utilizing this technique, only the weight groups with smaller norm values are regularized, while the rest of the weight groups remain unchanged. This approach focuses on pruning unimportant weight groups while preserving the important ones, leading to improved compression efficiency.

2.4.1 Evolution of Clipping Threshold

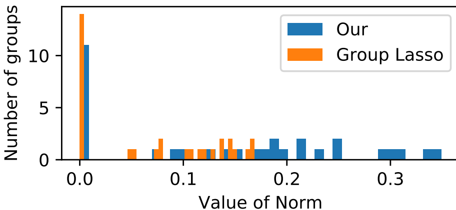
The clipping threshold and sparsity are two important dynamic factors during the training process. Figure 9 illustrates the dynamics of sparsity and clipping threshold during training. It can be observed that the sparsity increases significantly in the first 80 epochs and then reaches a stable state. Similarly, the clipping threshold follows a similar pattern, with its value becoming smaller over time. This behavior can be attributed to the large initial learning rate, which amplifies the effect of the Group Lasso regularizer on the weights. Additionally, we observe that the later layers tend to have larger sparsity compared to the front layers. This could be due to the fact that the later layers capture higher-level features and are thus more likely to contain redundant or less important weights.

Table 5. Ablation Study of ResNet-18 [41] on ImageNet.

	Acc	Overall sparsity	Group sparsity	Comp. rate
FP	69.75	-	-	1.0
Pruning	68.0	43	38	$\sim 1.6\times$
Tern	67.95	60	-	$\sim 16\times$
Naive combine	65.12	75	18	$\sim 19.5\times$
Ours	68.01	70	25	$\sim 21.3\times$

2.4.2 PE Capacity vs. Structured Sparsity Ratio

To evaluate the effectiveness of different PE-group sizes, we conducted experiments using ResNet-20 with 5 different PE sizes (e.g., PE-8 denotes a PE-group size of $8 \times K_{\mathrm{h}} \times K_{\mathrm{w}}$). All models were trained using the same hyper-parameter settings. The results are shown in Table 6. It can be observed that smaller PE-group sizes are more easily regularized, resulting in larger sparsity and greater accuracy loss compared to



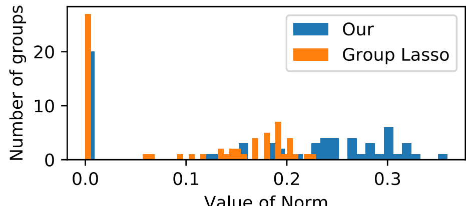


Figure 10. Norm-based Criterion on ResNet20 for CIFAR-10 dataset. (Left) Is the Conv8 Layer and (Right) Is Conv15 Layer.

models with larger PE-group sizes under the same hyper-parameter settings. This indicates that the choice of PE-group size can have an impact on the trade-off between sparsity and accuracy in the compressed model.

Table 6. PE Capacity Versus Structured Sparsity Ratio of ResNet-20 Trained on CIFAR-10 Dataset.

	PE-16	PE-8	PE-4	PE-2	PE-1
Accuracy (%)	90.89	90.56	90.29	89.93	89.66
Overall sparsity (%)	50	70.75	78.65	81.97	84.95
Group sparsity (%)	25	35.6	42.8	50.07	63.18

2.4.3 Relation to the Norm-based Criterion

In [43], two requirements are mentioned for pruning: 1) the norm deviation of the filters should be large, and 2) the minimum norm of the filters should be small. Figure 10 shows the norm distribution of our method compared to the naive combine approach. It can be observed that our method has a larger norm deviation, indicating that the norm distribution becomes more suitable for pruning during the training process. This suggests that it becomes easier to distinguish between important and unimportant weight groups based on the absolute value of the norm in our method.

Table 7. Convolutional Layer Implementation Setup and Speed Up on FPGA

Layer	Weight size	PE size	Non-sparsity groups	Speedup
Stage 1	$(16, 16, 3, 3)$	$16 \times 3 \times 3$	9	1.56×
Stage 2	$(32, 32, 3, 3)$	$16 \times 3 \times 3$	33	1.69×
Stage 3	$(64, 64, 3, 3)$	$16 \times 3 \times 3$	119	1.73×

2.4.4 FPGA Implementation

To evaluate the performance of our proposed method on real-world inference hardware accelerators, we deployed three representative convolutional layers of ResNet20 on an FPGA board. We compared our method with a weight-ternarization-only approach, which also replaces MAC operations with add/sub operations but does not include PE-wise pruning. To ensure a fair comparison, we used a fixed-point number representation with a bit-width of 16 bits for both methods. The detailed setup and speedup results are presented in Table.table:hardware. It is evident that our proposed PE-wise structured ternary network achieves significant speedup compared to the non-structured ternary network in real FPGA hardware implementation.

2.5 Summary

In this section, our main objective is to leverage the benefits of Group Lasso based pruning and ternarization to optimize the efficiency of DNN hardware deployment. We introduce the concept of PE-wise sparsity and observe the disharmony between these two methods. To address this issue, we propose a novel approach called weight penalty clipping with a self-adjustable threshold. By utilizing this method, we can achieve a more balanced and efficient combination of pruning and ternarization techniques.

Furthermore, when compared with existing works in weight pruning and ternarization, our proposed method demonstrates superior performance in terms of accuracy while achieving significantly higher compression rates. This highlights the effectiveness and competitiveness of our approach in the field of DNN model compression and hardware deployment.

Chapter 3

DYNAMIC DEEP NEURAL NETWORK WITH RUN-TIME TUNING OF ACCURACY AND LATENCY

Recently, there has been a growing interest in exploring training sparsity as a means to improve both training and inference efficiency. However, existing approaches focus on obtaining a single sparse model with a fixed sparsity ratio, which limits their flexibility in adapting to varying hardware resource availability. To address this limitation, the concept of dynamic inference or training-once-for-all has been proposed, where a single network is trained to include multiple sub-networks that can perform the same inference function with different computational complexity.

In this chapter, we introduce a novel approach called Alternating Sparse Training (AST) to enable dynamic inference without incurring extra training costs compared to training a single sparse model. Our approach involves training multiple sparse sub-networks alternately, allowing for efficient adaptation of computation complexity at runtime. To mitigate interference in weight updates among sub-networks and ensure optimal generalization, we propose gradient correction within the inner-group iterations. This correction mechanism helps to reduce interference while preserving the effectiveness of the optimization process. By employing the AST scheme, we enable efficient dynamic inference with the ability to adjust computational complexity based on available hardware resources. This approach provides a more flexible and practical solution compared to traditional dynamic inference methods that involve joint training and multi-objective optimization, which often suffer from significant training overhead.

3.1 Bascis of Sparse Training

As summarized in the survey paper by Hoefler et al. [47], the existing works on sparse training can be broadly classified into three categories based on when the sparsity is applied during the training process.

The first category is post-training sparsification, where sparsity is introduced by fine-tuning a pre-trained model. This approach involves removing weights or connections from the model after the initial training phase. Examples of post-training sparsification methods include Dettmers et al. [22], Evci et al. [25], Jayakumar et al. [51], Peste et al. [94], and Liu et al. [74]. The second category is before-training sparsification, where a sparse model is obtained before the main training procedure begins. This approach aims to initialize the model with sparsity, which can be achieved through various techniques such as magnitude-based pruning or weight picking. Lee et al. [60] and Wang et al. [106] are examples of before-training sparsification methods. The third category is in-training sparsification, where sparsity is introduced and optimized during the training process itself. This category includes methods that gradually remove weights or connections from the model as the training progresses. Examples of in-training sparsification methods are Liu et al. [74], Yuan et al. [127], Evci et al. [25], and Dettmers et al. [22].

3.1.1 In-training Sparsification

Unlike post-training and before-training sparsification approaches, incorporating sparsity during training eliminates the need for separate training processes. The pruning of the model occurs concurrently with weight optimization, allowing the

pruning topology to be refined and improved iteratively. By pruning the model during training, the algorithms have access to the gradients, which provides valuable information for making informed pruning decisions. This visibility into the gradients enables the algorithms to adjust and refine the pruning process based on the gradient information, resulting in improved accuracy compared to post-training sparsification methods.

Motivated by this, the *prune-and-regrow* technique [22] is introduced, which involves periodically removing unimportant non-zero weights from the sparse model and regrowing certain pruned weights during each mini-batch iteration of the training process. This iterative process allows for the refinement and optimization of the sparse model's connectivity pattern over time.

As a representative work, RigL [25] introduces a pruning technique where a certain ratio r of weights is pruned based on their magnitude. This is done by selecting the top $s-r$ proportion of weights with the highest magnitudes using the TopK operation, as defined by:

$$w' = \mathrm{TopK}(|w|, s-r), \tag{3.1}$$

where $|w|$ represents the magnitude of weights and s is the targeted sparsity ratio. This initial pruning step reduces the number of non-zero weights in the model. After pruning a certain ratio r of weights using Equation 3.1, the next step in RigL is to regenerate the pruned connections. This is done by introducing new connections based on the gradient magnitude during the same mini-batch iteration. Specifically, the top $s+r$ proportion of gradients, excluding the previously pruned weights w', are selected using the $TopK$ operation, and added back to the pruned weights as follows:

$$w = w' + TopK(g_{i \neq w'}, s+r), \tag{3.2}$$

where g represents the gradients of the weights and $i \neq w'$ indicates excluding the previously pruned weights from the selection. This regeneration step helps to recover some of the pruned connections based on their importance as indicated by the gradients.

By employing the prune-and-regrow scheme, the sparse connection is optimized with a fixed sparsity ratio r throughout the entire training process. Recent works have explored different weight importance criteria to perform prune-and-regrow. For example, SNFS [22] uses the momentum magnitude of weights for pruning, while MEST [127] considers the sum of weight and gradient magnitude as an indication of weight importance. Additionally, GraNet [74] follows a similar rule as RigL but initializes the sparsity at a lower value (e.g., 0.5) and gradually increases it to the target sparsity (e.g., 0.8, 0.9) using a cosine decay schedule. Furthermore, there are works such as [94, 87] that propose to alternately train a dense model and one of its sparse variants (sub-net) to obtain both an accurate dense model and a sparse sub-net. However, the underlying rationale for this approach is not clearly described. The objective of this section is to obtain multiple sparse sub-networks with the same training effort as individual network sparsification. Additionally, based on the in-training sparsification mechanism, [73] proposes a dynamic sparse training ensemble method to independently generate multiple sparse sub-nets for ensemble. This approach is orthogonal to our work and could be combined to further improve accuracy.

3.2 What is Dynamic Inference

Dynamic inference methods aim to train a single network composed of multiple subnets that can independently perform inference. These subnets share some weights, allowing for runtime switching and enabling dynamic trade-offs between accuracy and computational complexity. Notable examples include Slimmable Neural Network (S-NN)[124], Universally Slimmable Networks (US-NN)[122], BigNAS [123], Once-for-All [9], and Progressive Neural Architecture Search [114]. The concept of dynamic neural networks was first introduced by Slimmable Neural Network (S-Net)[124], which enables a single neural network to be executed at different channel widths. Building upon this idea, Once-for-All (OFA)[9] and BigNAS[123] further extend the concept by constructing dynamic DNNs that encompass a larger number of subnets across multiple dimensions, including depth, width, kernel size, and resolution. Note that, as discussed in [9], to enable sub-net switching without compromising individual inference accuracy, the structures of sub-nets are defined using a subset rule, where the smaller sub-net is a complete subset of the larger sub-net. Specifically, these works train a network in a joint training fashion, which can be expressed as:

$$\min \ \mathbb{E}_W\Big(\textstyle\sum_{i=1}^{N} \mathcal{L}(f(X, \{W_i\}); Y), \Big) \tag{3.3}$$

where X is the mini-batch of inputs with corresponding targets Y, and N is the number of sub-nets. $\mathcal{L}(\cdot;\cdot)$ calculates the cross-entropy loss of the DNN output and the target. $f(X, W_i)$ computes the output of the sub-net parameterized by W_i. However, this method requires a longer training time compared to training a single individual model, as all sub-nets have to perform forward and backward passes in each mini-batch iteration, as shown in Eq. 3.3.

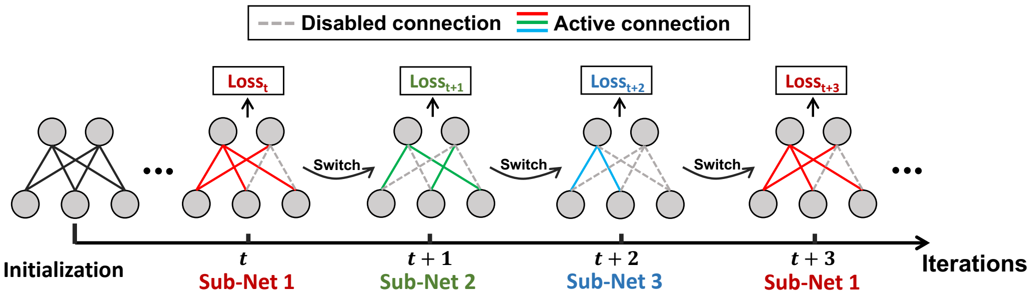


Figure 11. Alternating Sparse Training (AST): The Subset Network (Sub-net) are Iteratively Activated Throughout the Training. The Model Only Learns the Active Connections of Each Sub-net, Leading to the One-time-Trained Multiple Sub-nets.

3.3 Sparse Train More at Once

Different from joint training methods for dynamic inference, such as S-Net [124] and US-Net [122], where all sub-nets are updated in each mini-batch iteration, we propose the **alternating sparse training (AST)** scheme, which trains multiple sub-nets in an alternating fashion over time. Each iteration focuses on training a single sub-net.

In AST, each sub-net is defined as a subset of weights in the network that can perform inference independently. All the sub-nets are partially shared with each other within a single network. As illustrated in Figure 11, assuming we have three sub-nets with different sparsity ratios (e.g., sub-net 1 $<$ sub-net 2 $<$ sub-net 3), the training scheme starts by training sub-net 1 in the first iteration, followed by the training of sub-net 2 and sub-net 3 in the second and third iterations, respectively. During training, only the active connections (non-pruned weights) are updated, while the disabled weights (pruned for the current sparse sub-net) are ignored. This alternating process of updating sub-nets continues in a cyclical manner, repeatedly switching sub-nets every few consecutive iterations until the training process is completed.

3.3.1 Rationale of Alternating Sparse Training (AST)

The motivation behind AST is inspired by the insights from Reptile [91], a meta-learning algorithm originally designed to learn task-specific initializations for fast adaptation to new tasks. Reptile has shown that mini-batch stochastic gradient descent (SGD) implicitly regularizes the model by maximizing the dot product of consecutive mini-batches. We adopt this finding to our case. Considering the alternating training of two sub-nets, sub-net 1 and sub-net 2, with two consecutive mini-batches B_1 and B_2, respectively, we can express the gradient of sub-net 2, g_2, calculated by SGD using the Taylor Series expansion:

$$
\begin{aligned}
g_2 = \mathcal{L}^{'}(w_2) &= \mathcal{L}^{'}(w_1) + \mathcal{L}^{''}(w_1)(w_2 - w_1) + O(||w_2 - w_1||^2) \\
&= \mathcal{L}^{'}(w_1) + \overline{H}_1(w_2 - w_1) + O(\alpha^2) \\
&= \mathcal{L}^{'}(w_1) - \alpha\overline{H}_1 g_2 + O(\alpha^2) \quad \text{(using } w_2 - w_1 = \alpha g_2\text{)}
\end{aligned}
$$

Where $\overline{H}_1$ is Hessian of the sub-net 1 and α is the current learning rate. Similar to Reptile, the term $\alpha\overline{H}_1 g_2$ serves to maximize the dot-product of the consecutive sub-nets, where the expectation can be expressed as:

$$
\begin{aligned}
\mathbb{E}_{1,2}[\alpha\overline{H}_1 g_2] &= \mathbb{E}_{1,2}[\alpha\overline{H}_2 g_1] \\
&= \frac{1}{2}\mathbb{E}_{1,2}[\alpha\overline{H}_1 g_2 + \alpha\overline{H}_2 g_1] \\
&= \frac{1}{2}\mathbb{E}_{1,2}[\frac{\partial}{\partial w_1}(g_1 \cdot g_2)]
\end{aligned}
$$

We can observe that the term $-\alpha\overline{H}_1 g_2$ aims to maximize the inner product of two consecutive mini-batches. This indicates that the proposed AST scheme has an implicit regularization effect that aligns the weight updates between sub-nets.

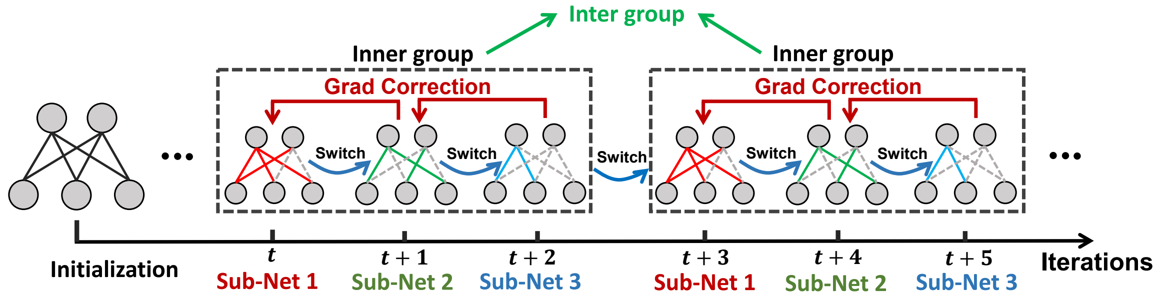


Figure 12. The Overview of the AST Process With Gradient Correction on Consecutive Inner-group Sub-nets (AST-GC)

.

3.3.2 AST with Gradient Correction

Although the implicit regularization of mini-batch SGD helps maximize the inner product of gradients between consecutive sub-nets, as discussed in Section **??**, we have observed that there can still be a partial number of negative inner products during training. This results in conflicting directions of weight updates among sub-nets, as shown in Figure 13. However, it is worth noting that in the context of optimizing a network using mini-batch SGD, the noise introduced by these conflicting gradient directions can actually be beneficial. It helps the optimization process escape from saddle points or local minima and can improve the generalization of the network [30, 6].

Due to these observations, we draw inspiration from the gradient projection method, which has been successfully applied in multi-task learning [125] and continual learning [66], and propose a *gradient correction* technique to mitigate the conflicting gradients within the inner-group iterations during training while still allowing negative gradients between inter-group sub-nets. As depicted in Figure 12, the *inner-group iterations* refer to the consecutive sub-nets within N mini-batch iterations, where N is the total number of sub-nets (e.g., $N == 3$ as shown in Figure 12). On the other hand,

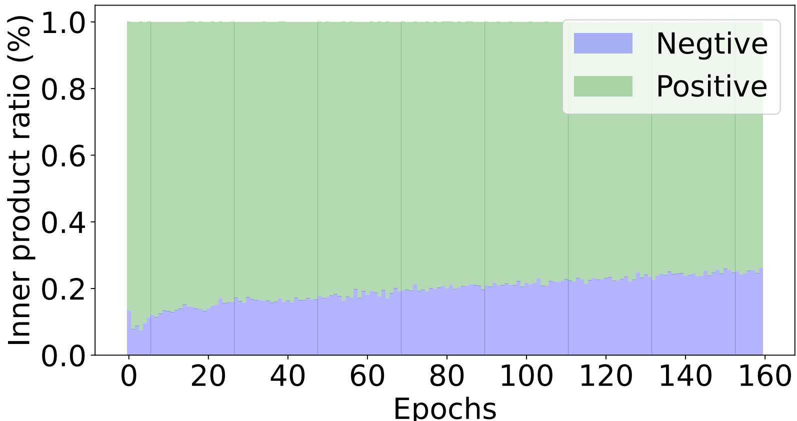


Figure 13. The Ratio of Negative and Positive Inner Product of Two Sub-nets During the AST Process on CIFAR-10 by Using Wide ResNet-32.

.

inter-group indicates the relationship between inner-group iterations. Specifically, the gradient correction employs a simple procedure within the inner-group sub-nets: if the gradients between two consecutive sub-nets are in conflict (i.e., their cosine similarity is negative), we project the gradient of the current sub-net onto the normal plane of the gradient of its prior sub-net. Otherwise, the original gradient remains unchanged. Considering two sub-nets in inner-group iterations with the gradients g_i and g_j respectively, the technique can be mathematically illustrated as:

$$g_i = \begin{cases} g_i - \frac{g_i \cdot g_j}{||g_j||^2} g_j, & \text{if } g_i \cdot g_j < 0 \\ g_i, & \text{otherwise} \end{cases} \tag{3.4}$$

Where $\tilde{g}_i$ represents the corrected gradient for sub-net i. By applying this gradient correction technique, we ensure that the gradients within the inner-group iterations are aligned and the conflicting gradients are corrected, while still allowing for negative gradients between inter-group sub-nets. This helps improve the optimization process and maintain the effectiveness of the alternating sparse training scheme.

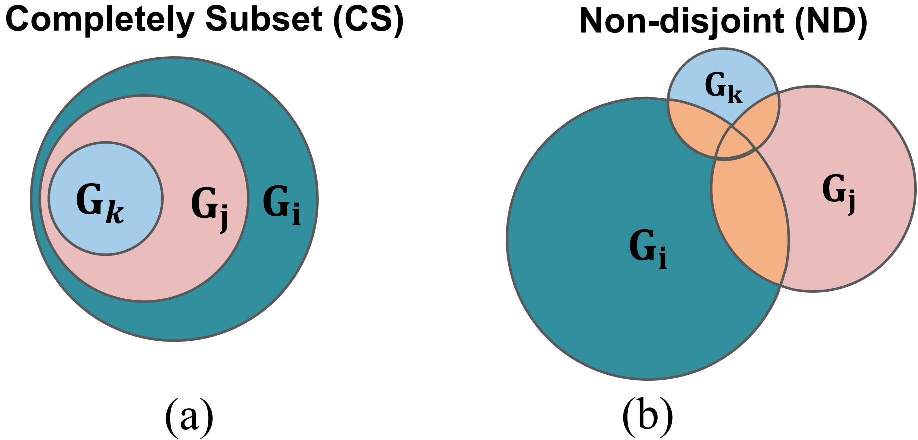


Table 8. Different Relationships Between Three Subnets G_i, G_j, and G_k: Completely Subset (CS) and Non-disjoint (ND).

3.3.3 Sparse Sub-net Training

Another important aspect of our AST scheme is to generate and train "sparse" sub-nets. Following most sparse training works, we adopt the "prune-and-regrow" scheme from GraNet [74] as the backbone technique. Specifically, AST starts from a randomly initialized sparse model and then applies the prune-and-regrow mechanism as described in Eq.3.1 and Eq.3.2 for each training iteration with the sub-net-specific sparsity ratio. Given the fact that the alternating training scheme switches the sub-net per iteration, the following questions arise:

1. What is the optimal architectural relationship between sub-nets?
2. How should pruning be scheduled for each sub-net?

Observation 1: *Enabling the freedom of exploring unique architectures of each sub-net elevates the learning capacity of AST over the S-Net.*

Table 9. Averaged Accuracy and Standard Deviations Between Different AST Training Schemes on CIFAR-10 Dataset (Three Times Experiments Each).

Dataset	**CIFAR-10**		
ResNet-32	Dense Model Acc. = 94.88		
Pruning Ratio	70%	95%	98%
Completely Subnet (CS)	93.69 ± 0.12	93.43 ± 0.03	92.49 ± 0.17
Non Disjoint (ND)	$\mathbf{94.47 \pm 0.10}$	$\mathbf{93.78 \pm 0.09}$	$\mathbf{92.76 \pm 0.21}$

To validate this observation, we generalize the sub-net relations into the following two categories:

- **Completely subset (CS):** As proposed by S-Net [124], the high-sparsity models are fully contained in the low-sparsity models (Figure 8(a)). Under the context of prune-and-regrow, the regrowing process is only performed in the lowest sparsity model, while the rest of the sub-nets rigorously extend the sparsity from the previous low-sparsity model by using magnitude-based pruning only. The one-time regrow guarantees the fully-subset relationships among different sub-nets.
- **Non disjoint (ND):** Each sparse sub-net performs prune-and-regrow individually to optimize the overall pruning decision during training. As depicted in Figure 8(b), sub-nets can freely exploit their own architectures while the intersections remain non-empty. Compared to the CS scheme, the non-disjoint AST empowers the subset networks with more architecture freedom. Figure 14 shows the layer-wise sparsity gap and non-overlap between two ND-trained sub-nets that target different final sparsity values (s_f). The non-overlap is computed by XORing the binary sparse masks between sub-nets, the percentages of "1" in the resultant tensor represent the level of non-overlap. Apparently, the distinction of

connections is larger than the sparsity difference, which implies the existence of the unique connections generated by the ND prune-and-regrow in each sub-net.

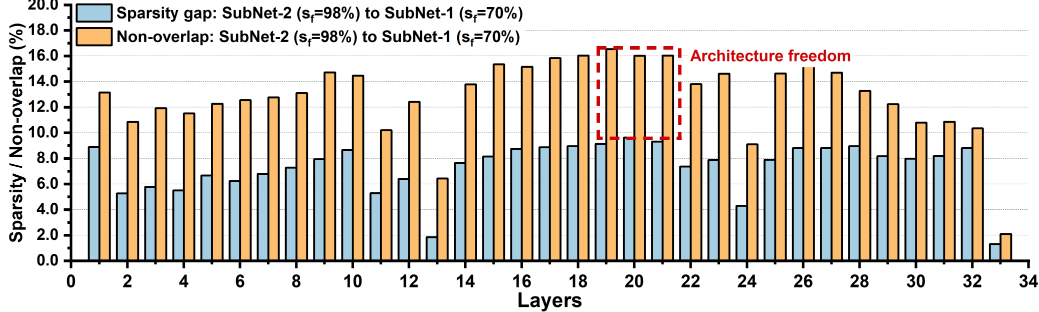


Figure 14. Layer-wise Sparsity and the Connection Dissimilarity Between Two ResNet-32 Sub-nets Trained by the Non-disjoint (ND) AST Scheme.

With the ND scheme, we observe that the small amount of architecture freedom shown in Figure 14 can lead to improved overall performance: Table 9 summarizes the accuracy of the wide ResNet-32 [127] trained by different AST schemes on CIFAR-10 dataset. Assisted by the comprehensive architecture exploration, the non-disjoint AST scheme achieves higher accuracy compared to the conservative completely-subset (CS) training. Thus, in this work, we use ND-scheme for all experiments.

Observation 2: *Intermittent sparsity increment among sub-nets stabilizes AST process.* The sparsity of the in-training sparsification method is periodically updated (e.g., 1,000 iterations) based on a pre-defined sparsity schedule [74]. Regarding the AST scheme, the sub-net model architectures are consecutively switched and trained, but the successive architecture switching does not imply the necessity of consecutive sparsity updates of each sub-net. On the contrary, the intensive sparsity increment of all sub-nets could destabilize the training. In this work, the sparsity of each sub-net increases periodically, but the sparsity increment of each sub-net is

intermittently performed with the adjustment period $\Delta\tau$. In the meantime, sub-nets are still consecutively switched during the $\Delta\tau$.

3.4 Experiments

3.4.1 Experimental Setup

The proposed AST method is extensively evaluated on multiple datasets, including CIFAR-10, CIFAR-100, and ImageNet. We adopt a similar training scheme to previous works such as Liu et al.[74] and Yuan et al.[127], where the models are trained for 160 epochs on CIFAR-10/100 and 100 epochs on ImageNet. The multiple sparse sub-nets are trained from scratch and pruned using the AST algorithm. For the CIFAR datasets, we utilize a cosine annealing learning rate schedule with an initial learning rate of 0.1. As for the ImageNet dataset, we include an additional warmup period of 5 epochs before applying the cosine annealing schedule. The pruning candidates are selected globally, while the first layer is kept dense. During the regrowing process, the percentage of regrow candidates gradually decreases from 0.5 to 0.0 following a cosine annealing schedule. Moreover, the extended adjustment period between sub-nets is set to 100 epochs for CIFAR experiments and 400 epochs for ImageNet experiments. While we acknowledge that more fine-grained hyperparameter tuning could potentially lead to better accuracy, we have chosen the above scheme for simplicity and reproducibility. The reported sub-net accuracy values are obtained from a single background model checkpoint. In all experiments, we report the average accuracy along with its variation across 3 runs to ensure the reliability of the results.

Table 10. CIFAR-10/100 Accuracy and Training Cost Comparison with SoTA Works on Wide ResNet-32.

Dataset	**CIFAR-10 Acc. (%)**			**CIFAR-100 Acc. (%)**			**Train. Cost**
ResNet-32	Dense Model Acc. = 94.88			Dense Model Acc. = 74.94			
Pruning Ratio	90%	95%	98%	90%	95%	98%	
Individual Training							
Lottery Ticket [28]	92.31	91.06	88.78	68.99	65.02	57.37	3×
SNIP [60]	92.59	91.01	87.51	68.89	65.22	54.81	3×
DSR [89]	92.97	91.61	88.46	69.63	68.20	61.24	3×
GraNet [74]($s_i = 0\%$)	94.12	94.02	92.98	73.18	72.56	69.89	3×
MEST [127]($s_i = 90\%$)	92.12 ± 0.13	90.86 ± 0.11	88.78 ± 0.26	69.35 ± 0.36	67.85 ± 0.23	62.58 ± 0.31	3×
MEST+EM[127]($s_i = 90\%$)	92.56 ± 0.07	91.15 ± 0.29	89.22 ± 0.11	70.44 ± 0.26	68.43 ± 0.32	64.59 ± 0.27	3×
MEST+EMS[127]($s_i = 90\%$)	93.27 ± 0.14	92.44 ± 0.13	90.51 ± 0.11	71.30 ± 0.31	70.36 ± 0.05	67.16 ± 0.25	3×
Training once for all							
Jointly-Trained [124]($s_i = 0\%$)	92.59 ± 0.21	92.58 ± 0.25	92.48 ± 0.24	70.40 ± 0.14	69.32 ± 0.84	66.85 ± 0.59	3×
AST($s_i = 0\%$)	93.51 ± 0.06	93.44 ± 0.08	92.44 ± 0.04	73.12 ± 0.10	72.39 ± 0.14	68.06 ± 0.21	**1×**
AST($s_i = 90\%$)	92.32 ± 0.06	92.19 ± 0.11	91.34 ± 0.04	69.82 ± 0.12	69.22 ± 0.07	66.37 ± 0.15	**1×**
AST+GC($s_i = 0\%$)	$\mathbf{93.88 \pm 0.19}$	$\mathbf{93.70 \pm 0.28}$	$\mathbf{92.69 \pm 0.09}$	$\mathbf{73.41 \pm 0.04}$	$\mathbf{72.57 \pm 0.15}$	$\mathbf{68.42 \pm 0.15}$	**1×**
AST+GC($s_i = 90\%$)	$\mathbf{92.90 \pm 0.13}$	$\mathbf{92.88 \pm 0.10}$	$\mathbf{91.97 \pm 0.18}$	$\mathbf{70.11 \pm 0.39}$	$\mathbf{70.01 \pm 0.54}$	$\mathbf{67.15 \pm 0.31}$	**1×**

3.4.2 Experimental Results

3.4.2.1 CIFAR-10/100

Table 10 presents the CIFAR-10/100 accuracy results achieved by the proposed AST algorithm using the wide ResNet-32 model, as utilized in [127]. Consistent with previous works reporting high sparsity results, we train three sub-nets simultaneously with sparsity ratios of 90%, 95%, and 98%. The training is conducted from scratch for all sub-nets, following the same number of epochs as in previous studies. To explore the benefits of high initial sparsity in terms of memory savings during the training process, we report the results for both dense ($s_i = 0\%$) and highly sparse ($s_i = 90\%$) initial models.

Compared to the state-of-the-art individually-trained models reported in prior works [74, 127], the proposed AST+GC algorithm achieves three highly sparse models through a single training process with negligible accuracy degradation. This results

in a significant reduction in the total training cost, providing benefits in terms of power consumption and latency. In contrast, the joint training scheme [124] requires multiple forward passes in each iteration, leading to increased computational cost and resulting in average performance. In addition to the reduced training cost, the proposed AST method outperforms the joint-training scheme [124] by up to 1.3% and 2.9% in terms of inference accuracy on the CIFAR-10 and CIFAR-100 datasets, respectively. Furthermore, the AST method achieves up to 2.63× reduction in training cost compared to the joint-training scheme. We also conduct experiments to verify the effectiveness of the AST scheme with a larger number of sub-nets and larger sparsity gaps.

3.4.2.2 ImageNet

We conducted further evaluation of the proposed method using ResNet-50 on the ImageNet dataset, as shown in Table 11. We observed that the 0.5× and 0.25× models of the jointly-trained S-Net [124], which correspond to weight sparsity of 72.98% and 92.23% respectively, resulted in an averaged overall sparsity of 82.6%, which is lower than our targeted sparsity of 85%. In comparison, the proposed AST training scheme outperforms the joint-training scheme by 7.5% in terms of inference accuracy when using 80% highly sparse initial models. Furthermore, the proposed AST scheme achieves comparable or even better performance than SNIP [60] and SET [86], while reducing the total training cost by up to 2.38×. Additionally, the inclusion of gradient correction (GC) in AST further improves accuracy by 0.6% and 0.8% respectively.

Table 11. ImageNet Accuracy and Training Cost Comparison with SoTA Works on ResNet-50.

Method	ImageNet-2012			
ResNet-50	Dense model Acc. = 76.8			
Prune Ratio	80%		90%	
	Individual Training			
	Top-1 Acc. (%)	**Training Cost**	**Top-1 Acc. (%)**	**Training Cost (×e18)**
SNIP [60]	69.7	1.67	62.0	0.91
SET [86]	72.9	0.74	69.6	0.32
DSR [89]	73.3	1.28	71.6	0.96
RigL [25]	74.6	0.74	72.0	0.39
MEST + EM [127]	75.8	0.74	73.6	0.39
GraNet [74]	76.0	1.18	74.5	0.80
	Training once for all			
Jointly-Trained [124]($s_i = 50\%$)	$71.9_{0.5\times}$	1.19	$65.0_{0.25\times}$	1.19
AST ($\mathbf{s_i = 50\%}$)	**72.6**	**0.59**	**72.3**	**0.41**
AST + GC ($\mathbf{s_i = 50\%}$)	**73.2**	**0.59**	**73.1**	**0.41**
AST + GC ($\mathbf{s_i = 80\%}$)	**72.6**	**0.37**	**72.5**	**0.13**

Table 12. Inference Acceleration and Negligible Accuracy Drop of the Proposed AST Algorithm with Structured Fine-grained Sparsity on ResNet-18 Model.

Dataset	**CIFAR-10 Acc. (%)**					**Training Cost**
N:M Sparse Pattern	Dense Model	2:4	3:4	7:8	15:16	
Individually Trained (SR-STE) [134]	95.07	94.89	94.47	94.25	93.92	2.33e+16 (3.95×)
AST + GC	-	94.63	94.26	94.31	93.79	**5.91e+15 (1×)**
Inference Time / 10K images (sec)	1.40	1.01	0.67	0.66	0.63	-

3.4.2.3 Extend to Structured Fine-grained Sparsity

The effectiveness of pruning algorithms can be demonstrated by showcasing the sparsity-induced speedup on GPUs. The Nvidia Ampere architecture features Sparse Tensor Cores, which accelerate neural network computations with N:M structured fine-grained sparsity. By varying the group size M and the number of sparse elements N, different overall sparsity values can be achieved. Building upon this, we extend the proposed AST algorithm to train multiple sub-nets with different N:M sparsity patterns simultaneously. The prune-and-regrow process is performed based on the group-wise summed weight or gradient magnitude.

With the assistance of the open-source Nvidia-ASP library[5], convolution computations can be accelerated when the sparse weight group size M is divisible by 4 (e.g., 4, 8, 16). In our proposed AST algorithm, we collectively train four ResNet-18 sub-nets with 2:4, 3:4, 7:8, and 15:16 sparse patterns, resulting in overall sparsity levels of 50%, 75%, 87.5%, and 93.75% respectively. Starting from scratch, the percentage of N:M sparse groups is gradually increased from 0% to 100% within each sub-net. Compared to individually trained dense models, our proposed AST scheme achieves up to a 2.3× inference speedup on GPUs with 4× less training efforts and negligible accuracy loss, as shown in Table 12. This demonstrates the effectiveness of the AST algorithm in achieving both sparsity-induced speedup and maintaining inference accuracy. The inference time is measured on a Nvidia 3090 GPU with FP32 data precision.

3.5 Ablation Study and Discussion

3.5.1 AST-GC with More Sub-nets

The effectiveness of the proposed AST scheme is verified by training different numbers of subset networks. Table 14 presents the CIFAR-10 performance of AST when training 2, 3, 4, and 5 sparse sub-networks collectively using ResNet-18. Comparing this approach to the individually-trained baseline sparse model, we observe that training more sub-nets with AST reduces the total training effort, albeit with a slight decline in accuracy, particularly in high sparsity models (e.g., 98% sparsity). However, these marginal accuracy reductions are outweighed by the significant advantage of reduced training cost. Notably, the tradeoff between accuracy and total training cost indicates that training three sub-nets achieves the best overall performance.

[5]https://github.com/NVIDIA/apex/tree/master/apex/contrib/sparsity

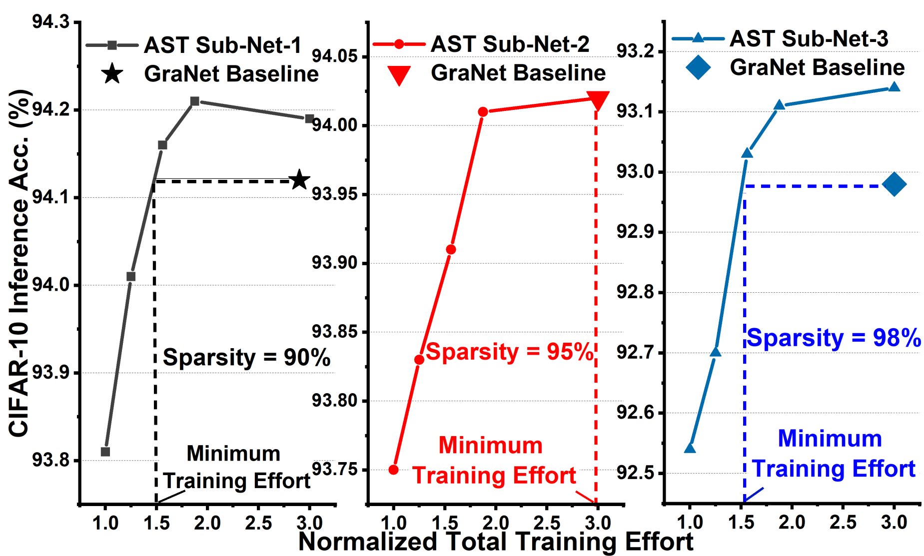


Table 13. AST with Extended Training Effort on CIFAR-10 with Wide ResNet-32 [127].

Table 14. ResNet-18 Training Results of AST (160 Epochs) with Various Sparsity Values and Numbers of Sub-nets for CIFAR-10 Dataset.

Dataset	**CIFAR-10 Acc. (%)**				
Sparsity	Indiv. trained	2 sub-nets	3 sub-nets	4 sub-nets	5 sub-nets
70%	95.11	94.88	94.75	94.79	94.81
80%	94.94	-	-	94.81	94.73
90%	94.93	-	-	-	94.85
95%	94.88	-	94.67	94.63	94.47
98%	94.50	94.76	94.26	94.22	94.18

3.5.2 AST Sub-nets with Different Sparsity Differences

In addition to collectively training highly sparse models, learning subset architectures with large sparsity gaps is also essential. As shown in Table 14 and Table 15, the proposed AST algorithm effectively optimizes model performance even with large sparsity gaps, such as 50% vs. 95%. This demonstrates the robustness and adaptability

Table 15. ImageNet Accuracy with Different Sparsity Combinations on ResNet-50.

Method	**ImageNet-2012**					
ResNet-50	**Dense Model Acc. = 76.8**					
Prune Ratio	50%	90%	50%	95%	80%	90%
AST+GC (s = 0%)	74.68	73.26	-	-	-	-
AST+GC (s = 0%)	-	-	74.21	71.07	-	-
AST+GC (s = 50%)	-	-	-	-	73.2	73.1
AST+GC (s = 80%)	-	-	-	-	72.6	72.5

of AST in handling diverse subset architectures, enabling efficient resource utilization across different power budgets.

3.5.3 Extended AST Training Efforts

In Figure 11, AST iteratively selects different sub-nets for each mini-batch iteration. Despite employing batch shuffling during training, each sub-net cannot be fully trained using the entire training set within each epoch. One straightforward solution to address this is to extend the total training efforts (i.e., increase the number of epochs). Assuming the unit training cost is 160 epochs ($1\times$), we evaluate the performance of AST with three wide ResNet-32 [127] sub-nets that have different training efforts, as shown in Figure 13. Compared to the individually trained GraNet [74] baseline (total=$3\times$), AST achieves the same accuracy in all three sparse models with only approximately $2\times$ the average total training effort. This demonstrates the efficiency and effectiveness of the AST algorithm in achieving comparable performance to individual training while reducing the overall training time.

3.6 Summary

In this chapter, we present the Alternating Sparse Training (AST) scheme, which allows the simultaneous training of multiple sparse neural networks. We also introduce gradient correction (GC) as a complementary technique to improve the performance of AST. Unlike previous approaches that involve repeated or ensembled training steps, AST achieves high accuracy and training efficiency. Building upon the prune-and-regrow scheme, the proposed AST-GC scheme leverages multiple sparse sub-nets simultaneously and achieves comparable or even higher accuracy compared to individually-trained state-of-the-art methods. The use of AST provides practical benefits for energy-efficient hardware computation while maintaining superior performance in terms of accuracy.

Chapter 4

ON-DEVICE LEARNING FOR TASK ADAPTION

Transfer learning plays a crucial role in on-device machine learning, allowing well-trained deep learning models to be applied to new tasks. However, the limited memory capacity of IoT/edge devices poses challenges for memory-efficient learning. While many existing approaches focus on reducing trainable parameters, this does not directly address the memory bottleneck, which is often caused by activations rather than parameters. To tackle the issue of memory-efficient on-device transfer learning, we propose a novel concept called intermediate feature reprogramming. In this work, we introduce the Reprogramming Network (Rep-Net), a lightweight model that is trained directly from the new task input data while keeping the backbone model frozen. The Rep-Net model exchanges feature with the backbone model using an activation connector at regular intervals, benefiting both models in a mutually beneficial manner. Through extensive experiments, we validate the design specifications of the proposed Rep-Net model, demonstrating its effectiveness in achieving highly memory-efficient on-device reprogramming.

4.1 Preliminaries

4.1.1 Memory Efficient Learning

In recent years, several approaches have been proposed to address the issue of training memory consumption. These methods can be broadly categorized into

activation re-computation and activation compression techniques. Activation re-computation techniques, such as those proposed in [15] and [32], aim to reduce memory usage by eliminating the storage of partial or full activations. Instead, activations are re-computed as needed during the backward pass. While this approach helps save memory, it comes at the cost of additional computation. On the other hand, activation compression methods, as presented in [72], focus on reducing the size of activations through pruning techniques. By removing unimportant or redundant activation values, the memory footprint can be reduced. Another approach, introduced in [24], leverages image compression algorithms to compress activations and then decompress them when needed during training. It is important to note that our method is orthogonal to activation compression techniques and offers a different solution to the problem of training memory consumption.

More recently, studies conducted by [10] and [111] have delved deeper into the analysis of memory utilization during training and have emphasized that activations, rather than the size of parameters, dominate the training memory consumption. To gain a better understanding of training memory consumption, let's consider a linear layer with the forward process modeled as $a_{i+1} = a_i \cdot W + b$. The backpropagation process of this linear layer can be expressed as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = a_i \frac{\partial \mathcal{L}}{\partial a_{i+1}}, \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a_{i+1}} \tag{4.1}$$

Where a represents the activation feature, $\mathbf{W}$ represents the learnable weights, and b represents the bias term. Equation 4.4 highlights that memory-intensive activations need to be stored for backward propagation during training when they have a multiplicative relationship with the learned parameters (i.e., weights). Conversely, activations with an additive relationship (e.g., bias) do not require additional memory for storage [111]. In this chapter, the Rep-Net is designed to have a purely additive

relationship with the fixed pre-trained model, resulting in improved training memory efficiency.

4.1.2 Transfer Learning

4.1.2.1 Transfer Learning via Fine-tuning

Currently, fine-tuning a deep neural network that has been pre-trained on large-scale datasets such as ImageNet [19] is the standard method for knowledge transfer [31, 55, 13, 133, 90, 18, 29, 36]. There are two main approaches to fine-tuning.

The first approach treats the pre-trained model as a fixed feature extractor and only fine-tunes the last classification layer [13, 133]. This method is memory-efficient as it eliminates the need to store intermediate activations of the pre-trained model. However, it has been demonstrated that this method has limited transfer capacity in previous works [55]. The second approach involves fine-tuning the full or partial pre-trained model [36, 55, 90, 18]. This method can achieve better accuracy. For example, [31] fine-tunes the entire model on domain-specific data to improve object detection accuracy. [18] proposes to update only the batch normalization layers. Additionally, [36] automatically determines the optimal set of layers to fine-tune on a new task. However, all of these methods involve updating the pre-trained model, resulting in a significant memory footprint for activations. Therefore, they are not suitable for on-device learning scenarios.

4.1.2.2 Transfer Learning via Input Reprogramming

The concept of reprogramming was initially introduced in the context of adversarial input reprogramming [23]. This approach trains additive features to modify the input data and transfer a model's knowledge to a new domain or dataset. The inspiration for this idea comes from adversarial example attacks [81], which showed that even small additive perturbations to the input can completely change the behavior of a target model.

Let's consider an input batch to a neural network as $\mathbf{x} \in \mathbb{R}^{k \times k \times c}$, where it is sampled from a dataset with an input dimension of $k \times k \times c$. Now, suppose we want to apply a pre-trained network with a forward function h_θ to this new domain data $\mathbf{x}$ without modifying the inference function $h(\cdot)$ or the parameter θ. One possible solution would be to introduce a new set of learnable additive parameters to the input, denoted as $\mathbf{p}$:

$$\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{p} \tag{4.2}$$

The objective of reprogramming is to minimize the loss function $\mathcal{L}_R$ for the correct label $\boldsymbol{y}$ by updating the input parameter $\boldsymbol{p}$:

$$\min_{\{\boldsymbol{p}\}} \ \mathbb{E}_{\mathbb{X}}\Big(\mathcal{L}_R(h_\theta(\hat{\boldsymbol{x}}), \boldsymbol{y})\Big) \tag{4.3}$$

The above process can be extended to the entire dataset $(\mathbf{X}, \mathbf{Y})$ of the new domain by training a single bias vector $\mathbf{p}$. The concept of input reprogramming is simple and does not require any modifications to the backbone model architecture or parameters. Building upon this idea, [54] proposes to improve the transfer capacity by resizing the input during reprogramming and retraining the last classification layer. Additionally, [104] extends input reprogramming to black-box transfer learning, where only the

input-output model responses are observable. However, these input reprogramming methods have some limitations. First, the learning capability of solely adding an input bias is limited, even after applying non-linear transformations (e.g., Hard Tanh) before addition [23]. Second, the addition of $k \times k \times c$ learnable parameters can be substantial, especially when dealing with large-scale datasets (e.g., in ImageNet models, $224 \times 224 \times 3$ corresponds to approximately 150k parameters). Third, the performance of input reprogramming is often inferior and fails to achieve satisfactory results on simple tasks like MNIST [20]. To overcome these challenges, we propose to train a lightweight side network using the input data from the new task and reprogram the intermediate activation features.

4.2 Training Memory Analysis

In this section, we begin by examining the training memory usage of various multi-domain learning methods. We then proceed to perform a quantitative analysis of the memory usage for each layer of the DNN model. This analysis will provide insights and guide us towards exploring potential solutions for achieving memory-efficient on-device learning methods.

4.2.1 Fine-tuning and Adaptor-based Methods

Both fine-tuning and adaptor-based training schemes are widely used in the field of multi-domain learning. Fine-tuning involves adjusting all or a subset of parameters in the pre-trained model to adapt it to the target dataset domain. On the other hand, adaptor-based methods employ additional convolution layers and modify the original

batch normalization (BN) layers during the fine-tuning process. This approach allows for more flexibility and customization in adapting the model to the target domain. To understand the training memory consumption, let's assume a linear layer whose forward process be modeled as: $a_{i+1} = a_i \times \mathbf{W} + b$, then its back-propagation process is

$$\begin{gathered}\frac{\partial \mathcal{L}}{\partial a_i} = \frac{\partial \mathcal{L}}{\partial a_{i+1}} \frac{\partial a_{i+1}}{\partial a_i} = \frac{\partial \mathcal{L}}{\partial a_{i+1}} \mathbf{W}^T, \\ \frac{\partial \mathcal{L}}{\partial \mathbf{W}} = a_i \frac{\partial \mathcal{L}}{\partial a_{i+1}}, \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a_{i+1}}\end{gathered} \tag{4.4}$$

Eq. 4.4 highlights the memory requirements in conventional backpropagation-based training, where both model weights ($\mathbf{W}$), gradients, and activations (a_i) need to be stored for computation, resulting in significant memory usage. However, an interesting observation is that when updating only the bias term, which has an additive relationship with the activation (a_i), there is no need to store the previous activation since it is not involved in the backward computation. This same phenomenon can also be observed in both convolutional (Conv) and batch normalization (BN) layers.

4.2.2 Mask-based Learning Method

For the mask-based learning method, let's consider a linear layer with the following forward process: $a_{i+1} = a_i(\mathbf{W} \times \mathbf{M}) + b$, where $\mathbf{M}$ is the mask to be learned, and $\mathbf{W}$ represents the fixed weights. In this case, we only train the mask $\mathbf{M}$ while keeping the weights $\mathbf{W}$ fixed. The backward process can be described as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{M}} = a_i \frac{\partial \mathcal{L}}{\partial a_{i+1}} \times \mathbf{W}, \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a_{i+1}} \tag{4.5}$$

Eq.4.5 shows that learning mask needs to store not only activation-a_i, but also the mask-$\mathbf{W}$ and weights-$\mathbf{W}$ during training. In terms of computation, comparing Eq.4.5

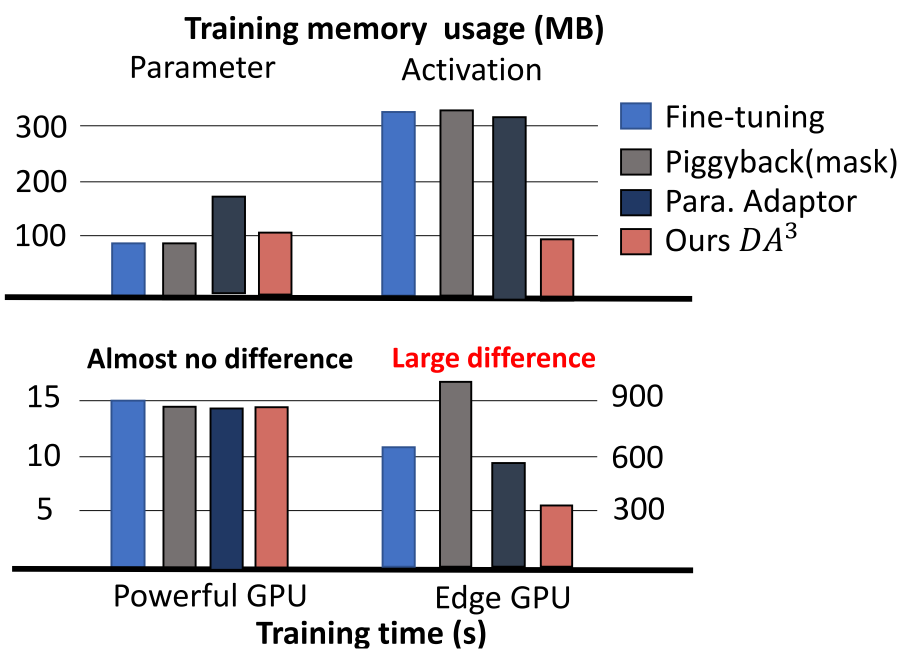


Figure 15. An Example of Adapting ResNet50 (Pre-trained on ImageNet dataset) to Flower Dataset. *Top:* Model Parameters and Activation Memory of Three Different Methods. *Bottom:* Training Time of One Epoch on Two Different Platforms: One Powerful GPU (Quadro RTX 5000) and One Edge GPU (Jetson Nano)

with Eq.4.4, such method also needs additional multiplication computation in both forward and backward pass. These observations explain why Piggyback has the largest training time in edge GPU as shown in Fig.15. Other mask-based methods [84, 119] even need more computation cost than Piggyback, since they involve additional reparameterization techniques. In addition, similar to fine-tuning and adaptor-based methods, training bias does not involve activation storage.

4.2.3 Training Memory Usage Analysis.

To evaluate the training cost of on-device multi-domain learning, we conducted experiments using three representative methods in three different directions. These

methods were tested on both powerful GPUs (such as Nvidia RTX5000 used in desktop or cloud server training) and edge GPUs (such as Nvidia Jetson Nano GPU used in edge device training). The training memory usage and training time were measured and are shown in Figure 15.

Observation 1: *The training process is memory-intensive, with the bottleneck being the intermediate activation buffering in memory during back-propagation. This bottleneck is at least 3 times larger than the model itself, as shown in Figure 15. This memory limitation significantly impacts the speed of on-device learning.*

During training, the memory usage for activation storage, which we refer to as "activation memory," is approximately three times larger than the model itself. This large training memory is not an issue in powerful GPUs with sufficient memory capacity and training time. However, it becomes a bottleneck for memory-limited edge GPUs commonly used in edge device training. As a result, different training methods for the same network and dataset exhibit significantly different training speeds, as shown in Figure 15. Most prior domain adaptation schemes focus solely on improving accuracy with minimal parameter updates, while neglecting the computational and memory-intensive nature of their methods. This lack of efficiency hinders their deployment on resource-limited edge-based training devices, such as mobile phones, embedded systems, and IoT devices.

Table 16. Summary of The Parameters and Activation Memory Consumption of Different Layers.

Layer Type	Trainable Param. (p)	Activation (a)
Conv	$c_{\text{in}} \times c_{\text{out}} \times kh \times kw$	$n \times c_{\text{in}} \times h \times w$
FC	$c_{\text{in}} \times c_{\text{out}} + c_{\text{out}}$	$n \times c_{\text{in}} \times h \times w$
BN	$2 \times c_{\text{out}}$	$n \times c_{\text{in}} \times h \times w$
ReLU	0	$n \times c_{\text{in}} \times h \times w$
Sigmoid	0	$n \times c_{\text{in}} \times h\times$ w

Moreover, we introduce the concept of training memory usage, which will be used throughout the paper. In Table 16, we observe that the memory usage during training is proportional to the number of parameters. We can categorize the parameters into two main groups: i) the number of trainable parameters (p), which includes weights and biases, and the corresponding gradients; ii) activation memory, which consists of the feature maps stored for updating the parameters of previous layers using the chain rule. In Table 16, we only list the number of trainable parameters (p).

In most convolutional layers, the kernel height and width are typically much smaller than the activation channel height and width (i.e., kh $\ll$ h; kw $\ll$ w). As a result, for a moderate batch size (e.g., $n = 64/128/256$), the memory size required to store the activations (a) is much larger than that of the trainable parameters (p). What's interesting is that even though batch normalization (BN) and sigmoid functions have a negligible number of trainable parameters (p), both functions produce activation outputs (a) of the same size as a convolutional or fully connected layer.

From the analysis above, it is evident that the storage of activation feature maps, rather than the model parameters themselves, dominates DNN training memory usage. Therefore, optimizing the memory usage of activation feature maps becomes crucial for achieving memory-efficient learning. Existing multi-domain learning methods, such as mask-based and fine-tuning methods, entail significant memory consumption during backward propagation, involving the storage of weights, gradients, and activation maps. Moreover, the mask-based method requires additional memory for storing masks. Interestingly, it is worth noting that if it is feasible to update only the bias during multi-domain learning, the predominant memory usage component - activation storage - becomes unnecessary. This is because bias exhibits an additive-only relationship

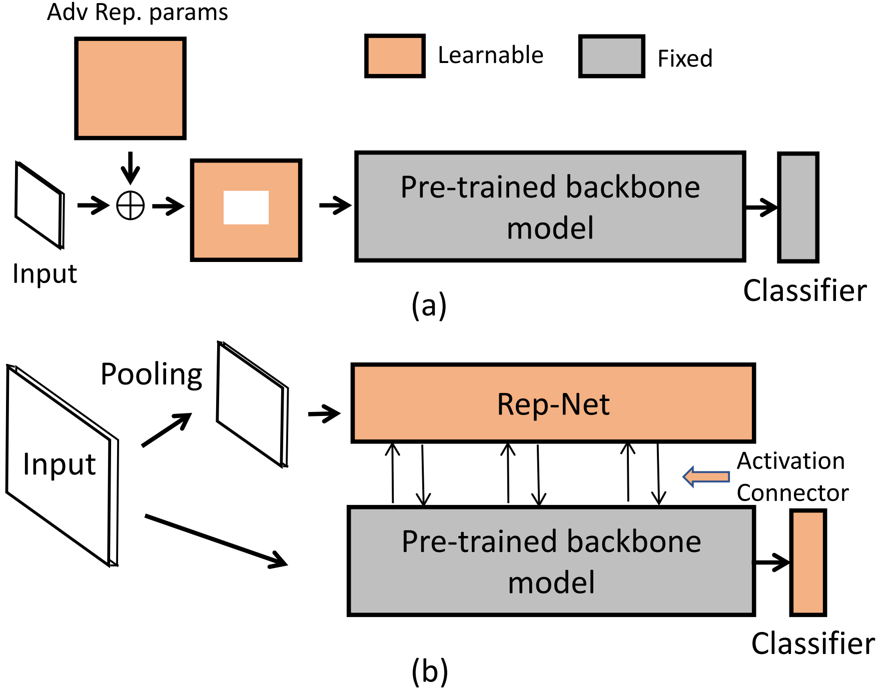


Figure 16. The Workflow of Adversarial Reprogramming (a) and the Rep-Net (b). Adversarial Reprogramming Reprograms the Input by Introducing An Additive Learnable Parameter. Differently, Rep-Net Takes the Input Data Directly and Learns to Reprogram the Intermediate Activation Features.

with input activation, enabling independent backward propagation. Based on above analysis, we summarize the underlying reason as the observation-2 below.

Observation 2. *The complete activation map needs to be stored for backward propagation during training if it has the* ***multiplicative relationship*** *with learned parameter (i.e., weight, mask), while the* ***additive relationship*** *(e.g., bias) is activation free.*

4.3 On-device Learning via Feature Reprogramming

In order to address the limitations of existing domain adaptation methods and enable efficient on-device transfer learning, we propose the Reprogramming Network (Rep-Net) approach. This novel approach takes a new perspective on the problem by focusing on the reprogramming of intermediate features.

As illustrated in Figure 16(b), the Rep-Net serves as a lightweight side-network that runs in parallel with the pre-trained backbone model. The key idea is to reprogram the fixed backbone model using the input data through an "activation connector" mechanism. This activation connector facilitates the exchange of features between the backbone model and the Rep-Net at regular intervals. The feature exchange process is based on an additive operation, which allows both the backbone model and the Rep-Net model to update and improve their respective features. This approach inherits the memory-efficiency property from adversarial input reprogramming. To enable on-device transfer learning, we only train the Rep-Net model and a task-specific last classification layer for the new task, while keeping the backbone model frozen. This ensures that the transfer learning process is efficient in terms of memory usage on resource-limited devices.

4.3.1 Architecture Overview

In this section, we propose a lightweight *Reprogramming Network (Rep-Net)* that is specifically designed for on-device learning. The Rep-Net learns to reprogram the activation features of the backbone model directly from the input data, as illustrated in Figure 16. The working mechanism of the Rep-Net involves inter-exchanging features

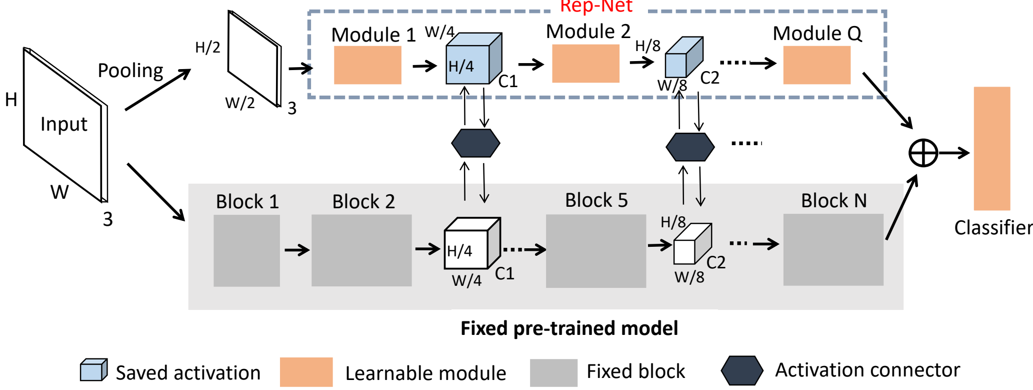


Figure 17. The Overview of Reprogramming Network (Rep-Net). The Proposed Rep-Net Model Learns Directly From the Input. The input Image is Directly Ded Into Both Rep-Net and the Backbone Model Parallelly. In Addition, Rep-Net Consists of A Small Number of Layers Positioned at Specific Locations Where the Backbone Model Observes a Feature Resolution Reduction.

with the backbone model at regular intervals. This feature exchange operation is performed using an additive operation, which allows both the backbone model and the Rep-Net to improve their respective features. During the reprogramming process, we only train the Rep-Net and a task-specific last classification layer for the new task, while keeping the backbone model frozen.

The proposed Rep-Net has several characteristics that make it suitable for on-device learning. Firstly, during training, the features from the Rep-Net help adapt the backbone model features to the new task, even though the backbone model is frozen. Secondly, the pre-trained backbone model provides valuable features to the Rep-Net, aiding its learning process for the new task. Thirdly, we optimize the positioning of the feature exchange operation based on observations from prior pruning works, ensuring that the Rep-Net requires only a few blocks or layers to learn the new task. This results in a lightweight network with minimal memory overhead. Lastly, we ensure that the feature exchange operations follow an additive rule, which contributes to memory-efficient on-device learning. As discussed in Section 4.1.1, the additive nature

of the learnable parameters eliminates the need to store intermediate activations in memory during back-propagation. In the following section, we provide detailed design specifications of the Rep-Net and validate each aspect through relevant ablation studies.

4.3.2 Activation Connector

In our design, the Rep-Net model exchanges its intermediate activation features with the backbone model using an *activation connector*. Let's consider a deep neural network with N layers as the backbone model, and our proposed Rep-Net model has Q layers ($Q \leq N$). In this case, the Rep-Net will exchange its features with the backbone model Q times at the activation connectors. We set the number of activation connectors to be equal to the number of layers in the Rep-Net (i.e., Q) (*Reason: refer to design intuition **3***). When the backbone model sends features $\boldsymbol{x}_i$ to the activation connector at the ith layer, and the Rep-Net model sends features $\boldsymbol{p}_i$ to the same connector (where ($i = 1, 2, 3, ..., Q$)), the forward path operation inside the activation connector performs feature reprogramming as follows:

$$\hat{\boldsymbol{x}}_i = \hat{\boldsymbol{p}}_i = \boldsymbol{x}_i + \boldsymbol{p}_i \tag{4.6}$$

The activation connector then sends the reprogrammed features $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{p}}_i$ to the backbone and Rep-Net models, respectively. This operation enhances the quality of features for both the backbone and Rep-Net models during training (*Evidence: refer to design intuition **2***).

4.3.3 Reprogramming Step

To reprogram the backbone model, which has an inference function h_θ parameterized by θ, we omit θ from our optimization step. Instead, we only train the Rep-Net model function g_β parameterized by β and a shared last classification layer with parameters α. We represent the combined function of the backbone and Rep-Net models as $f_{\theta,\beta,\alpha}$. For a given input and target pair $(\boldsymbol{x}, \boldsymbol{y})$, the reprogramming optimization step can be summarized as follows:

$$\min_{\{\beta,\alpha\}} \mathbb{E}_{\mathbb{X}}\Big(\mathcal{L}(f_{\theta,\beta,\alpha}(\boldsymbol{x}), \boldsymbol{y})\Big) \tag{4.7}$$

where $\mathcal{L}(.)$ is the cross-entropy loss function, which is minimized by updating the parameter set β, α. Now that we have summarized the design and training steps of the Rep-Net, we will introduce the design intuitions for optimizing the Rep-Net model to achieve an efficient on-device learning scheme while improving performance at the same time.

4.3.4 Design Intuition

Our proposed Rep-Net architecture has numerous design choices which could impact the eventual performance and efficiency of the reprogramming scheme. To validate our adopted design specs, we discuss three key questions:

1. *How to design the activation connector? (Refer to Design Intuition-1)*
2. *Is it necessary to reprogram both the features (i.e., $\hat{\boldsymbol{x}}_i$ & $\hat{\boldsymbol{p}}_i$)? What if we only reprogram either $\boldsymbol{x}$ or $\boldsymbol{p}$? (Refer to Design Intuition-2)*

3. *How many activation connector in Rep-Net is sufficient? or How many layers the Rep-Net model should have? (Refer to Design Intuition-3)*

Next, we present three design intuitions with relevant ablation study to validate our design choices by discussing those three questions.

4.3.4.1 Design Intuition 1

Typically, the feature exchange operation of the activation connector has two main choices: multiplication or addition. However, considering the memory efficiency during training as discussed in Section 4.1.1, we choose the addition operation for the feature exchange. This is because the additive relationship between the features allows us to avoid storing the complete activation of the pre-trained model during training, which significantly improves training memory efficiency.

4.3.4.2 Design Intuition 2

Following the Design Intuition 1, we consider different designs for the feature exchange operation in the activation connector. We present four alternative designs in Figure 18 to evaluate their effectiveness.

First, in Figure 18(a), we have the "no-connection" design where there is no exchange of intermediate features between the Rep-Net and backbone model. In this case, the Rep-Net model only interacts with the backbone model just before the last classification layer using an additive operation. In Figure 18(b), we have the "down connect" case where only the backbone model features are modified using Rep-Net features (i.e., $\boldsymbol{x} + \boldsymbol{p}$), while keeping the Rep-Net features unchanged. This design

Table 17. The Ablation Study to Validate the Four Design Choices. 'adv Rep + Last' Is the Adversarial Reprogramming Combining with Re-training Last Classifier. The 6^{Th} Row Shows the Results for Proposed Rep-Net. The Last Row (7^{Th}) Shows the Result for Using Dual Connector at All the Convolution Layer for the Backbone Model. We Use the Imagenet Pre-trained Proxylessnas-mobile as the Backbone Model.

Method	Train. mem	Flowers	Cars	CUB	Pets	Aircraft
Adv Rep + Last	36MB	91.1	56.0	66.1	88.2	44.4
No connect (a)	34MB	93.3	51.1	65.5	88.0	36.9
Down connect (b)	34MB	91.8	73.4	67.8	89.6	56.3
Up connect (c)	34MB	95.2	76.8	71.4	90.7	59.3
Dual connect (d) (proposed Rep-Net)	*34MB (↑ 2-3MB)*	*96.1*	*85.8*	*77.1*	*91.8*	*77.4*
Dual connect - all layers	37MB	96.0	85.8	76.8	91.2	78.7

highlights the importance of the proposed Rep-Net model features in reprogramming the backbone model. In Figure 18(c), we have the "up connect" case where only the Rep-Net features are modified while keeping the intermediate backbone features unchanged. This design evaluates the importance of backbone model features in training the Rep-Net model on the new task. Finally, in Figure 18(d), we present the "dual connection" case where both models mutually benefit by interchanging features between them. To validate the effectiveness of these designs, we conducted an ablation study summarized in Table 17. The study was performed on five datasets, and the results show that the dual connection design outperforms all the other cases. Additionally, it is observed that the "up connection" design outperforms the "down connection" design. This is because the Rep-Net model, being a smaller network, benefits from the pre-trained features of the backbone model to improve its learning capability. Furthermore, Figure 19 provides additional validation of the feature exchange operation, showing that the model converges faster when using the dual connection design. Overall, these results demonstrate the effectiveness of the dual connection design in enabling mutual benefit and faster convergence during on-device transfer learning.

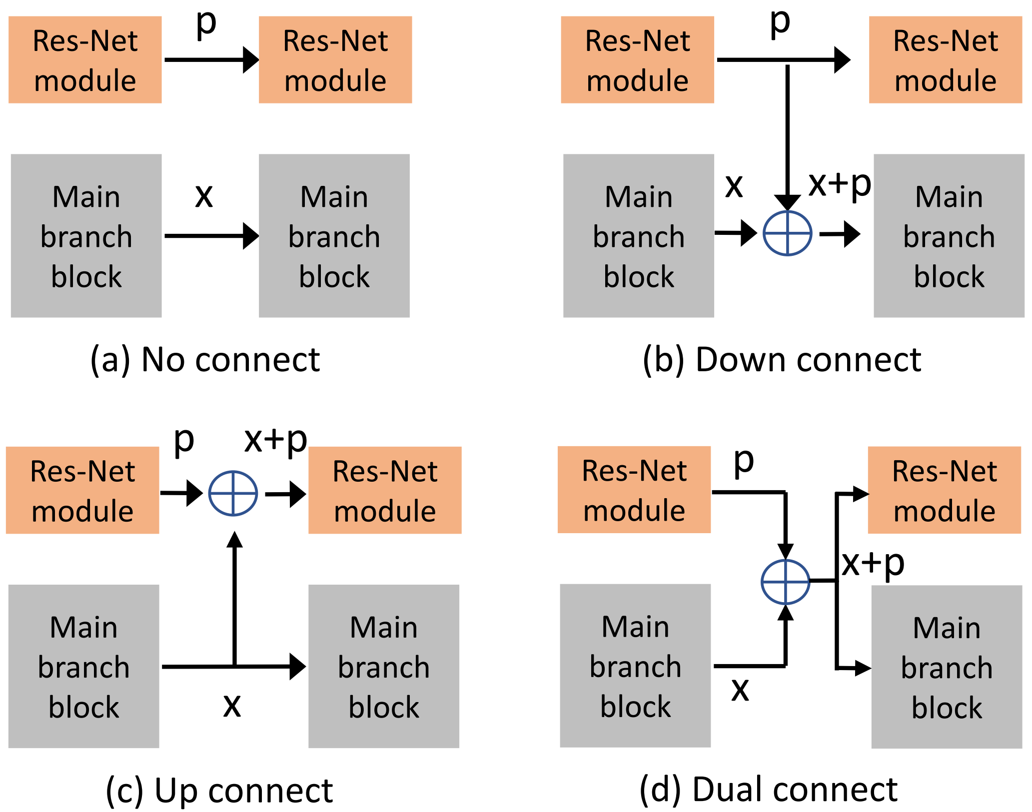


Figure 18. Design Choices for Feature Exchange Operation in Activation Connector.

4.3.4.3 Design Intuition 3

We adopt the following strategy to determine the number of activation connectors and number of layers in Rep-Net:

Strategy: The number of activation connectors in the Rep-Net is determined by the number of down-sampling operations in the pre-trained backbone model. Each down-sampling operation, such as a pooling layer or a convolutional block with a stride of 2, corresponds to one activation connector in the Rep-Net. The purpose of these activation connectors is to connect the features from the Rep-Net module with

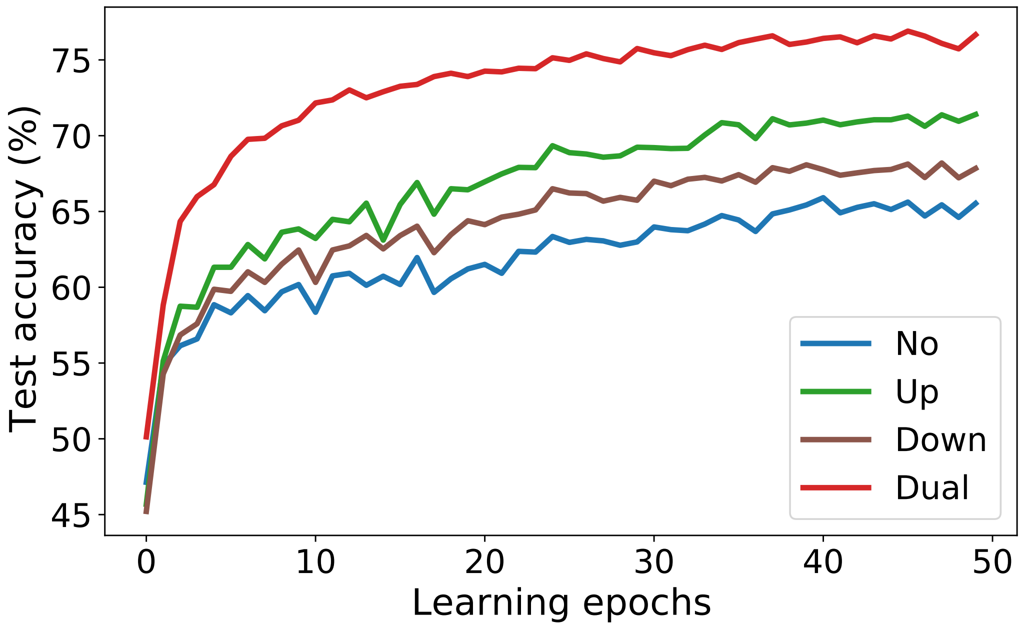


Figure 19. The Test Accuracy Vs Learning Epochs under Four Difference Design Choices.

the output activation feature of the corresponding down-sampling layer or block in the backbone model. By doing so, the Rep-Net features can be effectively exchanged and integrated with the backbone model features, enabling efficient feature reprogramming during on-device transfer learning.

The design strategy of using activation connectors at the down-sampling layers in the Rep-Net is motivated by the observation from model pruning works, such as [76] and [121]. These works have found that down-sampling operations in the backbone model can lead to a reduction in the resolution of the activation feature maps. To compensate for this resolution degradation, more channels are needed to carry the same amount of information. In our Rep-Net design, we address this issue by exchanging and updating features with the Rep-Net model through the activation connectors.

To validate the effectiveness of this design strategy, we compare it with a more

Table 18. Comparison with Input Reprogramming Works. 'adv. Rep' Is the Original Adversarial Reprogramming Work; 'adv Rep + Last' Is the Improved Adversarial Reprogramming Work That Further Re-train the Last Classifier.

Method	Net	Train. Mem.	MNIST	CIFAR10	Flower	CUB	Aircraft
Adv. Rep[23]	ResNet50	98MB	94.3	62.8	-	-	-
	MobileNetV2	53MB	93.1	58.3	-	-	-
	MobileNetV3	42MB	93.5	59.4	-	-	-
Adv. Rep + Last[54]	ResNet50	99MB	-	92.8	91.1	66.9	55.8
	MobileNetV2	54MB	-	85.1	91.8	66.8	52.2
	MobileNetV3	46MB	-	92.0	92.6	62.3	54.2
Ours	ResNet50	119MB	-	***96.4 (↑ 3.6 %)***	***96.7***	***80.3***	***85.9***
	MobileNetV2	***51MB***	-	***95.0 (↑ 9.9 %)***	***95.0***	***76.9***	***81.6***
	MobileNetV3	***43MB***	-	***95.3 (↑ 3.3 %)***	***95.1***	***77.5***	***78.9***

complex version of the Rep-Net model, where connector modules are added to all convolutional blocks. The results shown in Table 17 demonstrate that having connector modules only at the down-sampling layers (6th row) can achieve better performance compared to the Rep-Net model with connector modules at all layers (7th/last row). This suggests that the strategic placement of activation connectors at the down-sampling layers is sufficient to effectively exchange and update features, without the need for connectors at every layer.

Another advantage of using activation connectors at the down-sampling layers is that there are typically fewer of these layers compared to the total number of layers/blocks in modern model architectures. This means that the Rep-Net only needs a small number of modules, resulting in improved inference memory and computing efficiency. For instance, in the case of ResNet-50 architecture, the Rep-Net consists of only 6 modules. As highlighted in Table 17, using only 6 modules in the Rep-Net leads to a memory saving of approximately 3MB, while achieving identical accuracy compared to the Rep-Net with connectors at all layers. This demonstrates the efficiency and effectiveness of the Rep-Net design, as it requires fewer resources while still achieving comparable performance.

4.4 Experiments

4.4.1 Experimental Setup

4.4.1.1 Datasets and Networks.

In our experiments, we adopt the standard practice in previous transfer learning methods [55, 18, 90, 10] and use the ImageNet dataset [19] as the pre-training dataset for all the models. We then transfer these pre-trained models to 8 downstream object classification tasks, including Cars [56], Flowers [92], Aircraft [82], CUB [105], Pets [93], Food [5], CIFAR10 [57], and CIFAR100 [57].

4.4.1.2 Training details

We follow the setting in [10] and fine-tune the models for 50 epochs using the Adam optimizer [52] with a batch size of 8 on a single GPU. The initial learning rate is tuned for each dataset, and we adopt a cosine schedule [78] for learning rate decay.

4.4.1.3 Evaluation Metric

In our experiments, we evaluate the transfer accuracy of each dataset to measure the performance of our approach. Additionally, we assess the training efficiency by measuring the training memory consumption, which includes the parameter size and the activation memory storage during training. These metrics provide insights into the effectiveness and efficiency of our proposed method.

Table 19. Comparison with Previous State-of-the-art (Sota) Transfer Learning Methods Using Different Backbone Neural Networks, Where'i-v3' Is Inception-v3; 'n-a' Is Nasnet-a Mobile; 'm2-1.4' Is Mobilenetv2-1.4; 'r-50' Is Resnet-50; 'pm' Is Proxylessnas-mobile. In This Table, We Show Our Improvements in Comparison to Best Existing Transfer Learning Scheme Tinytl.

Method	Net	Train. mem	Reduce Ratio	Flowers	Cars	CUB	Food	Pets	Aircraft	CIFAR10	CIFAR100
FT-Full	I-V3[90]	850MB	1.0×	96.3	91.3	82.8	88.7	-	85.5	-	-
	R-50[55]	802MB	1.1×	97.5	91.7	-	87.8	92.5	86.6	96.8	84.5
	M2-1.4[55]	644MB	1.3×	97.5	91.8	-	87.7	91.0	86.8	96.1	82.5
	N-A[55]	566MB	1.5×	96.8	88.5	-	85.5	89.4	72.8	96.8	83.9
FT-Last	I-V3 [90]	94MB	9.0×	84.5	55.0	-	-	-	45.9	-	-
TinyTL-Random[11]	PM	37MB	22.9×	88.0	82.4	72.9	79.3	84.3	73.6	95.7	81.4
TinyTL[11]	PM	37MB	22.9×	95.5	85.0	77.1	79.7	91.8	75.4	95.9	81.4
Ours	PM	***34MB (↓ 3)***	***25×***	***96.1***	***85.8***	***77.8***	***80.5***	***91.8***	***77.4(↑2%)***	***95.9***	***81.9***
TinyTL[11]	PM@320	65MB	13.1×	96.8	88.8	81.0	82.9	92.9	82.3	96.1	81.5
Ours	PM@320	***61MB (↓ 4)***	***13.9×***	***97.1***	***89.0***	***82.3(↑1.3%)***	***83.3***	***92.5***	***82.4***	***96.6***	***82.3***

Table 20. Combining the Rep-Net with Learnable Binary Mask-based Method for Efficient Inference. 'tinytl-last' Means Only Re-training the Last Classifier on the Tinytl Imagenet Pre-trained Model. The Inference Flops Is Reported on Flowers Dataset.

Method	Inference Flops	Flowers		Cars		CUB		Food		Pets		Aircraft		CIFAR10		CIFAR100	
		Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity
TinyTL-Last [10]	394.6	90.1	-	50.9	-	73.3	-	68.7	-	91.3	-	44.9	-	85.9	-	68.8	-
Ours	346.8	96.1	-	85.8	-	77.8	-	80.5	-	91.8	-	77.4	-	95.9	-	81.9	-
Ours+Bin. Mask	280.9	96.3	19.3	85	30.3	79.2	19.4	83.7	33.8	92.2	5.9	82.5	21.4	96.2	30.3	82.7	31.8

4.4.2 Main Results

4.4.2.1 Comparison with Input Reprogramming Methods

As shown in Table 18, we compare the performance of the proposed Rep-Net with previous adversarial reprogramming methods [23, 54] using ResNet50 [41], MobileNetv2 [100], and MobileNetv3 [48] as backbone pre-trained models. First, we evaluate the performance on the CIFAR10 dataset, which is suitable for adversarial reprogramming methods due to its small image resolution. The results show that Rep-Net achieves a significant improvement in accuracy by more than ∼30% compared to previous methods. Furthermore, compared to the improved adversarial reprogramming method that includes re-training the final classifier, Rep-Net achieves accuracy improvements of 3.6%, 9.9%, and 3.3% on ResNet50, MobileNetv2, and MobileNetv3,

respectively. In addition, Rep-Net outperforms the accuracy of adversarial reprogramming [54] on the CUB and Aircraft datasets by more than ∼10%. In terms of memory, Rep-Net requires reduced training memory overhead for the MobileNet architecture, while slightly higher training memory is needed for the ResNet-50 backbone model.

4.4.2.2 Comparison with State-of-the-art Methods

In Table 19, we compare our method with state-of-the-art transfer learning techniques, including the recent technique TinyTL [10]. However, we want to highlight a crucial distinction between our method and TinyTL. By default, TinyTL uses pre-trained weights on the pre-training dataset to initialize their additional residual modules, giving them an advantage over our Rep-Net, which is trained from scratch with random initialization. Nevertheless, even without the advantage of pre-trained weights, our proposed Rep-Net achieves comparable or better accuracy than TinyTL on all seven transfer datasets. Additionally, Rep-Net requires approximately 3-4 MB less training memory compared to TinyTL. To provide a fair comparison, we also report the results of TinyTL with random initialization, which demonstrate significantly inferior performance compared to Rep-Net.

4.5 Ablation Study and Discussion

4.5.1 Does Rep-Net Transfer Better by Using Better ImageNet Models?

In [55], it is shown that there is a strong correlation between the pre-trained ImageNet accuracy and its corresponding transfer learning performance. This finding

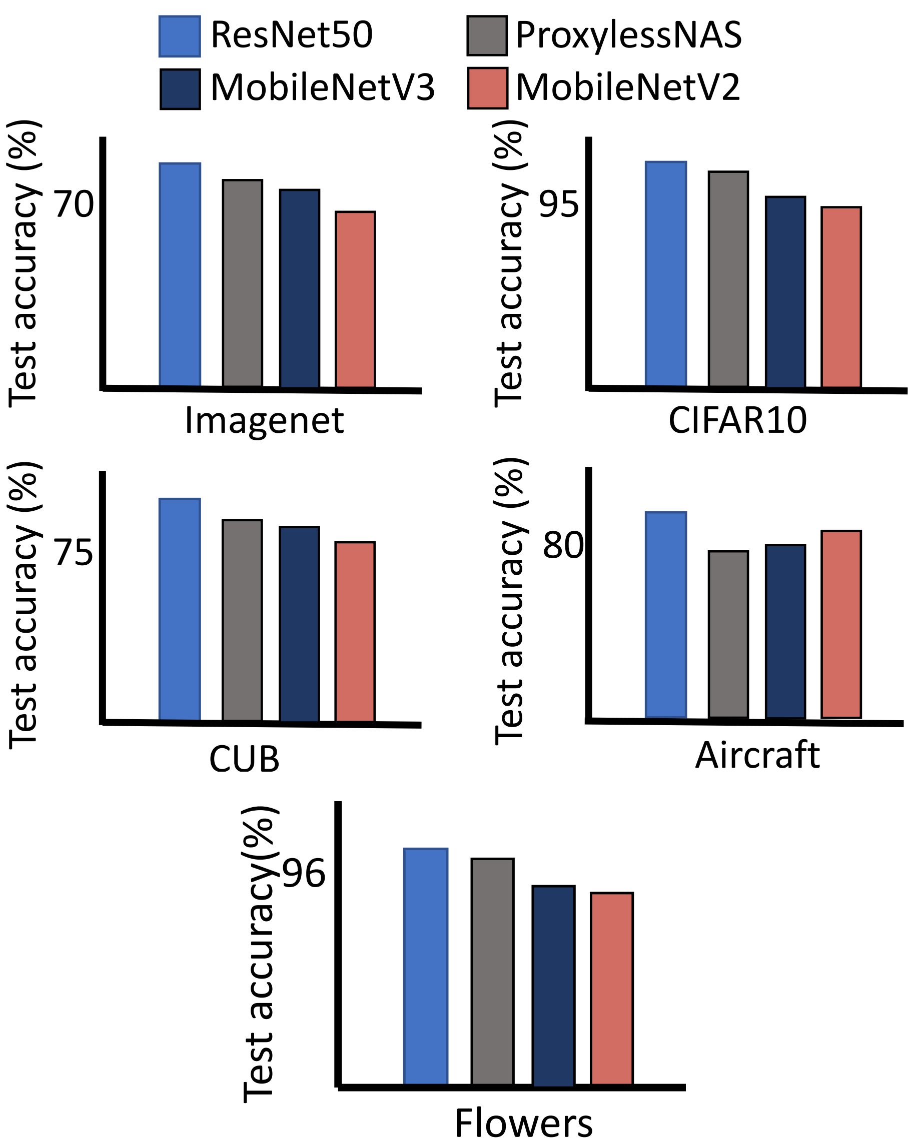


Figure 20. The Accuracy Comparison Between Pre-trained Imagenet and Transfer to CIFAR10, CUB, Aircraft and Flowers on Four Different Models.

prompted us to investigate if our proposed Rep-Net follows the same phenomenon. The results are summarized in Figure 20, which shows that Rep-Net also exhibits a similar trend. The pre-trained ImageNet model accuracy is predictive of the fine-tuning performance order on datasets such as Flowers, CUB, and CIFAR10. However,

ImageNet pretraining does not necessarily improve accuracy on the Aircraft dataset, as its data distribution differs significantly from that of ImageNet.

Interestingly, our proposed Rep-Net outperforms TinyTL [11] by a substantial 2% margin in terms of accuracy on the Aircraft dataset, as shown in Table 19. This highlights the effectiveness of Rep-Net in improving transfer learning performance even in challenging domains that are dissimilar to the ImageNet dataset.

4.5.2 Can Rep-Net Combine with Other Transfer Learning Methods for Efficient Inference?

In addition to reducing training memory, another advantage of Rep-Net is its compatibility with inference-efficient methods. In this subsection, we explore the combination of Rep-Net with learnable binary masks [83, 119] applied to the fixed main branch model. This combination aims to reduce the computational cost during inference. The results in Table 20 show that our method, referred to as "Ours+Bin. Mask," achieves improved accuracy while achieving an average sparsity of approximately 25% across all datasets.

In summary, Rep-Net can be incorporated into existing transfer learning schemes that involve modifying or training additional components (e.g., masks) of the backbone model. This combination not only reduces training memory but also enhances inference efficiency.

4.6 Summary

In this chapter, we have introduced a novel approach to transfer learning called feature reprogramming. Our proposed Rep-Net architecture enables on-device transfer learning by directly learning to reprogram the intermediate features of a pre-trained model. Through dedicated activation connectors, we facilitate the exchange of features between the backbone model and the Rep-Net model. Extensive experiments have demonstrated the effectiveness and memory efficiency of Rep-Net in the context of transfer learning. Our work represents a new perspective and a valuable contribution to the field of on-device transfer learning.

Chapter 5

EFFICIENT CONTINUAL LEARNING VIA TASK-CORRELATED PROGRESSIVE LAYER FREEZING

In recent years, self-supervised learning (SSL) has emerged as a powerful technique for learning visual representations from unlabeled data. SSL involves training a model to predict certain properties of the data without relying on explicit labels. This approach has shown great promise in various computer vision tasks.

Continual learning (CL) is another important area of research where the goal is to sequentially learn multiple tasks without forgetting previously learned ones. Traditional CL methods face challenges in dealing with catastrophic forgetting, where the model loses knowledge of previous tasks when learning new ones. This limitation has motivated researchers to explore SSL in the context of CL, giving rise to self-supervised continual learning (SSCL).

In SSCL, SSL is employed to learn representations that are more informative and resistant to catastrophic forgetting. This leads to better performance compared to supervised continual learning (SCL) methods. However, the training complexity of SSCL can be high due to the computational cost of SSL.

In this chapter, we focus on analyzing the task correlations within the SSCL setup. We make an interesting observation that the intermediate features of the SSL-learned background model exhibit high correlations across different tasks. Leveraging this observation, we propose a novel SSCL method called layer-wise freezing. The idea behind layer-wise freezing is to progressively freeze a portion of the network layers based on their correlation ratios for each task. This means that as the model learns

new tasks, we selectively freeze certain layers that are less correlated with the current task, while allowing other layers to continue learning. By doing so, we aim to improve training computation efficiency and memory efficiency while still benefiting from the SSL-learned representations. In the following sections, we will delve into the details of the layer-wise freezing method and present experimental results to demonstrate its effectiveness in SSCL. Our approach represents a new direction in leveraging SSL for continual learning while addressing the computational challenges associated with it.

5.1 Related Works and Background

5.1.1 Self-supervised Learning

Self-supervised learning has gained significant attention in recent years as a promising approach to learning visual representations without the need for expensive data labeling. Various state-of-the-art methods, such as Momentum Contrast [42], SimCLR [16], Bootstrap Your Own Latent (BYOL)[34], Barlow Twins[129], and SwAV [12], have demonstrated that self-supervised learning can achieve comparable or even superior performance to supervised representation learning.

The common strategy employed by these methods is to learn representations that are invariant under different data augmentations. This is typically achieved by maximizing the similarity between augmented views of the same image and minimizing the similarity with views of other images. Contrastive loss optimization is commonly used to achieve this objective.

One limitation of these approaches is that they often require large-sized batches and a sufficient number of negative samples. Large batch sizes are used to increase the

diversity of negative samples, which can improve the quality of learned representations. However, using large batch sizes may not be feasible in certain computational settings or when working with limited resources. Additionally, the requirement for negative samples can also increase the computational cost and memory requirements during training. Overcoming these limitations and making self-supervised learning more accessible and efficient is an active area of research. Recent advancements have explored alternative methods such as small-batch self-supervised learning [17], which aims to achieve comparable performance with smaller batch sizes. Specifically, SimSiam [17] addresses this issue by utilizing the stop-gradient technique to prevent the collapsing of Siamese networks. The Siamese network consists of an encoder network f and a prediction MLP h, where the encoder includes a backbone model (e.g., ResNet [41]) and a projection MLP. Given two randomly augmented views of x_1 and x_2 from an input image x, Simsiam aims to minimize the negative cosine similarity between the predictor output p_1 ($p_1 = f(h(x_1))$) and the projector output z_2 ($z_2 = f(x_2)$) with a symmetrized loss as:

$$L_{SSL} = \frac{1}{2}D(p_1, \text{stopgrad}(z_2)) + \frac{1}{2}D(p_2, \text{stopgrad}(z_1)) \tag{5.1}$$

where D is a negative cosine similarity function. Given the distorted versions of an instance, BarlowTwin [129] minimizes the redundancy between their embedding vector components while conserving the maximum information. This can be achieved by making the cross-correlation matrix, computed between the outputs of two identical networks, closer to the identity matrix, through the minimization of the following loss:

$$L_{SSL} = \sum_{i}(1 - C_{ii})^2 + \lambda \sum_{i} \sum_{j \neq i} C_{ij}. \tag{5.2}$$

Here λ is a positive constant scaling factor and C is the cross-correlation matrix computed between the outputs of the two identical networks along the batch dimension.

Since SimSiam and BarlowTwim have no requirements for large batch size and negative samples, in this work, we adopt these two works as base learning methods for self-supervised continual learning.

5.1.2 Continual Learning

Numerous continual learning methods have been developed in the context of supervised learning, which can generally be categorized into three main groups.

Regularization-based methods (e.g., [2, 61, 53]) aim to preserve the knowledge of previous tasks by incorporating an additional regularization term into the loss function. This regularization term helps constrain the weight updates during the learning of new tasks. For instance, Elastic Weight Consolidation (EWC) [53] determines the importance of weights using the Fisher Information matrix and imposes regularization on these important weights.

Structure-based methods (e.g., [101, 118]) focus on adapting different model parameters or architectures over the course of sequential tasks. These methods leverage task-specific adjustments to improve performance on each task while maintaining knowledge from previous tasks.

Memory-based methods can be further categorized into two subgroups: memory-replay methods and orthogonal-projection based methods. Memory-replay methods (e.g., [97, 35, 14]) store and replay data from previous tasks during the learning of new tasks. This helps the model retain and utilize knowledge gained from past experiences. On the other hand, orthogonal-projection based methods (e.g., [130, 26, 99, 67, 65]) update the model for each new task in a direction orthogonal to the subspace spanned

by the inputs of previous tasks. This approach minimizes interference between tasks and facilitates efficient learning of new tasks.

In recent years, several works have emerged to address the challenge of self-supervised continual learning, where the goal is to learn representations in a continual learning setting without the need for explicit task labels. These approaches have shown promising results in mitigating catastrophic forgetting and learning more generalized representations compared to supervised continual learning methods.

For instance, Rao et al. [95] proposed a method that learns task-specific representations on shared parameters. However, this approach is limited to simple low-resolution tasks and does not scale well to standard continual learning benchmark datasets. On the other hand, CaSSLe [27] and PFR [33] introduce a temporal projection module that ensures the newly learned feature space preserves information from previous tasks. These methods aim to maintain the representational fidelity across tasks to alleviate catastrophic forgetting.

Another approach, LUMP [80], leverages the Mixup technique [132] to interpolate data between the current task and instances from previous tasks:

$$\tilde{x}_{t,i} = \lambda \cdot x_{t,i} + (1 - \lambda) \cdot x_{M,l}, \tag{5.3}$$

where $x_{M,l}$ denotes the old task data selected using uniform sampling from replay buffer M and λ is randomly sampled from a *Beta* distribution. However, these works directly combine the existing self-supervised with continual learning techniques (e.g, knowledge distillation, mixup, memory replay, etc,.) that still suffer from large training costs, and the forgetting issue remains as well.

5.1.3 Layer Freezing

There have been several works that aim to accelerate the training of deep neural networks for a single task by utilizing layer freezing techniques [75, 40, 107, 1, 126]. These approaches are based on the observation that earlier layers in the network tend to extract more general features of the raw data, while deeper layers capture more task-specific and complex features. For instance, Liu et al.[75] propose an automatic layer freezing method that determines which layers to freeze based on the parameter gradients. Wang et al.[107] utilize knowledge distillation to guide the layer freezing schedule. Yuan et al. [126] apply layer freezing in the context of sparse training. However, these methods focus solely on a single task and typically follow a fixed order of progressively freezing layers in descending layer index. In contrast to these previous works, we argue that layer freezing in self-supervised continual learning (SSCL) needs to consider task correlations.

5.2 Efficient Self-supervised Continual Learning

5.2.1 Problem Formulation

In supervised continual learning, a model learns continuously from a sequential data stream, where new tasks (i.e., classification tasks with new classes) are introduced over time. Formally, we have a sequence of tasks $1, 2, ..., T$, where the task at time t is associated with training data $D_t = \boldsymbol{x}_{t,i}, \boldsymbol{y}_{t,i}{}_{i=1}^{N_t}$. Each task t can consist of a series of classes. We use the function $f(\cdot)$ to represent the feature extractor operation and $h(\cdot)$ to represent the classifier model. The main objective is to optimize the parameter $\boldsymbol{w}$

of both the feature extractor and the classifier:

$$\min_{\boldsymbol{w}_f, \boldsymbol{w}_h} \sum_{t=1}^{T} \sum_{i=1}^{N_t} \mathcal{L}_t(h(f(\boldsymbol{x}_{t,i})), \boldsymbol{y}_{t,i}) \tag{5.4}$$

where $\mathcal{L}_t(\cdot)$ is cross entropy loss function in general.

In contrast, self-supervised continual learning does not rely on labeled data during training. The goal is to learn a general representation that remains invariant under different augmentations for all tasks. Mathematically, this objective can be formulated as follows:

$$\min_{\boldsymbol{w}_f} \sum_{t=1}^{T} \sum_{i=1}^{N_t} \mathcal{L}_t(f(\boldsymbol{x}_{t,i}^1, \boldsymbol{x}_{t,i}^2)) \tag{5.5}$$

In self-supervised continual learning, the augmented images $\boldsymbol{x}_{t,i}^1$ and $\boldsymbol{x}_{t,i}^2$ are generated from the original image $\boldsymbol{x}_{t,i}$. The choice of the feature extractor and the loss function depends on the specific self-supervised learning method being used, such as SimSiam and BarlowTwin. Once the feature extractor is trained on all tasks, we can evaluate its performance using a K-nearest neighbor (KNN) classifier or linear classification, as suggested by Madaan et al. (2021). In this chapter, we focus on self-supervised continual learning in the context of task-incremental learning, where the model has a single classifier and task identifiers are provided during inference.

5.2.2 Progressive Task-correlated Layer Freezing

5.2.2.1 Overview

In our approach, the training of the model for the first task follows the standard self-supervised learning method without any modifications [95, 27, 33, 80]. However, for the subsequent tasks that are learned sequentially, we employ a memory replay-based method inspired by [8]. Specifically, we uniformly store the data of each task in

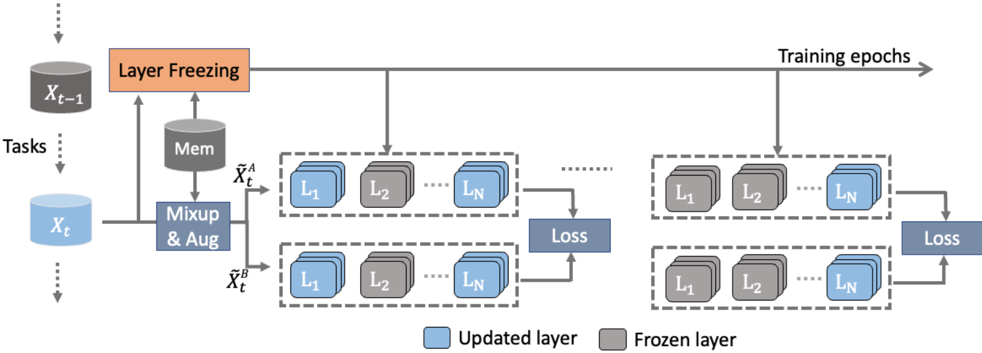


Figure 21. The Overview of Our Proposed Method Which Progressively Freezes Partial Layers During the Whole Training Process for Each Task.

a fixed-size buffer (e.g., 256) and utilize the mixup technique from LUMP [80]. To improve training efficiency, memory efficiency, and mitigate catastrophic forgetting, we introduce the concept of *progressive freezing*. During the training process for each task, we progressively freeze certain layers of the model based on task correlations. The specific layers to be frozen are determined by considering the inter-task correlations. In the following sections, we will provide a detailed explanation of our progressive freezing method and how it is employed in self-supervised continual learning.

5.2.2.2 Layer Freezing via Task Correlation

The training efficiency of self-supervised continual learning (SSCL) is a critical concern due to the high training cost associated with self-supervised learning (SSL) models, especially when new tasks arrive continuously. Improving the training efficiency of SSCL is essential for practical applications and the advancement of SSCL methods. Previous works have demonstrated that the representations learned through SSCL are more general and robust to catastrophic forgetting compared to supervised continual learning (SCL).

A key observation is that the intermediate features learned for each layer between the current task and prior tasks in SSCL are highly correlated. This suggests that if a new task has strong similarities with old tasks in certain layers, freezing those layers and not updating them during training may not significantly affect the learning performance on the new task. Based on this insight, we pose the question: "Can we leverage the generality of the learned representations from SSL and freeze the highly correlated layers during training for each task to improve training efficiency?"

By selectively freezing highly correlated layers, we can potentially reduce the computational and memory costs associated with updating those layers during training. This approach takes advantage of the transferability of the representations learned through SSL, allowing us to focus computational resources on updating the task-specific layers that contribute the most to the new task. This strategy aims to improve the training efficiency of SSCL while preserving the benefits of SSL in terms of generalization and resistance to catastrophic forgetting.

To answer this question, motivated by prior gradient orthogonal-projection based methods [67, 65] on SCL, we first investigate the correlation of tasks according to gradient projection. Specifically, to formally characterize the correlation between the current task and prior tasks, we **define the task correlation ratio** in layer-wise as:

$$r_l = \frac{\|_{S_t^l}(\nabla \mathcal{L}_t(\boldsymbol{w}_{t-1}^l))\|_2}{\|\nabla \mathcal{L}_t(\boldsymbol{w}_{t-1}^l))\|_2} \tag{5.6}$$

where $_{S_t^l}$ denotes the projection on the input subspace S_t^l of prior tasks $(1, 2, ..., t-1)$ on l^{th} layer, and $\boldsymbol{w}_{t-1}^l$ represents the lth layer weight in the model before learning task t. Here $_S(A) = AB(B)'$ for some matrix A and B is the bases for S. Due to the fact that the gradient lies in the span of the input [99], if the task correlation ratio $r_l \in (0, 1)$ has a large value, it implies that the current task t and prior tasks may

have sufficient common bases in l^{th} layer between their input subspaces and hence are strongly correlated. To quantitatively evaluate the task correlation on SSCL, we conduct the experiments on three settings (i.e., Split CIFAR-10, Split CIFAR-100, Split TinyImageNet) by using prior representative work LUMP [80]. As shown in effig:corr, we observe that:

Observations: *1) the variance of the task correlation ratio in SSCL is smaller than the counterparts in SCL; 2) the correlation ratios of SSCL are larger than the counterparts in SCL for most layers; 3) the correlation ratios of SCL consistently follow an ascending order, while the counterparts in SSCL are more varied that are usually higher for top and middle layers.*

The first two observations help to further explain that the learned representations of SSCL are more general than SCL. Moreover, the third observation indicates that following ascending order to freeze layer in supervised learning is a good choice [7, 75, 126] since the correlation ratios of the top layers are always larger than the later ones. However, for the SSCL, layer freezing needs to consider task correlations between tasks.

5.2.3 Subspace Construction via Memory Replay Data.

Prior orthogonal-projection-based methods (e.g., [99, 67]) target task-incremental learning, which calculates and then stores the bases of the input subspace of each prior task individually for orthogonal gradient descent. Such subspace storage consumes large memory costs, especially for the large models of self-supervised learning. For example, ResNet18 requires 116MB on CIFAR-10 dataset. Benefiting from utilizing data replay mechanism, we can construct the subspace of prior tasks on-the-fly instead

of storing the corresponding bases. In practice, before training the current task, we calculate the bases of the subspace for the data in the replay buffer using Singular Value Decomposition (SVD) on the representations. Specifically, given the model $\boldsymbol{w}_{t-1}$ before learning task t, we construct a representation matrix $\boldsymbol{R}_t^l = [\boldsymbol{x}_{r,1}^l, ..., \boldsymbol{x}_{r,n}^l] \in \mathbb{R}^{m\times n}$ with n samples from memory replay buffer, where each $\boldsymbol{x}_{r,i}^l \in \mathbb{R}^m$, is the representation at layer l by forwarding the sample $\boldsymbol{x}_{r,i}$ through the network. Then, we apply SVD to the matrix $\boldsymbol{R}_t^l$, i.e., $\boldsymbol{R}_t^l = \boldsymbol{U}_t^l \boldsymbol{\Sigma}_1^l (\boldsymbol{V}_t^l)'$, where $\boldsymbol{U}_t^l = [\boldsymbol{u}_{t,1}^l, ..., \boldsymbol{u}_{t,m}^l] \in \mathbb{R}^{m\times m}$ is an orthogonal matrix with left singular vector $\boldsymbol{u}_{t,i}^l \in \mathbb{R}^m$, $\boldsymbol{V}_t^l = [\boldsymbol{v}_{t,1}^l, ..., \boldsymbol{v}_{t,n}^l] \in \mathbb{R}^{n\times n}$ is an orthogonal matrix with right singular vector $\boldsymbol{v}_{t,i}^l \in \mathbb{R}^n$, and $\boldsymbol{\Sigma}_t^l \in \boldsymbol{R}^{m\times n}$ is a rectangular diagonal matrix with non-negative singular values $\{\sigma_{t,i}^l\}_{i=1}^{\min\{m,n\}}$ on the diagonal in a descending order. To obtain the bases for subspace S_t^l, we use k_t^l-rank matrix approximation to pick the first k_t^l left singular vectors in $\boldsymbol{U}_t^l$, such that the following condition is satisfied for a threshold $\eta_{th}^l \in (0, 1)$:

$$\|(\boldsymbol{R}_t^l)_{k_t^l}\|_F^2 \geq \epsilon_{th}^l \|\boldsymbol{R}_t^l\|_F^2 \tag{5.7}$$

where $(\boldsymbol{R}_t^l)_{k_t^l} = \sum_{i=1}^{k_t^l} \sigma_{t,i}^l \boldsymbol{u}_{t,i}^l (\boldsymbol{v}_{t,i}^l)'$ is a k_t^l-rank ($k_t^l \leq r$) approximation of the representation matrix $\boldsymbol{R}_t^l$ with rank $r \leq \min\{m, n\}$, and $\|\cdot\|_F$ is the Frobenius norm. Then the bases for subspace S_t^l can be constructed as $\boldsymbol{B}_t^l = [\boldsymbol{u}_{t,1}^l, ..., \boldsymbol{u}_{t,k_t^l}^l]$.

5.2.3.1 Progressive Task-correlated Freezing

Based on the proposed task-correlation metric, we further propose **progressive task-correlated freezing** in SSCL to progressively freeze partial layers with the highest correlation ratios during training for each task, in order to enhance the training computation and memory efficiency. Specifically, define the initial freeze ratio as k_i and final freeze ratio as k_f, which denote the ratio of the number of frozen layers to

the number of layers in the neural network. The total number of training epochs is N and the current training epoch is n. We adopt cosine annealing to progressively increase the freeze ratio in epoch-wise:

$$k_n = k_i + \frac{1}{2}(k_f - k_i)(1 + cos(\frac{n}{N}\pi)) \tag{5.8}$$

where k_n is the freeze ratio for the current epoch. In our experiments, we set the initial and final freeze ratios as 0 and 0.4 for all tasks by default.

Following that, once getting the freeze ratio for the current epoch, we adopt the following strategies to progressively and accumulately freeze the layers: 1) the layers with the highest task-correlation ratio under the current freeze ratio k_n will be frozen; 2) the frozen layers of prior epochs will be unchanged, and we will gradually increase the number of frozen layers according to the freeze ratio difference $(k_n - k_{n-1})$. This can be achieved by using a TopK function according to the layer-wise task correlation to select the layers to freeze:

$$F_n = \{l | r_n^l \in \text{TopK}(R_n, k_n - k_{n-1})\} \tag{5.9}$$

where R_n denotes a set of task correlations across all unfrozen layers in current epoch n and r_n^l is the task-correlation ratio for l^{th} layer as defined in 5.6. By doing so, we could generate a set of indexes of new frozen layers F_n for each task. Importantly,

One practical reason that we choose layer-wise freezing is that layer-wise freezing could enable actual training speedup in GPU by using general deep learning frameworks (e.g, Tensorflow, Pytorch). In addition, we find that if applying the proposed layer-wise weight freezing in supervised continual learning (SCL) setup, it will cause clear accuracy degradation, which also advocates that the representation learned by SSCL is more general and robust.

5.3 Experimental Results

5.3.1 Experimental Setup

Previous research on self-supervised continual learning (SSCL) [95, 27, 33, 80] has demonstrated the superior performance of SSCL methods compared to single-task supervised continual learning (SCL) approaches in class incremental learning. In this study, we conduct a comprehensive comparison with various self-supervised continual learning baselines, encompassing different categories of continual learning methods.

Firstly, we primarily compare our method to the state-of-the-art self-supervised continual learning techniques, namely CassLe [33] and LUMP [80]. It is important to note that we reproduce the reported results of CassLe using the same experimental setup as LUMP to ensure a fair comparison.

Secondly, following the approach in [80], we also present results for several self-supervised variants of SCL methods. Specifically, we evaluate the performance of FINETUNE, which is a vanilla supervised learning method trained on a sequence of tasks without regularization or episodic memory, as well as MULTITASK, which optimizes the model using complete data.

Additionally, we compare our method against previous SCL methods in a self-supervised learning setting. Specifically, we assess the effectiveness of SI [131] for regularization-based CL methods, PNN [98] for architecture-based methods, and DER [8] for memory replay methods, which adapt knowledge distillation through memory replay to align the network logits sampled during the optimization trajectory in continual learning.

Dataset We evaluate the performance of SSCL on various continual learning

benchmarks using the single-head ResNet-18 architecture proposed by He et al.[41]. The benchmarks we utilize include Split CIFAR-10[57], where each task involves two randomly selected classes from the ten available classes. For Split CIFAR-100 [57], each task comprises five random classes chosen from the 100 classes. Split Tiny-ImageNet is a modified version of the ImageNet dataset [19], with each task containing five random classes selected from the 100 classes, and the images are resized to 64×64 pixels. In Split ImageNet-100, each task encompasses 20 randomly chosen classes from the 100 classes. It is important to note that ImageNet-100 is a subset of the ILSVRC2012 dataset, which consists of approximately 130,000 high-resolution images resized to 224x224 pixels.

Experimental setup. We adopt the training and evaluation setup outlined in the work of Madaan et al.[80] for all the SSCL representation learning strategies applied to the Split CIFAR-10, CIFAR-100, and Split Tiny-ImageNet datasets. The learned representations are assessed using the K-nearest neighbors (KNN) classifier[109], with evaluations conducted across three independent runs for robustness. During training, we train all the SSCL methods for 200 epochs and subsequently evaluate their performance using the KNN classifier [109]. For the experiments, we set the memory buffer size to 256, and the models are optimized using the SGD optimizer with a base learning rate of 0.03, employing a batch size of 256. Moreover, in line with the methodology presented in Fini et al. [27] for the ImageNet-100 dataset, we employ the LARS optimizer for model training. For testing, we conduct linear evaluation.

Metrics. Following [80], two metrics are used to evaluate the performance: *Accuracy*, the average final accuracy over all tasks, and *Forgetting*, which measures the forgetting of each task between its maximum accuracy and accuracy at the completion

of training. Accuracy and Forgetting are defined as:

$$Accuracy = \frac{1}{T}\sum\nolimits_{i=1}^{T} A_{T,i} \tag{5.10}$$

$$Forgetting = \frac{1}{T-1}\sum\nolimits_{i=1}^{T-1} \max\nolimits_t(A_{T,i} - A_{i,i}) \tag{5.11}$$

where T is the number of tasks, $A_{T,i}$ is the accuracy of the model on i-th task after learning the T-th task sequentially. Furthermore, we utilize three metrics to measure training efficiency: Training time, we report the training time ratio compared to LUMP baseline which is measured on NVIDIA RTX A4000 GPU; memory, which includes model parameter size, training activation storage, and memory replay buffer size; Flops, which calculate the number of computational operations during backward.

5.3.2 Main Results

As presented in Table 21 and Table 22, we assess the performance of various SSCL methods utilizing the SimSiam [16] and BarlowTwin [129] SSL frameworks on the Split CIFAR-10, Split CIFAR-100, and Split Tiny-ImageNet datasets.

It is worth noting that the training memory cost comprises model parameters, memory replay data, and activations of each layer for backward propagation. When using the SimSiam framework, our method achieves notable reductions in training time, memory usage, and backward FLOPs across the three datasets. Specifically, we achieve a 12%, 14%, and 12% reduction in training time, a 23%, 26%, and 24% reduction in memory, and a 33%, 33%, and 32% reduction in backward FLOPs, respectively. Similarly, when employing the BarlowTwin framework, our method demonstrates significant improvements. We achieve a 13%, 12%, and 12% reduction in training time, a 22%, 21%, and 22% reduction in memory, and a 35%, 34%, and 33% reduction in backward FLOPs across the three datasets, respectively. Importantly,

with respect to the forgetting issue, our method effectively mitigates this problem compared to all prior methods. For instance, when comparing our method to LUMP on the Split CIFAR-100 and Split Tiny-ImageNet datasets using the SimSiam framework, we achieve reductions in forgetting by 1.31%, 1.98%, and 1.21%, respectively, while maintaining similar levels of accuracy.

Table 21. Accuracy and Forgetting of the Learned Representations on Split Cifar-10, Split CIFAR-100, and Split Tiny-imagenet on Resnet-18 Architecture with KNN Classifier. All the Values Are Measured by Computing Mean and Standard Deviation Across Three Trials. Note That, We Use the Layer Freezing Ratio As 0.4 by Default for All Our Results.

	Method	SPLIT CIFAR-10		SPLIT CIFAR-100		SPLIT TINY-IMAGENET	
		Accuracy	Forgetting	Accuracy	Forgetting	Accuracy	Forgetting
Simsiam	Finetune	90.11 (±0.12)	5.43 (±0.08)	75.42 (±0.78)	10.19 (±0.78)	71.07 (±0.20)	9.48 (±0.56)
	PNN [98]	90.93 (±0.22)	-	66.58 (±1.0)	-	62.15 (±1.35)	-
	SI [131]	92.75 (±0.06)	1.81 (±0.21)	80.08 (±1.3)	5.54 (±01.3)	72.34 (±0.42)	8.26 (±0.64)
	DER [8]	91.22 (±0.3)	4.63 (±0.26)	77.27 (±0.30)	9.31 (±0.09)	71.90 (±1.44)	8.36 (±2.06)
	CassLe [33]	**91.04** (±0.24)	2.24 (±0.23)	81.58 (±0.84)	5.02 (±1.12)	75.77 (±1.74)	4.42 (±1.24)
	LUMP [80]	91.00 (±0.40)	2.92 (±0.53)	**82.30** (±1.35)	4.71 (±1.52)	76.66 (±2.39)	3.54 (±1.04)
	Ours	91.03 (±0.44)	**1.61** (±0.23)	82.24 (±1.24)	**2.73**(±1.13)	**76.68** (±2.51)	**2.33**(±1.14)
	Multitask	95.76 (±0.08)	-	86.31 (±0.38)	-	82.89 (±0.49)	-
BarlowTwin	Finetune	87.72 (±0.32)	4.08 (±0.56)	71.97 (±0.54)	9.45 (±1.01)	66.28 (±1.23)	8.89 (±0.66)
	PNN [98]	87.52 (±0.33)	-	57.93 (±2.98)	-	48.70 (±2.59)	-
	SI [131]	**90.21** (±0.08)	2.03 (±0.22)	75.04 (±0.63)	7.43 (±0.67)	56.96 (±1.48)	17.04 (±0.89)
	DER [8]	88.67 (±0.30)	2.41 (±0.26)	73.48 (±0.53)	7.98 (±0.29)	68.56 (±1.47)	7.87 (±0.44)
	Cassle [33]	89.04 (±0.34)	1.89 (±0.14)	77.05 (±0.75)	2.47 (±0.44)	71.76 (±0.65)	2.88 (±0.45)
	LUMP [80]	89.72 (±0.30)	1.13 (±0.18)	80.24 (±1.04)	3.53 (±0.83)	72.17 (±0.89)	2.43 (±1.00)
	Ours	89.73 (±0.41)	**0.92** (±0.23)	**80.54** (±0.88)	**2.24** (±0.84)	**73.56** (±1.02)	**1.74** (±0.62)
	Multitask	95.48 (±0.14)	-	87.16 (±0.52)	-	82.42 (±0.74)	-

Furthermore, we extend our experiments to the more demanding ImageNet-100 dataset. In particular, we apply our proposed method to the BarlowTwin and MoCoV2+ frameworks, following the same experimental setup as CassLe. The results, as presented in Table 23, consistently demonstrate the effectiveness of our method in enhancing training efficiency and mitigating catastrophic forgetting while achieving comparable accuracy levels. By incorporating our proposed method into the BarlowTwin and MoCoV2+ frameworks, we observe significant improvements in training efficiency and alleviation of catastrophic forgetting, all while maintaining

Table 22. Training Time (Measured Time in Nvidia A4000 GPU), Memory Cost, And Computation Flops of the Learned Representations on Split CIFAR-10, Split Cifar-100 and Split Tiny-imagenet.

	Method	SPLIT CIFAR-10			SPLIT CIFAR-100			SPLIT TINY-IMAGENET		
		Time	Memory	FLOPs	Time	Memory	FLOPs	Time	Memory	FLOPs
Simsiam	PNN [98]	1.35x	1.35x	1.35x	1.35x	1.35x	1.35x	1.35x	1.35x	1.35x
	SI [131]	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x
	DER [8]	1x	1x	1x	1x	1x	1x	1x	1x	1x
	LUMP [80]	1x	1x	1x	1x	1x	1x	1x	1x	1x
	CassLe [33]	1.3x	1.3x	1.3x	1.3x	1.3x	1.3x	1.3	1.3x	1.3x
	Ours	**0.88x**	**0.77x**	**0.68x**	**0.86x**	**0.74x**	**0.67x**	**0.88x**	**0.76x**	**0.68x**
BarlowTwin	PNN [98]	1.35x	1.35x	1.35x	1.35x	1.35x	1.35x	1.35x	1.35x	1.35x
	SI [131]	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x	1.2x
	DER [8]	1x	1x	1x	1x	1x	1x	1x	1x	1x
	LUMP [80]	1x	1x	1x	1x	1x	1x	1x	1x	1x
	Cassle [33]	1.3x	1.3x	1.3x	1.3x	1.3x	1.3x	1.3x	1.3x	1.3x
	Ours	**0.87x**	**0.78x**	**0.65x**	**0.88x**	**0.79x**	**0.66x**	**0.88x**	**0.75x**	**0.67x**

similar accuracy to the baseline methods. These findings highlight the robustness and versatility of our approach across different SSL frameworks and challenging datasets like ImageNet-100.

Table 23. Accuracy, Forgetting, Training Time, and Training Memory Cost of the Learned Representations on Imagenet-100 with Linear Evaluation by Using Barlowtwin and Mocov2 Respectively.

	Setting	ImageNet-100			
		Accuracy	Forgeting	Time	Memory
BarlowTwin	CassLe	68.2	1.3	1x	1x
	Ours	**68.5**	**0.7**	**0.88x**	**0.78x**
MoCoV2	CassLe	**68.0**	2.2	1x	1x
	Ours	67.9	**1.4**	**0.89x**	**0.79x**

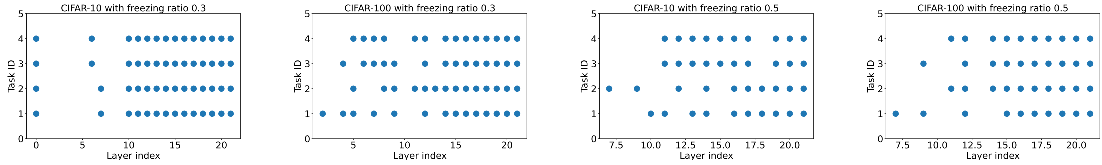


Figure 22. The Final Selection of Updated Layer for Each Task. The Freezing Ratios Are 0.3 And 0.5 Respectively. Note That, Each Blue Point Means the Index of the Updated Layer. For Split CIFAR-100 20 Tasks Setup, We Show the First Five Tasks for Simplification..

Table 24. The Ablation Study on the Proposed Method in Comparison to Layer Freezing in Ascending Layer Index (I.E., "Top Layer") Order on Both Split Cifar-10 and Cifar-100 Datasets by Using Barlowtwin as Backbone Method.

Setting	Split CIFAR-10		Split CIFAR-100	
	Forgetting	Accuracy	Forgetting	Accuracy
Top layers.	0.96	89.24	2.57	78.87
Ours	**0.92**	**90.03**	**2.24**	**80.54**

5.4 Ablation Study and Discussion

5.4.1 Task-correlated Layer Freezing vs. Ascending Order Layer Freezing

In order to assess the efficacy of the proposed progressive layer freezing technique using task correlation, we compare it to the commonly employed ascending order layer freezing approach typically used in supervised learning settings to enhance training efficiency for a single task. To ensure a fair comparison, we also utilize the same cosine annealing strategy to progressively freeze layers, employing an equivalent freezing ratio of 0.4.

As presented in Table 24, our proposed method consistently achieves higher accuracy while maintaining similar levels of forgetting when compared to Naive Selection. These results underscore the importance of considering task correlation in the context of SSCL. The findings highlight that task correlation between different tasks

should be taken into account when designing SSCL methods, as it can significantly impact the overall performance and ability to mitigate forgetting.

5.4.2 Self-supervised Layer Decision

To analyze the layer freezing decision, we experiment with a freezing ratio of 0.4 for each task on the Split CIFAR-10 and Split CIFAR-100 datasets. In Figure 22, we make the following observations:

Inter-tasks: The freezing decisions for different tasks exhibit a high degree of similarity. This indicates that the layers selected for freezing in the first task remain unchanged throughout the subsequent tasks. This finding suggests that the learned representations through SSCL are general and robust, as the frozen layers consistently capture task-agnostic information.

Intra-task: Interestingly, we observe that the first layer and a significant number of last layers are updated during training. We propose a conjecture to explain this observation. The last layers are responsible for learning high-level features that are sensitive to the specific input. Therefore, it is expected that they need to be updated to adapt to the new task. Additionally, although the first layer learns general low-level features, SSCL applies strong augmentation techniques (e.g., color jittering, grayscale conversion, Gaussian blurring, solarization) to the input. Consequently, it is reasonable for the first layer to be updated as well to accommodate the transformed input.

These findings shed light on the layer freezing decision in SSCL. They highlight the importance of considering both inter-task and intra-task dynamics, as well as the role of different layers in learning and adapting representations for continual learning tasks.

5.5 Summary

In this chapter, we first explore the task correlation in SSCL and observe a strong correlation among intermediate features across different tasks. Building upon this observation, we propose a novel approach called progressive task-correlated layer freezing. This method involves gradually freezing a subset of layers with the highest correlation ratios for each task.

Through extensive experiments conducted on multiple datasets, our findings demonstrate the effectiveness of our proposed method. Specifically, our approach significantly enhances training computation and memory efficiency while effectively mitigating catastrophic forgetting. These results establish our method as a state-of-the-art (SoTA) solution in the field of SSCL.

By leveraging the task correlation to guide the layer freezing process, we achieve notable improvements over existing SSCL methods. Our approach offers a promising avenue for improving both the efficiency and robustness of SSCL, contributing to advancements in continual learning research.

Chapter 6

CONCLUSION AND OUTLOOK

This dissertation has discussed our research on enabling deep learning at edge, from efficient and dynamic inference to on-device learning. From the inference perspective, deep learning models suffer from large memory and computation cost which is inefficient or even impractical to be deployed on edge devices. To tackle the challenge, a hardware-friendly model compression method is proposed, which combines weight ternerization and structured pruning to maximize inference efficiency. Moreover, the conventional model compression methods usually lead to a fixed/static compressed model which can not meet the requirement of dynamic hardware resources and environment, such as varied power budget, dynamic workload, different resource allocation, and so on. To address this issue, we further explore dynamic inference which consists of multiple sub-networks with different model sizes to enable the run-time adjustment of model size, computation, and latency. From learning perspective, to achieve on-device learning, we first find that the training process is memory-intensive and the intermediate activation memory during training is the bottleneck. Following this rule, a memory-efficient transfer learning method (i.e., Rep-net) is proposed which is a feature reprograming side-network that significantly reduces the training memory cost while improving knowledge transfer capacity without forgetting prior learned knowledge. Furthermore, we explore efficient continual learning method via progressive task-correlated layer freezing which aims to continually learn multiple tasks without forgetting prior learned knowledge.

In the future, I plan to further explore full-stack efficient AI in the following

three aspects: 1) the efficient and continual on-device learning algorithm, model, and system; 2) label- and data-efficient learning algorithm; 3) efficient and trustworthy AI in Cyber-Physical System.

REFERENCES

[1] Gustavo Aguilar et al. "Knowledge distillation from internal representations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05. 2020, pp. 7350–7357.

[2] Rahaf Aljundi et al. "Memory aware synapses: Learning what (not) to forget". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 139–154.

[3] Jose M Alvarez and Mathieu Salzmann. "Learning the number of neurons in deep networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2270–2278.

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation". In: *arXiv preprint arXiv:1308.3432* (2013).

[5] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. "Food-101–mining discriminative components with random forests". In: *European conference on computer vision*. Springer. 2014, pp. 446–461.

[6] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *International Conference Computational Statistics*. 2010, pp. 177–186.

[7] Andrew Brock et al. "Freezeout: Accelerate training by progressively freezing layers". In: *arXiv preprint arXiv:1706.04983* (2017).

[8] Pietro Buzzega et al. "Dark experience for general continual learning: a strong, simple baseline". In: *Advances in neural information processing systems* 33 (2020), pp. 15920–15930.

[9] Han Cai et al. "m". In: *International Conference on Learning Representations*. 2019.

[10] Han Cai et al. "Tiny Transfer Learning: Towards Memory-Efficient On-Device Learning". In: *arXiv preprint arXiv:2007.11622* (2020).

[11] Han Cai et al. "Tinytl: Reduce memory, not parameters for efficient on-device learning". In: *arXiv preprint arXiv:2007.11622* (2020).

[12] Mathilde Caron et al. "Emerging properties in self-supervised vision transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 9650–9660.

[13] Ken Chatfield et al. "Return of the devil in the details: Delving deep into convolutional nets". In: *arXiv preprint arXiv:1405.3531* (2014).

[14] Arslan Chaudhry et al. "Efficient lifelong learning with a-gem". In: *arXiv preprint arXiv:1812.00420* (2018).

[15] Tianqi Chen et al. "Training deep nets with sublinear memory cost". In: *arXiv preprint arXiv:1604.06174* (2016).

[16] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[17] Xinlei Chen and Kaiming He. "Exploring simple siamese representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15750–15758.

[18] Yin Cui et al. "Large scale fine-grained categorization and domain-specific transfer learning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4109–4118.

[19] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[20] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[21] Emily L Denton et al. "Exploiting linear structure within convolutional networks for efficient evaluation". In: *NIPS*. 2014, pp. 1269–1277.

[22] Tim Dettmers and Luke Zettlemoyer. "Sparse networks from scratch: Faster training without losing performance". In: *arXiv preprint arXiv:1907.04840* (2019).

[23] Gamaleldin F Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. "Adversarial reprogramming of neural networks". In: *arXiv preprint arXiv:1806.11146* (2018).

[24] R David Evans, Lufei Liu, and Tor M Aamodt. "Jpeg-act: accelerating deep learning via transform-based lossy compression". In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2020, pp. 860–873.

[25] Utku Evci et al. "Rigging the lottery: Making all tickets winners". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 2943–2952.

[26] Mehrdad Farajtabar et al. "Orthogonal gradient descent for continual learning". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 3762–3773.

[27] Enrico Fini et al. "Self-supervised models are continual learners". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 9621–9630.

[28] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations*. 2018.

[29] Jonathan Frankle, David J Schwab, and Ari S Morcos. "Training batchnorm and only batchnorm: On the expressive power of random features in cnns". In: *arXiv preprint arXiv:2003.00152* (2020).

[30] Rong Ge et al. "Escaping from saddle points—online stochastic gradient for tensor decomposition". In: *Conference on Learning Theory*. 2015.

[31] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[32] Aidan N Gomez et al. "The reversible residual network: Backpropagation without storing activations". In: *Advances in neural information processing systems*. 2017, pp. 2214–2224.

[33] Alex Gomez-Villa et al. "Continually Learning Self-Supervised Representations with Projected Functional Regularization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 3867–3877.

[34] Jean-Bastien Grill et al. "Bootstrap your own latent-a new approach to self-supervised learning". In: *Advances in neural information processing systems* 33 (2020), pp. 21271–21284.

[35] Yunhui Guo et al. "Improved schemes for episodic memory-based lifelong learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1023–1035.

[36] Yunhui Guo et al. "Spottune: transfer learning through adaptive fine-tuning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4805–4814.

[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 (ICLR)* (2016).

[38] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[39] Babak Hassibi and David G Stork. "Second order derivatives for network pruning: Optimal brain surgeon". In: *Advances in neural information processing systems*. 1993, pp. 164–171.

[40] Chaoyang He et al. "Pipetransformer: Automated elastic pipelining for distributed training of transformers". In: *arXiv preprint arXiv:2102.03161* (2021).

[41] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[42] Kaiming He et al. "Momentum contrast for unsupervised visual representation learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738.

[43] Yang He et al. "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.

[44] Yihui He, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1389–1397.

[45] Zhezhi He, Boqing Gong, and Deliang Fan. "Optimize deep convolutional neural network with ternarized weights and high accuracy". In: *2019 IEEE*

Winter Conference on Applications of Computer Vision (WACV). IEEE. 2019, pp. 913–921.

[46] Geoffrey Hinton et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[47] Torsten Hoefler et al. "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks". In: *Journal of Machine Learning Research* 22.241 (2021), pp. 1–124.

[48] Andrew Howard et al. "Searching for mobilenetv3". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1314–1324.

[49] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[50] Itay Hubara et al. "Quantized neural networks: Training neural networks with low precision weights and activations". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6869–6898.

[51] Siddhant Jayakumar et al. "Top-KAST: Top-K always sparse training". In: *Advances in Neural Information Processing Systems*. 2020.

[52] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[53] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: vol. 114. 13. National Acad Sciences, 2017, pp. 3521–3526.

[54] Eliska Kloberdanz, Jin Tian, and Wei Le. "An Improved (Adversarial) Reprogramming Technique for Neural Networks". In: *International Conference on Artificial Neural Networks*. Springer. 2021, pp. 3–15.

[55] Simon Kornblith, Jonathon Shlens, and Quoc V Le. "Do better imagenet models transfer better?" In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2661–2671.

[56] Jonathan Krause et al. "3d object representations for fine-grained categorization". In: *Proceedings of the IEEE international conference on computer vision workshops*. 2013, pp. 554–561.

[57] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[58] Vadim Lebedev and Victor Lempitsky. "Fast convnets using group-wise brain damage". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2554–2564.

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

[60] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. "SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY". In: *International Conference on Learning Representations*. 2018.

[61] Sang-Woo Lee et al. "Overcoming catastrophic forgetting by incremental moment matching". In: *Advances in neural information processing systems* 30 (2017).

[62] Cong Leng et al. "Extremely low bit neural network: Squeeze the last bit out with admm". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[63] Fengfu Li, Bo Zhang, and Bin Liu. "Ternary weight networks". In: *arXiv preprint arXiv:1605.04711* (2016).

[64] Hao Li et al. "Pruning filters for efficient convnets". In: *arXiv preprint arXiv:1608.08710* (2016).

[65] Sen Lin et al. "Beyond not-forgetting: Continual learning with backward knowledge transfer". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 16165–16177.

[66] Sen Lin et al. "TRGP: Trust Region Gradient Projection for Continual Learning". In: *International Conference on Learning Representations*. 2021.

[67] Sen Lin et al. "TRGP: Trust Region Gradient Projection for Continual Learning". In: *arXiv preprint arXiv:2202.02931* (2022).

[68] Xiaofan Lin, Cong Zhao, and Wei Pan. "Towards accurate binary convolutional neural network". In: *Advances in Neural Information Processing Systems*. 2017, pp. 345–353.

[69] Baoyuan Liu et al. "Sparse convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 806–814.

[70] Chenxi Liu et al. “Progressive neural architecture search”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 19–34.

[71] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “Darts: Differentiable architecture search”. In: *arXiv preprint arXiv:1806.09055* (2018).

[72] Liu Liu et al. “Dynamic sparse graph for efficient deep learning”. In: *arXiv preprint arXiv:1810.00859* (2018).

[73] Shiwei Liu et al. “Deep ensembling with no overhead for either training or testing: The all-round blessings of dynamic sparsity”. In: *arXiv preprint arXiv:2106.14568* (2021).

[74] Shiwei Liu et al. “Sparse training via boosting pruning plasticity with neuroregeneration”. In: *Advances in Neural Information Processing Systems*. 2021.

[75] Yuhan Liu, Saurabh Agarwal, and Shivaram Venkataraman. “Autofreeze: Automatically freezing model blocks to accelerate fine-tuning”. In: *arXiv preprint arXiv:2102.01386* (2021).

[76] Zechun Liu et al. “Metapruning: Meta learning for automatic neural network channel pruning”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3296–3305.

[77] Zhuang Liu et al. “Learning efficient convolutional networks through network slimming”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2736–2744.

[78] Ilya Loshchilov and Frank Hutter. “Sgdr: Stochastic gradient descent with warm restarts”. In: *arXiv preprint arXiv:1608.03983* (2016).

[79] Christos Louizos, Max Welling, and Diederik P Kingma. “Learning Sparse Neural Networks through L_0 Regularization”. In: *arXiv preprint arXiv:1712.01312* (2017).

[80] Divyam Madaan et al. “Representational continuity for unsupervised continual learning”. In: *International Conference on Learning Representations*. 2021.

[81] Aleksander Madry et al. “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rJzIBfZAb.

[82] Subhransu Maji et al. “Fine-grained visual classification of aircraft”. In: *arXiv preprint arXiv:1306.5151* (2013).

[83] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. “Piggyback: Adapting a single network to multiple tasks by learning to mask weights”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 67–82.

[84] Massimiliano Mancini et al. “Adding new tasks to a single network with weight transformations using binary masks”. In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 0–0.

[85] Jian Meng et al. “Contrastive dual gating: Learning sparse features with contrastive learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12257–12265.

[86] Decebal Constantin Mocanu et al. “Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science”. In: *Nature Communications* 9.1 (2018), pp. 1–12.

[87] Amirkeivan Mohtashami, Martin Jaggi, and Sebastian U Stich. “Masked Training of Neural Networks with Partial Gradients”. In: *Proceedings of Machine Learning Research* (2022).

[88] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. “Variational dropout sparsifies deep neural networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2498–2507.

[89] Hesham Mostafa and Xin Wang. “Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization”. In: *International Conference on Machine Learning*. 2019.

[90] Pramod Kaushik Mudrakarta et al. “K for the price of 1: Parameter-efficient multi-task and transfer learning”. In: *arXiv preprint arXiv:1810.10703* (2018).

[91] Alex Nichol, Joshua Achiam, and John Schulman. “On first-order meta-learning algorithms”. In: *arXiv preprint arXiv:1803.02999* (2018).

[92] Maria-Elena Nilsback and Andrew Zisserman. “Automated flower classification over a large number of classes”. In: *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE. 2008, pp. 722–729.

[93] Omkar M Parkhi et al. “Cats and dogs”. In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 3498–3505.

[94] Alexandra Peste et al. "AC/DC: Alternating compressed/decompressed training of deep neural networks". In: *Advances in Neural Information Processing Systems*. 2021.

[95] Dushyant Rao et al. "Continual unsupervised representation learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[96] Mohammad Rastegari et al. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[97] Matthew Riemer et al. "Learning to learn without forgetting by maximizing transfer and minimizing interference". In: *arXiv preprint arXiv:1810.11910* (2018).

[98] Andrei A Rusu et al. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[99] Gobinda Saha, Isha Garg, and Kaushik Roy. "Gradient projection memory for continual learning". In: *arXiv preprint arXiv:2103.09762* (2021).

[100] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520.

[101] Joan Serra et al. "Overcoming catastrophic forgetting with hard attention to the task". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4548–4557.

[102] Suraj Srinivas and R Venkatesh Babu. "Data-free parameter pruning for deep neural networks". In: *arXiv preprint arXiv:1507.06149* (2015).

[103] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. "Resiliency of deep neural networks under quantization". In: *arXiv preprint arXiv:1511.06488* (2015).

[104] Yun-Yun Tsai, Pin-Yu Chen, and Tsung-Yi Ho. "Transfer learning without knowing: Reprogramming black-box machine learning models with scarce data and limited resources". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9614–9624.

[105] Catherine Wah et al. "The caltech-ucsd birds-200-2011 dataset". In: (2011).

[106] Chaoqi Wang, Guodong Zhang, and Roger Grosse. “Picking Winning Tickets Before Training by Preserving Gradient Flow”. In: *International Conference on Learning Representations*. 2019.

[107] Yiding Wang et al. “Efficient dnn training with knowledge-guided layer freezing”. In: *arXiv preprint arXiv:2201.06227* (2022).

[108] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 2074–2082.

[109] Zhirong Wu et al. “Unsupervised feature learning via non-parametric instance discrimination”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3733–3742.

[110] Li Yang, Zhezhi He, and Deliang Fan. “Harmonious coexistence of structured weight pruning and ternarization for deep neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 6623–6630.

[111] Li Yang, Adnan Siraj Rakin, and Deliang Fan. “*DA*3: Deep Additive Attention Adaption for Memory-Efficient On-Device Multi-Domain Learning”. In: *arXiv preprint arXiv:2012.01362* (2020).

[112] Li Yang, Adnan Siraj Rakin, and Deliang Fan. “DA3: Dynamic Additive Attention Adaption for Memory-Efficient On-Device Multi-Domain Learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 2619–2627.

[113] Li Yang, Adnan Siraj Rakin, and Deliang Fan. “Rep-net: Efficient on-device learning via feature reprogramming”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12277–12286.

[114] Li Yang et al. “A Progressive Sub-Network Searching Framework for Dynamic Inference”. In: *arXiv preprint arXiv:2009.05681* (2020).

[115] Li Yang et al. “A progressive subnetwork searching framework for dynamic inference”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[116] Li Yang et al. “Efficient Self-supervised Continual Learning with Progressive Task-correlated Layer Freezing”. In: *arXiv preprint arXiv:2303.07477* (2023).

[117] Li Yang et al. "Get More at Once: Alternating Sparse Training with Gradient Correction". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 30840–30850.

[118] Li Yang et al. "GROWN: GRow Only When Necessary for Continual Learning". In: *arXiv preprint arXiv:2110.00908* (2021).

[119] Li Yang et al. "KSM: Fast Multiple Task Adaption via Kernel-wise Soft Mask Learning". In: *arXiv preprint arXiv:2009.05668* (2020).

[120] Li Yang et al. "Ksm: Fast multiple task adaption via kernel-wise soft mask learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13845–13853.

[121] Li Yang et al. "Non-uniform dnn structured subnets sampling for dynamic inference". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6.

[122] Jiahui Yu and Thomas Huang. "Universally slimmable networks and improved training techniques". In: *arXiv preprint arXiv:1903.05134* (2019).

[123] Jiahui Yu et al. "BigNAS: Scaling up neural architecture search with big single-stage models". In: *European Conference on Computer Vision*. 2020.

[124] Jiahui Yu et al. "Slimmable Neural Networks". In: *International Conference on Learning Representations*. 2018.

[125] Tianhe Yu et al. "Gradient surgery for multi-task learning". In: *Advances in Neural Information Processing Systems*. 2020.

[126] Geng Yuan et al. "Layer Freezing & Data Sieving: Missing Pieces of a Generic Framework for Sparse Training". In: *arXiv preprint arXiv:2209.11204* (2022).

[127] Geng Yuan et al. "MEST: Accurate and Fast Memory-Economic Sparse Training Framework on the Edge". In: *Advances in Neural Information Processing Systems*. 2021.

[128] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[129] Jure Zbontar et al. "Barlow twins: Self-supervised learning via redundancy reduction". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12310–12320.

[130] Guanxiong Zeng et al. "Continual learning of context-dependent processing in neural networks". In: *Nature Machine Intelligence* 1.8 (2019), pp. 364–372.

[131] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual learning through synaptic intelligence". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3987–3995.

[132] Hongyi Zhang et al. "mixup: Beyond empirical risk minimization". In: *arXiv preprint arXiv:1710.09412* (2017).

[133] Guoqiang Zhong et al. "Reducing and stretching deep convolutional activation features for accurate image classification". In: *Cognitive Computation* 10.1 (2018), pp. 179–186.

[134] Aojun Zhou et al. "Learning N: M Fine-grained Structured Sparse Neural Networks From Scratch". In: *International Conference on Learning Representations*. 2020.

[135] Chenzhuo Zhu et al. "Trained ternary quantization". In: *arXiv preprint arXiv:1612.01064* (2016).

[136] Barret Zoph and Quoc V Le. "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (2016).